On the Undecidability of Asynchronous Session Subtyping

(with appendices)

Julien Lange and Nobuko Yoshida

Imperial College London, UK

Abstract. Asynchronous session subtyping has been studied extensively in [9, 10, 29–32] and applied in [24, 33, 34, 36]. An open question was whether this subtyping relation is decidable. This paper settles the question in the negative. To prove this result, we first introduce a new subclass of two-party communicating finite-state machines (CFSMs), called asynchronous duplex (ADs), which we show to be Turing complete. Secondly, we give a compatibility relation over CFSMs, which is sound and complete wrt. safety for ADs, and is equivalent to the asynchronous subtyping. Then we show that the halting problem reduces to checking whether two CFSMs are in the relation. In addition, we show the compatibility relation to be decidable for three sub-classes of ADs.

1 Introduction

Session types [23, 25, 35] specify the expected interaction patterns of concurrent systems and can be used to automatically determine whether communicating processes interact correctly with other processes. A crucial theory in session types is *subtyping* which makes the typing discipline more flexible and therefore easier to integrate in real programming languages and systems [1]. The first subtyping relations for session types targeted synchronous communications [6, 7, 18, 19], by allowing subtypes to make fewer selections and offer more branches. More recent relations treat asynchronous (buffered) communications [9, 10, 12, 13, 16, 29–32]. They include synchronous subtyping and additionally allow an optimisation by message permutations where outputs can be performed in advance without affecting correctness with respect to the delayed inputs (there are two buffers per session). Only the relative order of outputs (resp. inputs) needs to be preserved to avoid communication mismatches. The asynchronous subtyping is important in parallel and distributed session-based implementations [24, 33, 34, 36], as it reduces message synchronisations without safety violation.

Theoretically, the asynchronous subtyping has been shown to be *precise*, in the sense that: (i) if T is a subtype of U, then a process of type T may be used whenever a process of type U is required and (ii) if T is *not* a subtype of U, then there is a system, requiring a process of type U, for which using a process of type T leads to an error (e.g., deadlock). The subtyping is also denotationally

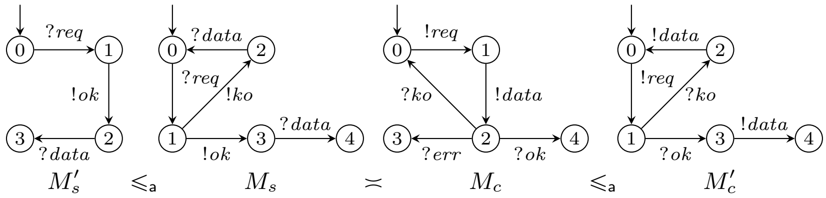


Fig. 1. Asynchronous subtyping and compatibility: examples.

precise taking the standard interpretation of type T as the set of processes typed by T [9, 16].

An open question in [9, 10, 29–32] was whether the asynchronous subtyping relation is decidable, i.e., is there an algorithm to decide whether two types are in the relation. The answer to that question was thought to be positive, see [10, § 7] and § 6.

Asynchronous subtyping, informally. In this work, we consider session types in the form of CFSMs [4], along the lines of [3, 14, 15, 27]. This enables us to characterise the asynchronous subtyping in CFSMs and reduce the undecidability problem to the Turing completeness of CFSMs. Consider a system of CFSMs consisting of machines M_s (server) and M_c (client) in Figure 1, which communicate via two unbounded queues, one in each direction. A transition $!a$ represents the (asynchronous) emission of a message a, while $?a$ represents the receptions of a message a from a buffer. For instance, the transition labelled by $!req$ in M_c says that the client sends a request to the server M_s, later the server can consume this message from its buffer by firing the transition labelled by $?req$. We say that the system (M_s, M_c), i.e., the parallel composition of M_s and M_c, is *safe* if (i) the pair never reaches a deadlock and (ii) whenever a message is sent by one party, it will eventually be received by the other.

The key property of session subtyping is that, e.g., if the system (M_s, M'_c) is safe and M_c is a subtype of M'_c, the system (M_s, M_c) is also safe. We write $\leqslant_{\mathsf{a}}$ for the asynchronous subtyping relation, which intuitively requires that, if, e.g., $M_c \leqslant_{\mathsf{a}} M'_c$, then M_c is ready to receive no fewer messages than M'_c and it may not send more messages than M'_c. For instance, M_c can receive all the messages that M'_c can handle, plus the message err. Observe that M_c is an optimised version of M'_c wrt. asynchrony: the output action $!data$ is performed in advance of the branching. Thus in the system (M_s, M_c), when both machines are in state 2 (respectively), both queues contain messages. Instead, in the system (M_s, M'_c), it is never the case that both queues are non-empty. Note that anticipating the sending of *data* in M_c does not affect safety as it is sent in both branches of M'_c.

Our approach. Using CFSMs, we give the first automata characterisation of asynchronous subtyping and the first proof of its undecidability. To do this, we introduce a new sub-class of CFSMs, called *asynchronous duplex* (AD) which let us study directly the relationship between safety and asynchronous subtyping in CFSMs. Our development consists of the following steps:

Step 1. In § 2, we define a new sub-class of (two-party) CFSMs, called asynchronous duplex (AD), which strictly includes *half-duplex* (HD) systems [8].
Step 2. In § 3, we introduce a compatibility relation ($\asymp$) for CFSMs which is sound and complete wrt. safety in AD CFSMs, i.e., an AD system has no deadlocks nor orphan messages if and only if its machines are $\asymp$-related.
Step 3. Adapting the result of [17], we show in § 4 that AD systems are Turing complete, hence membership of $\asymp$ is generally undecidable.
Step 4. In § 5, we show that the $\asymp$-relation for CFSMs is equivalent to the asynchronous subtyping for session types, thus establishing that the latter is also undecidable.

Throughout the paper, we also show that our approach naturally encompasses the correspondence between synchronous subtyping and safety in HD systems.

In § 4.1, we show that the $\asymp$-relation is decidable for three sub-classes of CFSMs (HD, alternating [21], and non-branching) which are useful to specify real-world protocols. In § 6, we discuss related works and conclude.

2 A new class of CFSMs: Asynchronous duplex systems

This section develops **Step 1** by defining a new sub-class of CFSMs, called *asynchronous duplex*, which characterises machines that can only simultaneously write on their respective channels if they can only do so for finitely many consecutive send actions before executing a receive action. In § 2.1, we recall definitions about CFSMs, then we give the definition of safety. In § 2.2, we introduce the sub-class of AD systems and give a few examples of such systems.

2.1 CFSMs and their properties

Let $\mathbb{A}$ be a (finite) alphabet, ranged over by a, b, etc. We let ω, π, and φ range over words in $\mathbb{A}^*$ and write $\cdot$ for the concatenation operator. The set of actions is $Act = \{!, ?\} \times \mathbb{A}$, ranged over by ℓ, $!a$ represents the emission of a message a, while $?a$ represents the reception of a. We let ψ range over Act^* and define $dir(!a) \overset{\text{def}}{=} !$ and $dir(?a) \overset{\text{def}}{=} ?$.

Since our ultimate goal is to relate CFSMs and session types, we only consider deterministic communicating finite-state machines, without mixed states (i.e., states that can fire both send and receive actions) as in [14, 15].

Definition 2.1 (Communicating machine). *A (communicating) machine M is a tuple (Q, q_0, δ) where Q is the (finite) set of states, $q_0 \in Q$ is the initial state, and $\delta \in Q \times Act \times Q$ is the transition relation such that $\forall q, q', q'' \in Q : \forall \ell, \ell' \in Act :$ (1) $(q, \ell, q'), (q, \ell', q'') \in \delta \implies dir(\ell) = dir(\ell')$, and (2) $(q, \ell, q'), (q, \ell, q'') \in \delta \implies q' = q''$.*

We write $q \xrightarrow{\ell} q'$ for $(q, \ell, q') \in \delta$, omit the label ℓ when unnecessary, and write $\to^*$ for the reflexive transitive closure of $\to$.

Given $M = (Q, q_0, \delta)$, we say that $q \in Q$ is *final*, written $q \not\to$, iff $\forall q' \in Q : \forall \ell \in Act : (q, \ell, q') \notin \delta$. A state $q \in Q$ is *sending* (resp. *receiving*) iff q is not final and $\forall q' \in Q : \forall \ell \in Act : (q, \ell, q') \in \delta : dir(\ell) = !$ (resp. $dir(\ell) = ?$). The dual of M, written $\overline{M}$, is M where each sending transition $(q, !a, q') \in \delta$ is replaced by $(q, ?a, q')$, and vice-versa for receive transitions, e.g., $\overline{M_s} = M'_c$ in Figure 1.

We write $q_0 \xrightarrow{\ell_1 \cdots \ell_k} q_k$ iff there are $q_1, \ldots, q_{k-1} \in Q$ such that $q_{i-1} \xrightarrow{\ell_i} q_i$ for $1 \leqslant i \leqslant k$. Given a list of messages $\omega = a_1 \cdots a_k$ ($k \geqslant 0$), we write $?\omega$ for the list $?a_1 \cdots ?a_k$ and $!\omega$ for $!a_1 \cdots !a_k$. We write $q \xrightarrow{!}{}^* q'$ iff $\exists \omega \in \mathbb{A}^* : q \xrightarrow{!\omega} q'$ and $q \xrightarrow{?}{}^* q'$ iff $\exists \omega \in \mathbb{A}^* : q \xrightarrow{?\omega} q'$ (note that ω may be empty, in which case $q = q'$).

Definition 2.2 (System). *A system $S = (M_1, M_2)$ is a pair of machines $M_i = (Q_i, q_{0_i}, \delta_i)$ with $i \in \{1, 2\}$.*

Hereafter, we fix $S = (M_1, M_2)$ and assume $M_i = (Q_i, q_{0_i}, \delta_i)$ for $i \in \{1, 2\}$ such that $Q_1 \cap Q_2 = \varnothing$. Hence, for $q, q' \in Q_i$, we can write $q \xrightarrow{\ell} q'$ to refer unambiguously to δ_i.

We let λ range over the set $\{{}_{ij}!a \mid i \neq j \in \{1, 2\}\} \cup \{{}_{ij}?a \mid i \neq j \in \{1, 2\}\}$ and ϕ range over (possibly empty) sequences of $\lambda_1 \cdots \lambda_k$.

Definition 2.3 (Reachable configuration). *A configuration of S is a tuple $s = (q_1, \omega_1, q_2, \omega_2)$ such that $q_i \in Q_i$, and $\omega_i \in \mathbb{A}^*$. A configuration $s' = (q'_1, \omega'_1, q'_2, \omega'_2)$ is reachable from $s = (q_1, \omega_1, q_2, \omega_2)$, written $s \overset{\lambda}{\Rightarrow} s'$, iff*

1. *$q_i \xrightarrow{!a} q'_i$, $\omega'_i = \omega_i \cdot a$, $q_j = q'_j$, and $\omega_j = \omega'_j$, $\lambda = {}_{ij}!a$, for $i \neq j \in \{1, 2\}$, or*
2. *$q_i \xrightarrow{?a} q'_i$, $\omega_j = a \cdot \omega'_j$, $q_j = q'_j$, and $\omega_i = \omega'_i$, $\lambda = {}_{ji}?a$, for $i \neq j \in \{1, 2\}$.*

We write $s \Rightarrow s'$ when the label is irrelevant and $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$.

In Definition 2.3, (1) says that machine M_i puts a message on queue i, to be received by machine M_j, while (2) says that machine M_i consumes a message from queue j, which was sent by M_j.

Given a system S, we write s_0 for its initial configuration $(q_{0_1}, \epsilon, q_{0_2}, \epsilon)$ and let $RS(S) \overset{\text{def}}{=} \{s \mid s_0 \Rightarrow^* s\}$.

Definition 2.4 (Safety). *A configuration $s = (q_1, \omega_1, q_2, \omega_2)$ is a* deadlock *iff $\omega_1 = \omega_2 = \epsilon$, q_i is a receiving state, and q_j is either receiving or final for $i \neq j \in \{1, 2\}$. System S satisfies* eventual reception *iff $\forall s = (q_1, \omega_1, q_2, \omega_2) \in RS(S) : \forall i \neq j \in \{1, 2\} : \omega_i \in a \cdot \mathbb{A}^* \implies \forall q'_j \in Q_j : q_j \xrightarrow{!}{}^* q'_j \implies q'_j \xrightarrow{!}{}^* \xrightarrow{?a}$.*

S is safe *iff (i) for all $s \in RS(S)$, s is not a deadlock, and (ii) S satisfies eventual reception (i.e., every sent message is eventually received).*

Lemma 2.1 below shows that safety implies progress and that a configuration with at least one empty buffer is always reachable.

Fig. 2. Examples of AD (left) and non-AD (right) systems.

Lemma 2.1. *If S is* safe*, then for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$*

1. *Either (i) q_1 and q_2 are final and $\omega_1 = \omega_2 = \epsilon$, or (ii) $\exists s' \in RS(S) : s \Rightarrow s'$.*
2. *$\exists s', s'' \in RS(S) : s \Rightarrow^* s' = (q_1, \epsilon, q'_2, \omega_2 \cdot \omega'_2) \wedge s \Rightarrow^* s'' = (q''_1, \omega_1 \cdot \omega''_1, q_2, \epsilon)$.*

2.2 Asynchronous duplex systems

We define asynchronous duplex systems, a sub-class of two-party CFSMs. Below we introduce a predicate which guarantees that when a machine is in a given state, it cannot send infinitely many messages without executing receive actions periodically. This predicate mirrors one of the premises of the defining rules of the asynchronous subtyping ($\leqslant_{\mathsf{a}}$), cf. Lemma 5.1. Given $M = (Q, q_0, \delta)$ and $q \in Q$, we define $\mathtt{fin}(q) \iff \mathtt{fin}(q, \varnothing)$, where

$$\mathtt{fin}(q, R) \stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } q \xrightarrow{?a} \\ \forall q' \in \{q' \mid q \xrightarrow{!a} q'\} : \mathtt{fin}(q', R \cup \{q\}) & \text{if } q \xrightarrow{!a} \wedge q \notin R \\ \mathit{false} & \text{otherwise} \end{cases}$$

Definition 2.5 (Asynchronous duplex). *A system $S = (M_1, M_2)$ is* Asynchronous Duplex *(AD) if for each $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S) : \omega_1 \neq \epsilon \wedge \omega_2 \neq \epsilon \implies \mathtt{fin}(q_1) \wedge \mathtt{fin}(q_2)$.*

AD systems are a strict extension of half-duplex systems [8]: S is half-duplex (HD) if for all $(q_1, \omega_1, q_2, \omega_2) \in RS(S) : \omega_1 = \epsilon \vee \omega_2 = \epsilon$. AD requires that for any reachable configuration either (i) at most one channel is non-empty (i.e., it is a half-duplex configuration) or (ii) each machine is in a state where the predicate $\mathtt{fin}(\_)$ holds, i.e., each machine will reach a receiving state after firing *finitely* many send actions. The AD restriction is reasonable for real-word systems. It intuitively amounts to say that if two parties are *simultaneously* sending data to each other, they should both ensure that they will periodically check what the other party has been sending.

Example 2.1. Consider the machines in Figure 2. The system (M_1, M_2) is AD: $\mathtt{fin}(\_)$ holds for each state in M_1 and M_2. The system $(\hat{M}_1, \hat{M}_2)$ is not AD. For instance, the configuration $(0, a, 0, b)$ is reachable but we have $\neg\mathtt{fin}(0)$ for both initial states of $\hat{M}_1$ and $\hat{M}_2$. Observe that both systems are *safe*, cf. Definition 2.4.

3 A compatibility relation for CFSMs

This section develops **Step 2**: we introduce a binary relation $\asymp$ on CFSMs which is sound and complete wrt. safety (cf. Definition 2.4) for AD systems. That is $M_1 \asymp M_2$ holds if and only if (M_1, M_2) is a safe asynchronous duplex system.

Definition 3.1 (Compatibility). *Let $M_i = (Q_i, q_{0_i}, \delta_i)$ for $i \in \{1, 2\}$ such that $Q_1 \cap Q_2 = \emptyset$, and let $p \in Q_1$, $q \in Q_2$, and $\pi \in \mathbb{A}^*$.*

The compatibility *relation is defined as follows: $\pi \blacktriangleright p \asymp_0 q$ always holds, and if $k \geqslant 0$, then $\pi \blacktriangleright p \asymp_{k+1} q$ holds iff*

1. *if $p \nrightarrow$ then $\pi = \epsilon$ and $q \nrightarrow$*
2. *if $p \xrightarrow{?a}$ then*
   (a) *if $\pi = \epsilon$ then, $q \xrightarrow{!b}$ and $\forall b \in \mathbb{A} : q \xrightarrow{!b} q' \implies (p \xrightarrow{?b} p' \wedge \epsilon \blacktriangleright p' \asymp_k q')$,*
   (b) *if $\pi = b \cdot \pi'$ then, $\exists p' \in Q_1 : p \xrightarrow{?b} p' \wedge \pi' \blacktriangleright p' \asymp_k q$*
3. *if $p \xrightarrow{!a} p'$ then either*
   (a) *$\pi = \epsilon$ and $\exists q' \in Q_2 : q \xrightarrow{?a} q' \wedge \epsilon \blacktriangleright p' \asymp_k q'$, or*
   (b) *$\mathtt{fin}(p)$, $\mathtt{fin}(q)$, and $\forall q' \in Q_2 : \forall \pi' \in \mathbb{A}^* : q \xrightarrow{!\pi'} q'$, there exist $\pi'' \in \mathbb{A}^*$ and $q'' \in Q_2$ such that $q' \xrightarrow{!\pi''}\xrightarrow{?a} q''$ and $\pi \cdot \pi' \cdot \pi'' \blacktriangleright p' \asymp_k q''$*

Define $\pi \blacktriangleright p \asymp q \overset{def}{=} \forall k \in \mathbb{N} : \pi \blacktriangleright p \asymp_k q$ and $M_1 \asymp M_2 \overset{def}{=} \epsilon \blacktriangleright q_{0_1} \asymp q_{0_2}$.

The relation $M_1 \asymp M_2$ checks that the two machines are compatible by executing M_1 while recording what M_2 asynchronously sends to M_1 in the π message list. The definition first differentiates the type of state p:

Final. Case (1) says that if M_1 is in a final state, then M_2 must also be in a final state and π must be empty (i.e., M_1 has emptied its input buffer).

Receiving. Case (2) says that if M_1 is in a receiving state, then either π is empty and M_1 must be ready to receive any message sent by M_2, cf. case (2a); otherwise, case (2b) must apply: M_1 must consume the head of the message list π, this models the FIFO consumption of messages sent by M_2.

Sending. Case (3) says that if M_1 is ready to send a, then either M_2 must be able to receive a directly, cf. case (3a). Otherwise, $\mathtt{fin}(p)$ and $\mathtt{fin}(q)$ must hold so that case (3b) applies. M_2 may delay the reception of a by sending messages (which are recorded in $\pi' \cdot \pi''$). Whichever sending path M_2 chooses, it must always eventually receive a.

We write $\asymp_s$ for the *synchronous compatibility relation*, i.e., Definition 3.1 without cases (2b) and (3b).

Example 3.1. **(1)** Recall the machines from Figure 1, we have $M_s \asymp M_c$, in particular: $\epsilon \blacktriangleright 0 \asymp 0$ and $data \blacktriangleright 2 \asymp 0$. The latter relation represents the fact that M_c and M_s have exchanged the messages *req* and *ko*, but M_s has yet to process the reception of *data*. Observe that we also have $M_s' \asymp M_c'$ and $M_s' \asymp_s M_c'$.
(2) Consider the systems in Figure 2. We have $M_1 \asymp M_2$ and $\hat{M}_1 \not\asymp \hat{M}_2$. The latter does not hold since both initial states are sending states, but the predicate $\mathtt{fin}(\_)$ does not hold for either state, e.g., we have $\neg\mathtt{fin}(0, \{0\})$ in $\hat{M}_1$.

Soundness of $\asymp$. We show the soundness of the $\asymp$-relation wrt. safety. More precisely we show that if $M_1 \asymp M_2$ holds, then the system (M_1, M_2) is a safe AD system. We first give two auxiliary definitions which are convenient to relate safety with the definition of $\asymp$. Fixing $M = (Q, q_0, \delta)$, the predicate $A(q, \omega)$ asserts when a list of messages ω is "accepted" from a state $q \in Q$, which implies eventual reception of the messages in ω. The function $W(q, \omega)$ is used to connect a configuration to a triple in the $\asymp$-relation.

Definition 3.2. *Let $q \in Q$ and $\omega \in \mathbb{A}^*$, we define*

$$A(q,\omega) \iff \begin{cases} \forall q' : q \xrightarrow{!}{}^* q' : \exists \hat{q} : q' \xrightarrow{!}{}^* \xrightarrow{?a} \hat{q} \wedge A(\hat{q}, \omega') & \text{if } \omega = a \cdot \omega' \\ \mathit{true} & \text{if } \omega = \epsilon \end{cases}$$

Given $q \in Q$ and $\omega \in \mathbb{A}^*$, the predicate $A(q, \omega)$ is true iff the list of messages ω can always be consumed entirely from state q, for all paths reachable from q by send actions. Note the similarity with case (3b) of Definition 3.1.

Definition 3.3. *Let $q \in Q$ and $\omega \in \mathbb{A}^*$, $W(q, \omega) \subseteq \mathbb{A}^* \times Q$ is the set such that*

$$(\pi, \hat{q}) \in W(q,\omega) \iff \begin{cases} (\varphi, \hat{q}) \in W(q', \omega') & \text{if } \omega = a \cdot \omega', q \xrightarrow{!\pi' \cdot ?a} q', \pi = \pi' \cdot \varphi \\ \pi = \epsilon \wedge \hat{q} = q & \text{if } \omega = \epsilon \end{cases}$$

Each pair $(\pi, \hat{q})$ in $W(q, \omega)$ represents a state $\hat{q} \in Q$ reachable directly after having consumed the list of messages ω, while π is the list of messages that are *sent* along a path between q and $\hat{q}$. For example, consider M_c from Figure 1. We have $A(0, ko \cdot ko \cdot err)$ and $W(0, ko \cdot ko \cdot err) = \{(req \cdot data \cdot req \cdot data, 3)\}$; instead, $\neg A(0, ok \cdot ko)$ and $\neg A(4, ko)$.

Lemma 3.1. *Let $M = (Q, q_0, \delta)$, $q \in Q$ and $\omega \in \mathbb{A}^*$. If $A(q, \omega)$ and $\forall (\varphi, q') \in W(q, \omega) : A(q', a)$ then $A(q, \omega \cdot a)$.*

Lemma 3.1, shown by induction on the size of ω, is useful in the proof of the main soundness lemma below.

Lemma 3.2. *Let $S = (M_1, M_2)$. If $M_1 \asymp M_2$, then for all $s = (p, \omega_1, q, \omega_2) \in RS(S)$ the following holds:* (1) *s is not a deadlock,* (2) $A(q, \omega_1)$, (3) $\forall (\varphi, q') \in W(q, \omega_1) : \omega_2 \cdot \varphi \blacktriangleright p \asymp q'$, *and* (4) $A(p, \omega_2)$.

Lemma 3.2 states that for any configuration s: (1) s is not a deadlock; (2) M_2 can consume the list ω_1 from state q; (3) for each state q', reached after consuming ω_1, the relation $\omega_2 \cdot \varphi \blacktriangleright p \asymp q'$ holds, where φ contains the messages that M_2 sent while consuming ω_1; and (4) M_1 can consume the list ω_2 from state p. The proof of Lemma 3.2 is by induction on the length of an execution from s_0 to s, then by case analysis on the last action fired to reach s. Lemma 3.1 is used for the case $s_0 \Rightarrow^* \xRightarrow{12!a} s$, i.e., to show that $A(q, \omega_1 \cdot a)$ holds.

Lemma 3.3. *Let $S = (M_1, M_2)$. If for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$: $A(q_1, \omega_1)$ and $A(q_2, \omega_2)$, then S satisfies* eventual reception.

Lemma 3.3 simply shows a correspondence between eventual reception and Definition 3.2. The proof essentially shows that if $A(q_i, \omega_j)$ holds, then we can always reach a configuration where the list ω_j has been entirely consumed.

Finally, we state our final soundness results. Theorem 3.1 is a consequence of Lemmas 2.1, 3.2, 3.3, and 3.4. Theorem 3.2 essentially follows from Theorem 3.1 and the fact that $\asymp_s \subseteq \asymp$.

Theorem 3.1. *If* $M_1 \asymp M_2$*, then* (M_1, M_2) *is a* safe AD *system.*

Theorem 3.2. *If* $M_1 \asymp_s M_2$*, then* (M_1, M_2) *is a* safe HD *system.*

Completeness of $\asymp$. Our completeness result shows that for any safe asynchronous duplex system $S = (M_1, M_2)$, $M_1 \asymp M_2$ holds. Below we show that any reachable configuration of S whose first queue is empty can be mapped to a triple that is in the relation of Definition 3.1.

Lemma 3.4. *Let* S *be* safe *and* AD*, then* $\forall (p, \epsilon, q, \omega) \in RS(S) : \omega \blacktriangleright p \asymp q$*.*

The proof of Lemma 3.4 is by induction on the k^{th} approximation of $\asymp$, i.e., assuming that $\omega \blacktriangleright p \asymp_k q$ holds, we show that $\omega \blacktriangleright p \asymp_{k+1} q$ holds. The proof is a rather straightforward case analysis on the type of p and whether or not $\omega = \epsilon$.

Theorem 3.3. *If* (M_1, M_2) *is a* safe AD *system, then* $M_1 \asymp M_2$*.*

Proof. Take $(q_{0_1}, \epsilon, q_{0_2}, \epsilon) \in RS(S)$, $\epsilon \blacktriangleright q_{0_1} \asymp q_{0_2}$ holds by Lemma 3.4. ◻

Following a similar (but simpler) argument, we have Theorem 3.4 below.

Theorem 3.4. *If* (M_1, M_2) *is a* safe HD *system, then* $M_1 \asymp_s M_2$*.*

Theorem 3.5. *If* $M_1 \asymp M_2$ *(resp.* $M_1 \asymp_s M_2$*), then* $M_2 \asymp M_1$ *(resp.* $M_2 \asymp_s M_1$*).*

Proof. We show the $\asymp$ part. By Theorem 3.1, (M_1, M_2) is safe, hence by definition of safety, (M_2, M_1) is also safe. Thus by Theorem 3.3, we have $M_2 \asymp M_1$. ◻

4 Undecidability of the $\asymp$-relation

This section addresses **Step 3**: we show that the problem of checking $M_1 \asymp M_2$ is undecidable. We show that AD systems are Turing complete, then show that the halting problem reduces to deciding whether or not a system is safe.

Preliminaries. We adapt the relevant part of the proof of Finkel and McKenzie [17] to demonstrate that the problem of deciding whether two machines are $\asymp$-related is undecidable. For this we need to show that there is indeed a Turing machine encoding that is an AD system.

Definition 4.1 (Turing machine [17]). *A Turing machine (T.M.) is a tuple* $TM = (V, \mathbb{A}, \Gamma, t_0, \mathtt{B}, \gamma)$ *where* V *is the set of states,* $\mathbb{A}$ *is the input alphabet,* Γ *is the tape alphabet,* $t_0 \in V$ *is the initial state,* $\mathtt{B}$ *is the blank symbol, and* $\gamma : V \times \Gamma \to V \times \Gamma \times \{\mathit{left}, \mathit{right}\}$ *is the (partial) transition function.*

Assume TM accepts an input $\omega \in \mathbb{A}^*$ iff TM halts on ω, and if TM does not halt on ω, then TM eventually moves its tape head arbitrarily far to the right.

Definition 4.2 (Configuration of a T.M. [17]). *A configuration of the Turing machine TM is a word $\omega_1 t \omega_2 \#$ with $\omega_1\omega_2 \in \mathbb{A}^*$, $t \in V$, and $\# \notin \Gamma$.*

The word $\omega_1 t \omega_2 \#$ represents TM in state $t \in V$ with the tape content set to $\omega_1\omega_2$ and the rest blank, and TM's head positioned under the first symbol to the right of ω_1. Symbol # is a symbol used to mark the end of the tape.

T.M. encoding. We present an AD system which encodes a Turing machine $TM = (V, \mathbb{A}, \Gamma, t_0, \mathtt{B}, \gamma)$ with initial tape ω into a system of two CFSMs as in [17].

We explain the T.M. encoding. The two channels represent the tape of the Turing machine, with a marker # separating the two ends of the tape. Each machine represents the control of the Turing machine as well as a transmitter from a channel to another. The head is represented by writing the current control state $t \in V$ on the channel. Whenever a machine receives a message that is $t \in V$, then it proceeds with one execution step of the Turing machine. Any other symbol is simply consumed from one channel and sent on the other.

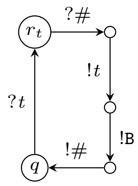
The only difference wrt. [17] is that we construct machines which are deterministic and which do not contain mixed states, cf. Definition 2.1. We also do not require the machines to be identical hence we encode the initial tape content as a sequence of transitions in the first machine. These slight modifications do not affect the rest of Finkel and McKenzie's proof in [17]. The system consists of two CFSMs $A_i = (Q_i, q_{0_i}, \delta_i)$, $i \in \{1, 2\}$ over the alphabet $\mathbb{A} \cup \{\#\}$. The definitions of δ_i are given below, the sets Q_i are induced by δ_i. The transition relation δ_1 consists in a sequence of transitions from the initial state q_{0_1} to a central state q and a number of elementary cycles around state q, cf. Figure 3; while δ_2 is like δ_1 without the initial sequence of transitions and $q = q_{0_2}$. The initial sequence of transitions in δ_1 is of the form:

$$q_{0_1} \xrightarrow{!t_0} q_1 \xrightarrow{!a_1} \cdots q_k \xrightarrow{!a_k} q \qquad \text{such that } a_1 \cdots a_k = \omega \cdot \#$$

Both δ_1 and δ_2 contain six types of elementary cycles given in Figure 3. For each type of cycle, we illustrate the behaviour of the system from the point view of machine A_2 by giving the type of configuration this cycle applies to as well the configuration obtained after A_2 has finished executing the cycle.

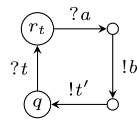
When computing each δ_i and Q_i from the description above, we assume that each "anonymous" state maintain its own identity, while "named" states, i.e., q, r_t, r_x and r_x^t from Figure 3, are to be identified and redundant transitions to be removed. This ensures that each machine so obtained is deterministic. Besides this determinisation, the only changes from [17] concerns the copying cycles. (1) Each copying cycle is expanded to receive (then send) two symbols so to ensure the absence of mixed states once merged with left head motion cycles. (2) We add a cycle which only re-emits # symbols (to make up for absence of it in the first reception of the copying cycles). (3) We add another blank insertion cycle to deal with the special case where the head is followed by the # symbol.

Blank insertion cycles (1). For each $t \in V$, there is a cycle of the form:



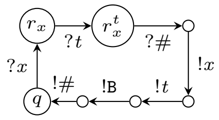
$$(q', t \cdot \# \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot t \cdot \texttt{B} \cdot \#)$$

Right head motion cycles. For each $(t, a, t', b) \in V \times \Gamma \times V \times \Gamma$ such that $\gamma(t, a) = (t', b, \mathit{right})$, there is a cycle of the form:



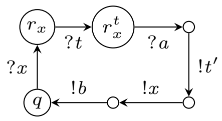
$$(q', t \cdot a \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot b \cdot t')$$

Blank insertion cycles (2). For each $x \in \Gamma$ and $t \in V$ there is a cycle of the form:



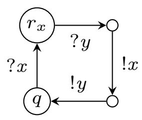
$$(q', x \cdot t \cdot \# \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot x \cdot t \cdot \texttt{B} \cdot \#)$$

Left head motion cycles. For each $(x, t, a, t', b) \in \Gamma \times V \times \Gamma \times V \times \Gamma$ such that $\gamma(t, a) = (t', b, \mathit{left})$, there is a cycle of the form:



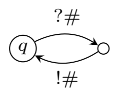
$$(q', x \cdot t \cdot a \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot t' \cdot x \cdot b)$$

Copying cycles. For all $x \in \Gamma$ and $y \in \Gamma \cup \{\#\}$, there is a cycle of the form:



$$(q', x \cdot y \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot x \cdot y)$$

Marker transmission cycle. There is one cycle specified by:



$$(q', \# \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot \#)$$

Fig. 3. Definition of δ_i (elementary cycles).

Definition 4.3 (Turing machine encoding [17]). *Given a Turing machine TM and an initial tape content ω, we write $S(TM, \omega)$ for the system (A_1, A_2) with each A_i constructed as described above.*

The rest follows the proof of [17]. Here we recall informally the final result: any execution of a Turing machine TM with initial word ω can be simulated by $S(TM, \omega)$, and vice-versa.

Lemma 4.1. *For any TM and word ω, $S(TM, \omega) = (A_1, A_2)$ is* AD.

Proof. Take $A_i = (Q_i, q_{0_i}, \delta_i)$, we show $\forall q \in Q_i : \texttt{fin}(q)$, which implies that the system is AD. If there was $q \in Q_i$ such that $\neg\texttt{fin}(q)$, there would a cycle of send actions only, the construction of A_i clearly prevents this (see Figure 3). □

Theorem 4.1 (Undecidability of $\asymp$). *Given two machines M_1 and M_2, it is generally undecidable whether $M_1 \asymp M_2$ holds.*

The proof of Theorem 4.1 shows that the following statements are equivalent: (1) TM accepts ω, (2) $S(TM, \omega) = (A_1, A_2)$ is *not* safe, and (3) $\neg(A_1 \asymp A_2)$. We show (1) $\Rightarrow$ (2) by Lemma 2.1, (2) $\Rightarrow$ (1) from the definition of safety, and (2) $\Leftrightarrow$ (3) by Theorems 3.1 and 3.3 and the fact that (A_1, A_2) is AD.

4.1 Decidable sub-classes of CFSMs

We now identify three sub-classes of CFSMs for which the $\asymp$-relation is decidable. We say that $M_1 \asymp M_2$ is decidable iff it is decidable whether or not $M_1 \asymp M_2$ holds. The first sub-class is HD systems: HD is a decidable property and safety is decidable within that class [8], hence $\asymp$ is decidable in HD and it is equivalent to $\asymp_s$ within HD. The second sub-class is taken from the CFSMs literature and the third is limited to systems that contain at least one machine that has no branching. We define the last two sub-classes below.

The following definition is convenient to formalise our decidability results. Given $M_i = (Q_i, q_{0_i}, \delta_i)$ for $i \in \{1, 2\}$, the *derivation tree* of a triple $\pi \blacktriangleright p \asymp q$ is a tree whose nodes are labelled by elements of $\mathbb{A}^* \times Q_1 \times Q_2$ such that the children of a node are exactly the triple generated by applying one step of Definition 3.1.

For example, consider the machines M_1 and M_2 from Figure 2, we have a tree which consists of a unique (infinite) branch:
$\epsilon \blacktriangleright 0 \asymp 0 \longrightarrow b \blacktriangleright 1 \asymp 0 \longrightarrow bb \blacktriangleright 2 \asymp 0 \longrightarrow b \blacktriangleright 0 \asymp 0 \longrightarrow bb \blacktriangleright 1 \asymp 0 \longrightarrow bbb \blacktriangleright 2 \asymp 0 \cdots$

Lemma 4.2. *The derivation tree of $\pi \blacktriangleright p \asymp q$ is finitely branching.*

Lemma 4.2 follows from the fact that each machine is finitely branching and the predicate `fin(_)` guarantees finiteness for case (3b) of Definition 3.1.

Alternating machines. Alternating machines were introduced in [21] where it is shown that the progress problem (corresponding to our notion of safety) is decidable for such systems. A machine is *alternating* if each of its sending transition is followed by a receiving transition, e.g., M_s and M'_s in Figure 1 are alternating, as well as the specification of the alternating-bit protocol in [21]. Observe that alternating machines form AD systems.

Theorem 4.2. *If M_1 and M_2 are* alternating*, then $M_1 \asymp M_2$ is decidable.*

The proof simply shows that the π part of the relation (cf. Definition 3.1) is bounded by 1, by induction on the depth of the derivation tree.

Non-branching machines. Given $M = (Q, q_0, \delta)$ we say that M is non-branching if each of its state has at most one successor, i.e., if $\forall q \in Q : |\delta(q)| \leqslant 1$. For example, M'_s in Figure 1 is non-branching. Non-branching machines are used notably in [34, 36] to specify parallel programs which can be optimised through asynchronous message permutations.

Theorem 4.3. *Let M_1 and M_2 be two machines such that at least one of them is* non-branching*, then $M_1 \asymp M_2$ is decidable.*

The proof relies on the fact that (i) the derivation tree is finitely branching (Lemma 4.2), hence there is a semi-algorithm to checker whether $\neg(M_1 \asymp M_2)$ and (ii) over any infinite branches we can find two nodes of the form $\mathtt{c} = \pi^n \blacktriangleright p \asymp q$ and $\mathtt{c}' = \pi^m \blacktriangleright p \asymp q$, with $n \leqslant m$. If n is large enough, this implies that the relation holds (i.e., the branch is indeed infinite).

5 Correspondence between compatibility and subtyping

We show a precise correspondence between the asynchronous subtyping for session types and the $\asymp$-relation for CFSMs, i.e., **Step 4**. We first recall the syntax of session types and as well as the definition of asynchronous subtyping.

Session types and subtyping. The syntax of session types is given by

$$T, U \coloneqq \quad \mathtt{end} \quad | \quad \oplus_{i\in I} ! a_i . T_i \quad | \quad \&_{i\in I} ? a_i . T_i \quad | \quad \mathtt{rec}\, \mathtt{x}.T \quad | \quad \mathtt{x}$$

where $I \neq \varnothing$ is finite and $a_i \neq a_j$ for $i \neq j$. Type $\mathtt{end}$ indicates the end of a session. Type $\oplus_{i\in I} ! a_i . T_i$ specifies an *internal* choice, indicating that the program chooses to send one of the a_i messages, then behaves as T_i. Type $\&_{i\in I} ? a_i . T_i$ specifies an *external* choice, saying that the program waits to receive one of the a_i messages, then behaves as T_i. Types $\mathtt{rec}\, \mathtt{x}.T$ and $\mathtt{x}$ are used to specify recursive behaviours. We only consider closed types, i.e., without free variables.

Since our goal is to relate a binary relation defined on CFSMs to a binary relation on session types, we first introduce transformations from one to another.

Definition 5.1. *Given a type T, we write $\mathcal{M}(T)$ for the CFSM induced by T. Given a CFSM M, we write $\mathcal{T}(M)$ for the type constructed from M.*

We assume the existence of two algorithms such that $T = \mathcal{T}(\mathcal{M}(T))$ and $M = \mathcal{M}(\mathcal{T}(M))$ for any type T and machine M. These algorithms are rather trivial since each session type induces a finite automaton, see [15] for instance.

We write $\overline{T}$ for the *dual* of type T, i.e., $\overline{\mathtt{end}} = \mathtt{end}$, $\overline{\mathtt{x}} = \mathtt{x}$, $\overline{\mathtt{rec}\, \mathtt{x}.T} = \mathtt{rec}\, \mathtt{x}.\overline{T}$, $\overline{\oplus_{i\in I} ! a_i . T_i} = \&_{i\in I} ? a_i . \overline{T}_i$, and $\overline{\&_{i\in I} ? a_i . T_i} = \oplus_{i\in I} ! a_i . \overline{T}_i$.

Hereafter, we write $\leqslant_{\mathsf{a}}$ for the relation in [9] (abstracting away from carried types) which we recall below. An *asynchronous context* [9] is defined by

$$\mathcal{A} \coloneqq [\,]^n \;\mid\; \&_{i\in I} ? a_i . \mathcal{A}_i$$

We write $\mathcal{A}[\,]^{n\in N}$ to denote a context with holes indexed by elements of N and $\mathcal{A}[T_n]^{n\in N}$ to denote the same context when the hole $[\,]^n$ has been filled with T_n.

The predicate $\& \in T$ holds if it can be derived from the following rules:

$$\frac{}{\& \in \&_{i\in I} ? a_i . T_i} \qquad \frac{\forall i \in I \,:\, \& \in T_i}{\& \in \oplus_{i\in I} ! a_i . T_i} \qquad \frac{\& \in T}{\& \in \mathtt{rec}\, \mathtt{x}.T}$$

$\& \in T$ holds whenever T always eventually performs a receive action, i.e., it cannot loop on send actions only. It is the counterpart of the predicate $\mathtt{fin}(\_)$ for CFSMs, cf. Lemma 5.1.

Definition 5.2 ($\leqslant_\mathsf{a}$ [9])**.** *The asynchronous subtyping,* $\leqslant_\mathsf{a}$*, is the* largest *relation that contains the rules:*[1]

$$\dfrac{\forall i \in I \,:\, T_i \leqslant_\mathsf{a} U_i}{\bigoplus_{i\in I} !a_i.\, T_i \leqslant_\mathsf{a} \bigoplus_{i\in I\cup J} !a_i.\, U_i}\ [\textsc{sel}] \qquad \dfrac{\forall i \in I \,:\, T_i \leqslant_\mathsf{a} U_i}{\&_{i\in I\cup J} ?a_i.\, T_i \leqslant_\mathsf{a} \&_{i\in I} ?a_i.\, U_i}\ [\textsc{bra}]$$

$$\dfrac{\forall i \in I \,:\, T_i \leqslant_\mathsf{a} \mathcal{A}[U_i^n]^{n\in N} \quad \& \in T_i}{\bigoplus_{i\in I} !a_i.\, T_i \leqslant_\mathsf{a} \mathcal{A}[\bigoplus_{i\in I\cup J_n} !a_i.\, U_i^n]^{n\in N}}\ [\textsc{async}] \qquad \dfrac{}{\mathtt{end} \leqslant_\mathsf{a} \mathtt{end}}\ [\textsc{end}]$$

The double line in the rules indicates that the rules should be interpreted coinductively*. We are assuming an equi-recursive view of types.*

Rule [sel] lets the subtype make fewer selections than its supertype, while rule [bra] allows the subtype to offer more branches. Rule [async] allows safe permutations of actions, by which a protocol can be refined to maximise asynchrony without violating safety. Note that the *synchronous* subtyping $\leqslant_\mathsf{s}$ [11, 19, 20] is defined as Definition 5.2 without rule [async], hence $\leqslant_\mathsf{s} \subseteq \leqslant_\mathsf{a}$. In Figure 1, $\mathcal{T}(M'_s) \leqslant_\mathsf{s} \mathcal{T}(M_s)$, $\mathcal{T}(M'_s) \leqslant_\mathsf{a} \mathcal{T}(M_s)$, and $\mathcal{T}(M_c) \leqslant_\mathsf{a} \mathcal{T}(M'_c)$.

The correspondence between $\asymp$ (Definition 3.1) and $\leqslant_\mathsf{a}$ (Definition 5.2) can be understood as follows. Case (1) of Definition 3.1 corresponds to rule [end]. Case (2a) corresponds to rule [bra]. Case (3a) corresponds to rule [sel]. Cases (2b) and (3b) together correspond to rule [async].

Correspondences. We show that $\asymp$ on CFSMs and $\leqslant_\mathsf{a}$ on session types are equivalent, and, as a consequence, deciding whether two types are $\leqslant_\mathsf{a}$-related is undecidable. We first introduce a few auxiliary lemmas and definitions.

Lemma 5.1. *Let* $M = (Q, q_0, \delta)$ *and* T *be a session type.*

1. *For each* $q \in Q$*, if* $\mathtt{fin}(q)$*, then* $\& \in \mathcal{T}(Q, q, \delta)$*.*
2. *If* $\& \in T$ *and* $\mathcal{M}(T) = (\hat{Q}, q, \hat{\delta})$*, then* $\mathtt{fin}(q)$*.*
3. *If* $T = \overline{\mathcal{A}[\bigoplus_{i\in I} !a_i.\, U_i^n]^{n\in N}}$ *then* $\& \in T$*.*

Lemma 5.1 states the relationship between $\& \in T$ and $\mathtt{fin}(\_)$ (cf. § 2.2).

We write $\pi \in \mathcal{A}$ if π is a branch in the context $\mathcal{A}$. Formally, given $\mathcal{A}$ and $\pi \in \mathbb{A}^*$, we define the predicate $\pi \in \mathcal{A}$ as follows:

$$\pi \in \mathcal{A} \iff \begin{cases} \pi = \epsilon & \text{if } \mathcal{A} = [\,] \\ \pi = a_j \cdot \pi_j & \text{if } \mathcal{A} = \&_{i\in I} ?a_i.\, \mathcal{A}_i, \pi_j \in \mathcal{A}_j, \text{ with } j \in I \end{cases}$$

The next lemma shows that the $\leqslant_\mathsf{a}$-relation implies the $\asymp$-relation.

Lemma 5.2. *Let* T *and* U *be two session types, such that* $\mathcal{M}(T) = (Q^T, q_0^T, \delta^T)$ *and* $\mathcal{M}(U) = (Q^U, q_0^U, \delta^U)$*, then* $T \leqslant_\mathsf{a} \mathcal{A}[\overline{U}] \implies \forall \pi \in \mathcal{A} \,:\, \pi \blacktriangleright q_0^T \asymp q_0^U$*.*

[1] Note that in [9] rule [async] has a redundant additional premise, $\& \in \mathcal{A}$, which is only used to make the application of the rules deterministic.

The proof of Lemma 5.2 is by coinduction on the derivation of $\pi \blacktriangleright p \asymp q$. We use Lemma 5.1 to show that premise of rule [ASYNC] implies that $\mathtt{fin}(q_0^T)$ and $\mathtt{fin}(q_0^U)$ hold when necessary.

The next lemma shows that the $\asymp$-relation implies the $\leqslant_{\mathsf{a}}$-relation.

Lemma 5.3. *Let $M_i = (Q_i, q_{0_i}, \delta_i)$, $i \in \{1,2\}$ and $\pi = a_1 \cdots a_k \in \mathbb{A}^*$, for all $p \in Q_1$ and $q \in Q_2$, $\pi \blacktriangleright p \asymp q \implies \mathcal{T}(Q_1, p, \delta_1) \leqslant_{\mathsf{a}} \overline{?a_1 \cdots ?a_k.[\mathcal{T}(Q_2, q, \delta_2)]}$.*

The proof of Lemma 5.3 is by coinduction on the rules of Definition 5.2, using Lemma 5.1 to match the requirements of the respective relations.

We are now ready to state the final equivalence result.

Theorem 5.1. *The relations $\asymp$ and $\leqslant_{\mathsf{a}}$ are equivalent in the following sense:*

1. *Let T_1 and T_2 be two session types, then $T_1 \leqslant_{\mathsf{a}} \overline{T_2} \implies \mathcal{M}(T_1) \asymp \mathcal{M}(T_2)$.*
2. *Let M_1 and M_2 be two machines, then $M_1 \asymp M_2 \implies \mathcal{T}(M_1) \leqslant_{\mathsf{a}} \overline{\mathcal{T}(M_2)}$.*

Proof. (1) follows from Lemma 5.2, with $T_1 = T$, $T_2 = U$, and $\mathcal{A} = [\,]$. (2) follows from Lemma 5.3, with $\pi = \epsilon$, $p = q_{0_1}$, and $q = q_{0_2}$. □

A consequence of the correspondence between the two relations is that the $\asymp$-relation is transitive in the following sense:

Theorem 5.2. *If $M_1 \asymp \overline{M}$ and $M \asymp M_2$, then $M_1 \asymp M_2$.*

Proof. By Theorem 5.1 we have (1) $M_1 \asymp \overline{M} \iff M_1 \leqslant_{\mathsf{a}} M$ (2) $M \asymp M_2 \iff M \leqslant_{\mathsf{a}} \overline{M_2}$. Since $\leqslant_{\mathsf{a}}$ is transitive [10], we have $M_1 \leqslant_{\mathsf{a}} \overline{M_2}$. Thus, using Theorem 5.1 again, we have $M_1 \leqslant_{\mathsf{a}} \overline{M_2} \iff M_1 \asymp M_2$. □

As a consequence of Theorem 4.1 and Theorem 5.1, we have the undecidability of the asynchronous subtyping:

Theorem 5.3 (Undecidability of $\leqslant_{\mathsf{a}}$). *Given two session types T_1 and T_2, it is generally undecidable whether $T_1 \leqslant_{\mathsf{a}} T_2$ holds.*

We state the equivalence between $\asymp_s$ and $\leqslant_{\mathsf{s}}$, and the transitivity of $\asymp_s$.

Theorem 5.4. *The relations $\asymp_s$ and $\leqslant_{\mathsf{s}}$ are equivalent in the following sense:*

1. *Let T_1 and T_2 be two session types, then $T_1 \leqslant_{\mathsf{s}} \overline{T_2} \implies \mathcal{M}(T_1) \asymp_s \mathcal{M}(T_2)$.*
2. *Let M_1 and M_2 be two machines, then $M_1 \asymp_s M_2 \implies \mathcal{T}(M_1) \leqslant_{\mathsf{s}} \overline{\mathcal{T}(M_2)}$.*

Theorem 5.5. *If $M_1 \asymp_s \overline{M}$ and $M \asymp_s M_2$, then $M_1 \asymp_s M_2$.*

Theorem 5.1 together with the soundness and completeness of $\asymp$ wrt. safety in AD systems (Theorems 3.1 and 3.3) imply a tight relationship between $\leqslant_{\mathsf{a}}$ and session types corresponding to AD systems. A similar correspondence between $\leqslant_{\mathsf{s}}$ and HD systems exists, by Theorems 3.2, 3.4, and 5.4.

6 Conclusions and related work

We have introduced a new sub-class of CFSMs (AD), which includes HD, and a compatibility relation $\asymp$ (resp. $\asymp_s$) that is sound and complete wrt. safety within AD (resp. HD) and equivalent to asynchronous (resp. synchronous) subtyping. Our results in § 4.1 suggest that $\asymp$ is a convenient basis for designing safety checking algorithms for some sub-classes of CFSMs. Given the results in the present paper, we plan to study bounded approximations of $\asymp$ that can be used for session typed applications. Such approximations would make asynchronous subtyping available for real-world programs and thus facilitate the integration of flexible session typing.

Related work. The first (synchronous) subtyping for session types in the π-calculus was introduced in [19] and shown to be decidable in [20]. Its complexity was further studied in [28] which encodes synchronous subtyping as a model checking problem. The first version of asynchronous subtyping was introduced in [32] for multiparty session types and further studied in [29–31] for binary session types in the higher-order π-calculus. These works and [10] stated or conjectured the decidability of the relations. The technical report [5] (announced after the submission of the present paper) independently studied the undecidability of these relations. Note that the subtyping relation in [29,31] only differs from the one in [9,10] by the omission of the premise $\& \in T_i$ in rule [ASYNC]. This subtyping is not sound wrt. our definition of safety as it does not guarantee eventual reception [9,10]. We conjecture that it is sound and complete wrt. progress (either both machines are in a final state or one can eventually make a move) in (the full class of) CFSMs (Definition 2.1), hence it is also undecidable since progress corresponds to rejection of a word by a Turing machine, cf. § 4.

The operational and denotational preciseness of (synchronous and asynchronous) subtyping for session types was studied in [9,10] where the authors give soundness and completeness of each subtyping through the set of π-calculus processes which can be assigned a given type. In this paper, we study the soundness and completeness of $\asymp$ (resp. $\asymp_s$) in CFSMs through AD (resp. HD) systems.

CFSMs have long been known to be Turing complete [4,17] even when restricted to deterministic machines without mixed states [21]. The first paper to relate formally CFSMs and session types was [14], which was followed by a series of work using CFSMs as session types [3,15,27]. The article [2] shows, in a similar fashion to [17], that the compliance of contracts based on asynchronous session types is undecidable. Here, we show that the encoding of [17] is indeed AD and that safety is equivalent to word acceptance by a Turing machine.

Acknowledgements. We thank A. Scalas, B. Toninho and G. Zavattaro for their comments on earlier versions of this paper, in particular G. Zavattaro for identifying the need for additional blank insertion cycles (in Fig. 3). This work is partially supported by EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1; and by EU FP7 612985 (UPSCALE).

References

1. D. Ancona, V. Bono, M. Bravetti, J. Campos, G. Castagna, P. Deniélou, S. J. Gay, N. Gesbert, E. Giachino, R. Hu, E. B. Johnsen, F. Martins, V. Mascardi, F. Montesi, R. Neykova, N. Ng, L. Padovani, V. T. Vasconcelos, and N. Yoshida. Behavioral types in programming languages. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016.
2. M. Bartoletti, A. Scalas, E. Tuosto, and R. Zunino. Honesty by Typing. *Logical Methods in Computer Science*, Volume 12, Issue 4, Dec. 2016.
3. L. Bocchi, J. Lange, and N. Yoshida. Meeting deadlines together. In *CONCUR 2015*, pages 283–296, 2015.
4. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
5. M. Bravetti, M. Carbone, and G. Zavattaro. Undecidability of asynchronous session subtyping. *CoRR*, abs/1611.05026, 2016.
6. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *ESOP 2007*, pages 2–17, 2007.
7. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8, 2012.
8. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
9. T.-C. Chen, M. Dezani-Ciancaglini, A. Scalas, and N. Yoshida. On the preciseness of subtyping in session types. *LMCS*, 2016. to appear.
10. T.-C. Chen, M. Dezani-Ciancaglini, and N. Yoshida. On the preciseness of subtyping in session types. In *PPDP 2014*, pages 146–135. ACM Press, 2014.
11. R. Demangeon and K. Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR 2011*, pages 280–296, 2011.
12. R. Demangeon and N. Yoshida. On the expressiveness of multiparty sessions. In P. Harsha and G. Ramalingam, editors, *FSTTCS 2015*, volume 45 of *LIPIcs*, pages 560–574. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
13. P. Deniélou and N. Yoshida. Buffered communication analysis in distributed multiparty sessions. In *CONCUR 2010*, pages 343–357, 2010.
14. P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP 2012*, pages 194–213, 2012.
15. P. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP 2013*, pages 174–186, 2013.
16. M. Dezani-Ciancaglini, S. Ghilezan, S. Jaksic, J. Pantovic, and N. Yoshida. Denotational and operational preciseness of subtyping: A roadmap. In *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, pages 155–172, 2016.
17. A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theor. Comput. Sci.*, 174(1-2):217–230, 1997.
18. S. J. Gay. Subtyping supports safe session substitution. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, pages 95–108, 2016.
19. S. J. Gay and M. Hole. Types and subtypes for client-server interactions. In *ESOP 1999*, pages 74–90, 1999.
20. S. J. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.

21. M. G. Gouda, E. G. Manning, and Y. Yu. On the progress of communications between two finite state machines. *Information and Control*, 63(3):200–216, 1984.
22. Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. *Electr. Notes Theor. Comput. Sci.*, 5:2–13, 1996.
23. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP 1998*, pages 122–138, 1998.
24. R. Hu and N. Yoshida. Hybrid session verification through endpoint API generation. In *FASE 2016*, pages 401–418, 2016.
25. H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. 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, 2016.
26. P. Jancar and F. Moller. Techniques for decidability and undecidability of bisimilarity. In *CONCUR 1999*, pages 30–45, 1999.
27. J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *POPL 2015*, pages 221–232, 2015.
28. J. Lange and N. Yoshida. Characteristic formulae for session types. In *TACAS*, pages 833–850, 2016.
29. D. Mostrous. *Session Types in Concurrent Calculi: Higher-Order Processes and Objects*. PhD thesis, Imperial College London, November 2009.
30. D. Mostrous and N. Yoshida. Session-based communication optimisation for higher-order mobile processes. In *TLCA 2009*, pages 203–218, 2009.
31. D. Mostrous and N. Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015.
32. D. Mostrous, N. Yoshida, and K. Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP 2009*, pages 316–332, 2009.
33. N. Ng, J. G. de Figueiredo Coutinho, and N. Yoshida. Protocols by default - safe MPI code generation based on session types. In *CC 2015*, pages 212–232, 2015.
34. N. Ng, N. Yoshida, and K. Honda. Multiparty session C: safe parallel programming with message optimisation. In *TOOLS 2012*, pages 202–218, 2012.
35. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE 1994*, pages 398–413, 1994.
36. N. Yoshida, V. T. Vasconcelos, H. Paulino, and K. Honda. Session-based compilation framework for multicore programming. In *FMCO 2008*, pages 226–246, 2008.

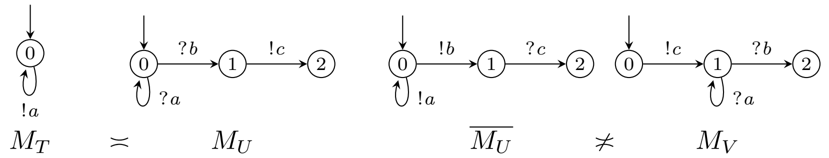
A Comments on transitivity of $\asymp$

A key property of the asynchronous subtyping is that it is transitive. Let us recall our transitivity result for $\asymp$.

Theorem 5.2. *If $M_1 \asymp \overline{M}$ and $M \asymp M_2$, then $M_1 \asymp M_2$.*

In practice, this means that if software practitioners define a new two-party protocol, specified by M, whose parties are to be implemented by different teams, then the teams do not have to check whether their respective implementations, e.g., M_1 and M_2, are compatible, it is enough for them to refer to either M or $\overline{M}$ (depending on which party they are implementing) so that they can, e.g., optimise their implementation as they wish.

The $\texttt{fin}(\_)$ requirements in AD and $\asymp$ (and $\& \in T$ in asynchronous subtyping) is key to guarantee transitivity as we illustrate below.



We have $M_T \asymp M_U$ and system (M_T, M_U) is safe and AD. We have $\overline{M_U} \not\asymp M_V$ (since $\neg\texttt{fin}(0)$ in $\overline{M_U}$ and both initial states are sending), but system $(\overline{M_U}, M_V)$ is safe (although not AD).

If we were to remove the $\texttt{fin}(\_)$ conditions in Definition 3.1, we would have $\overline{M_U} \asymp M_V$, which by transitivity would give us $M_T \asymp M_V$. However, the system (M_T, M_V) is not safe since the message c (sent by M_V) will never be received by M_T.

B Proofs for Section 2.1 (Properties of CFSMs)

Lemma B.1. *If S is* safe*, then for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$: either (i) q_1 and q_2 are final and $\omega_1 = \omega_2 = \epsilon$, or (ii) $\exists s' \in RS(S) : s \Rightarrow s'$.*

Proof. Take $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$, we make a case analysis on the type of q_1 and q_2, and whether or not the queues are empty.

1. If q_1 and q_2 are final, then $\omega_1 = \omega_2 = \epsilon$, otherwise s would be a deadlock (which contradicts the safety assumption).
2. If $q_i \xrightarrow{!a}$ for $i \in \{1, 2\}$, then the result holds trivially: by Definition 2.3, we have $s \xRightarrow{ij!a}$.
3. If there is $i \in \{1, 2\}$ such that q_i is a receiving state, we have the following sub-cases (letting $j \in \{1, 2\} \setminus \{i\}$):

(a) if $\omega_j = a \cdot \omega'_j$, then by eventual reception we have $q_i \xrightarrow{?a}$ (q_i is a receiving state), hence $s \xRightarrow{ji?a}$.
(b) $\omega_j = \epsilon$, then we have either:
   i. $q_j \xrightarrow{!b}$ and the result holds (see (2) above)
   ii. q_j is final, then $\omega_i = \epsilon$ (otherwise, we have a contradiction with eventual reception), hence for s not to be a deadlock q_i must be either sending or final, a contradiction with the assumption of this sub-case.
   iii. $q_j \xrightarrow{?b}$, then either
      – $\omega_i = \epsilon$, which implies that s is a deadlock, i.e., a contradiction, or
      – $\omega_i = c \cdot \omega'_i$, which implies that $q_j \xrightarrow{?c}$ (eventual reception), hence $s \xRightarrow{ij?c}$. □

Lemma B.2. *If $S = (M_1, M_2)$ is* safe*, then for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$:*

1. *$\exists s' \in RS(S) : s \Rightarrow^* s' = (q_1, \epsilon, q'_2, \omega_2 \cdot \omega'_2)$, and*
2. *$\exists s'' \in RS(S) : s \Rightarrow^* s'' = (q''_1, \omega_1 \cdot \omega''_1, q_2, \epsilon)$.*

Proof. Take an arbitrary $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$, let us show how a configuration $s' = (q_1, \epsilon, q'_2, \omega'_2)$ is reachable from s'.

– If q_2 is a sending state, let $q_2 \xrightarrow{!a} q''_2$, then we obtain $s'' = (q_1, \omega_1, q''_2, \omega'_2 \cdot a)$ and we can repeat the procedure from s'' (note that ω_1 is unchanged).
– If q_2 is a receiving state, then either
  - $\omega_1 = \epsilon$ and we have obtained the expected result, or
  - $\omega_1 = a \cdot \omega''_1$ and $q_2 \xrightarrow{?a} q''_2$, then we obtain $s'' = (q_1, \omega''_1, q''_2, \omega_2)$, and we repeat the procedure from s'' (note that $|\omega_1| > |\omega''_1|$);
  - or $\forall (q_2, a, q''_2) \in \delta_2 : \omega_1 \notin a \cdot \mathbb{A}^*$ which contradicts the fact that S is *safe* (it does not satisfy eventual reception).
– If q_2 is final, then either $\omega_1 = \epsilon$ and we have obtained the expected result, or $\omega_1 = a \cdot \omega''_1$ and we have a contradiction with the fact that S is *safe*.

The procedure must terminate since (i) the size of ω_1 does not increase and (ii) eventual reception guarantees that all messages are eventually consumed.

The procedure to show that $s'' = (q''_1, \omega''_1, q_2, \epsilon)$ is reachable is similar to the one above (making M_1 move instead of M_2). □

C Proofs for Section 3 (Soundness of $\asymp$)

We give a convenient definition used in the proofs below.

Definition C.1. *Given a system S, we says that $s = (p, \omega_1, q, \omega_2) \in RS(S)$ is* well-formed*, written* $\mathtt{WF}(s)$*, if*

$$A(q, \omega_1) \quad \textit{and} \quad \forall (\varphi, q') \in W(q, \omega_1) : \omega_2 \cdot \varphi \blacktriangleright p \asymp q'$$

Lemma 3.2. *Let $S = (M_1, M_2)$. If $M_1 \asymp M_2$, then for all $s = (p, \omega_1, q, \omega_2) \in RS(S)$ the following holds:* (1) *s is not a deadlock,* (2) *$A(q, \omega_1)$,* (3) *$\forall(\varphi, q') \in W(q, \omega_1) : \omega_2 \cdot \varphi \blacktriangleright p \asymp q'$, and* (4) *$A(p, \omega_2)$.*

Proof. We show this by induction on the length of the execution from s_0 to $s \in RS(S)$. Consider $s = (p, \omega_1, q, \omega_2)$ such that $s_0 \overset{\phi}{\Rightarrow} s$ with $|\phi| = k$.

Base case. If $k = 0$, then $s = s_0 = (p_0, \epsilon, q_0, \epsilon) = (p, \epsilon, q, \epsilon)$.

We have to show

- $A(p, \epsilon)$, which is trivially true,
- $A(q, \epsilon)$, which is trivially true, and
- $\forall(\varphi, q') \in W(q_0, \epsilon) : \varphi \blacktriangleright p_0 \asymp q'$. Since, $\omega_1 = \epsilon$, we have $W(q, \epsilon) = \{(q_0, \epsilon)\}$, hence we only have to show that $\epsilon \blacktriangleright p_0 \asymp q_0$, which holds by assumption that $M_1 \asymp M_2$.

Next, we show that s_0 is not a deadlock, by contradiction. Assume we have $s_0 = (p, \epsilon, q, \epsilon)$. By definition of a deadlock, if s_0 is a deadlock we must have either:

1. $p \xrightarrow{?a} p''$ and q is either final or $q \xrightarrow{?b}$, or
2. $q \xrightarrow{?a} q''$ and p is either final or $p \xrightarrow{?b}$.

Assume case (1) (the other case is similar) above holds and let us show that it leads to a contradiction. We have $W(q, \epsilon) = \{(\epsilon, q)\}$, and thus $\epsilon \blacktriangleright p \asymp q$.

Since $p \xrightarrow{?a} p''$, only case (2a) of Definition 3.1 would apply. This lead to a contradiction since we have $p \xrightarrow{?a} p''$ and either q is final or $q \xrightarrow{?b}$ both cases rule out case (2a)).

Inductive case. Assume the results holds for any $k - 1 \geqslant 0$ and let us show that it holds for k. Pose $s' = (p', \omega_1', q', \omega_2')$ such that $s_0 \overset{\phi'}{\Rightarrow} s' \overset{\lambda}{\Rightarrow} s$.

There are four cases depending on the form of λ:

1. If $\lambda = {}_{12}!a$, then we have $s = (p, \omega_1' \cdot a, q', \omega_2')$. We first note that, since $p' \xrightarrow{!a} p$, $A(p', \omega_2') \implies A(p, \omega_2')$.

   Then, by induction hypothesis, we have $\mathtt{WF}(s')$ hence we have

   $$A(q', \omega_1') \quad \text{and} \quad \forall(\varphi, q'') \in W(q', \omega_1') : \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$$

   Since $\lambda = {}_{12}!a$, we have $p' \xrightarrow{!a} p$, hence case (3) of Definition (3.1) must apply for each $\omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$.

   First, we show that $A(q', \omega_1 \cdot a)$.

   - If $\omega_2 \cdot \varphi \neq \epsilon$, then we have, by Definition (3.1):
     $\forall q_1 \in Q_2 : \forall \pi \in \mathbb{A}^* : q'' \xrightarrow{!\pi} q_1$, there exist $\pi' \in \mathbb{A}^*$ and $q_2, q_3 \in Q_2$ such that $q_1 \xrightarrow{!\pi'} q_2 \xrightarrow{?a} q_3$ and $\omega_2 \cdot \varphi \cdot \pi \cdot \pi' \blacktriangleright p \asymp q_3$
     Which implies that we have:
     $\forall q_1 \in Q_2 : q'' \xrightarrow{!}{}^{*} q_1$ there exist $q_2, q_3 \in Q_2 : q_1 \xrightarrow{!}{}^{*} q_2 \xrightarrow{?a} q_3$, or in other words: $A(q'', a)$.

  - If $\omega_2 \cdot \varphi = \epsilon$, then we must have, by Definition 3.1: $q'' \xrightarrow{?a} q_3$ and $\epsilon \blacktriangleright p \asymp q_3$. Hence, we have $A(q'', a)$.

Since we have $A(q', \omega_1')$ (by assumption) and $\forall (\psi, q'') \in W(q', \omega_1) : A(q'', a)$ (from the development above), we have $A(q', \omega_1 \cdot a)$ by Lemma 3.1.
Second, we show that

$$\forall (\varphi', q'') \in W(q', \omega_1 \cdot a) : \omega_2 \cdot \varphi' \blacktriangleright p \asymp q''$$

which follows from the first part of our argument. It suffices to notice that for all $(\varphi', q'') \in W(q', \omega_1 \cdot a)$ there are $(\hat{\varphi}, \hat{q}) \in W(q', \omega_1)$, and $q_2, q_3 \in Q_2$ such that $\hat{q} \xrightarrow{!\varphi''} q_2 \xrightarrow{?a} q_3$ with $\varphi' = \hat{\varphi} \cdot \varphi''$ and $q'' = q_3$.
Finally, note that s is not a deadlock since $\omega_1 \neq \epsilon$.

2. If $\lambda = {}_{21}!a$, then we have $s = (p', \omega_1', q, \omega_2' \cdot a)$. By induction hypothesis, we have $\mathtt{WF}(s')$ hence we have

$$A(q', \omega_1') \quad \text{and} \quad \forall (\varphi, q'') \in W(q', \omega_1') : \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$$

We have to show that,

$$A(q, \omega_1') \quad \text{and} \quad \forall (\varphi, q'') \in W(q, \omega_1') : \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$$

which follows trivially since $q' \xrightarrow{!a} q$.
Next, we have to show that $A(p', \omega_2' \cdot a)$, knowing that $A(p', \omega_2')$ holds by induction hypothesis. This follows from Lemma F.1: since $\mathtt{fin}(p'')$ holds for any p'' appearing in the following derivations, it must be the case that ω_2' is eventually totally consumed (by repeated applications of case (2b)) and thus eventually reaches a step where a must be consumed by the first machine.
Finally, note that s is not a deadlock since $\omega_2 \neq \epsilon$.

3. If $\lambda = {}_{12}?a$, then we have $s = (p', \omega_1, q, \omega_2')$, with $\omega_1' = a \cdot \omega_1$.
We first note that, since $p' = p$, $A(p', \omega_2') \implies A(p, \omega_2')$.
By induction hypothesis, we have $\mathtt{WF}(s')$ hence we have

$$A(q', a \cdot \omega_1) \quad \text{and} \quad \forall (\varphi, q'') \in W(q', a \cdot \omega_1) : \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$$

We have to show that

$$A(q, \omega_1) \quad \text{and} \quad \forall (\varphi, q'') \in W(q, \omega_1) : \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$$

which follows trivially since $q' \xrightarrow{?a} q$.
Finally, if $\omega_1 = \epsilon = \omega_2'$, we can show that $s = (p', \omega_1, q, \omega_2')$ is not a deadlock by contradiction, just like in the base case.

4. If $\lambda = {}_{21}?a$, then we have $s = (p, \omega_1', q', \omega_2)$, with $\omega_2' = a \cdot \omega_2$.
We first note that, since $p' \xrightarrow{?a} p$, $A(p', a \cdot \omega_2') \implies A(p, \omega_2)$.
By induction hypothesis, we have $\mathtt{WF}(s')$ hence we have

$$A(q', \omega_1') \quad \text{and} \quad \forall (\varphi, q'') \in W(q', \omega_1') : a \cdot \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$$

We have to show that

$$A(q', \omega_1') \quad \text{and} \quad \forall (\varphi, q'') \in W(q', \omega_1') \,:\, \omega_2 \cdot \varphi \blacktriangleright p \asymp q''$$

Since $p' \xrightarrow{?a} p$, the second sub-case (2a) of Definition (3.1) must apply to each $a \cdot \omega_2 \cdot \varphi \blacktriangleright p' \asymp q''$. Hence, we must have $\omega_2 \cdot \varphi \blacktriangleright p \asymp q''$, which is the expected result.
Finally, if $\omega_1' = \epsilon = \omega_2$, we can show that $s = (p, \omega_1', q', \omega_2)$ is not a deadlock by contradiction, just like in the base case.

□

Lemma 3.3. *Let $S = (M_1, M_2)$. If for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$: $A(q_1, \omega_1)$ and $A(q_2, \omega_2)$, then S satisfies* eventual reception.

Proof. We show that, for all $s = (p, \omega, q, \omega') \in RS(S)$

$$A(p, \omega') \implies \exists s' \,:\, s \Rightarrow^* s' = (p', \omega \cdot \varphi, q, \epsilon) \tag{1}$$

and

$$A(q, \omega) \implies \exists s' \,:\, s \Rightarrow^* s' = (p, \epsilon, q', \omega' \cdot \varphi') \tag{2}$$

which implies naturally eventual reception (when (1) and (2) hold for all s).

We show only (2) since (1) follows the same argument. Below, we show that the predicate $A(q, \omega)$ is preserved by the moves done by q. The case where $\omega = \epsilon$ follows trivially, hence we only detail the case where $\omega = a \cdot \omega'$.

- If q is a sending state, then we must show that $\forall b, q_1 \,:\, q \xrightarrow{!b} q_1 \implies A(q_1, \omega)$. By hypothesis, we have $A(q, \omega)$, i.e., $\forall q' \,:\, q \xrightarrow{!}{}^* q' \,:\, \exists q'', \hat{q} \,:\, q' \xrightarrow{!}{}^* q'' \xrightarrow{?a} \hat{q} \wedge A(\hat{q}, \omega')$.
  In addition, we have
  $$\left\{ q \mid q_1 \xrightarrow{!}{}^* q' \right\} \subseteq \left\{ q \mid q \xrightarrow{!}{}^* q' \right\}$$
  since $q_1 \xrightarrow{!}{}^* q' \implies q \xrightarrow{!b} \xrightarrow{!}{}^* q'$ for some b.
  Hence we obtain:
  $$\begin{aligned} &\forall q' \,:\, q_1 \xrightarrow{!}{}^* q' \,:\, \exists q'', \hat{q} \,:\, q' \xrightarrow{!}{}^* q'' \xrightarrow{?a} \hat{q} \wedge A(\hat{q}, \omega') \\ &\qquad\qquad \iff \quad \forall b, q_1 \,:\, q \xrightarrow{!b} q_1 \,:\, A(q_1, \omega') \end{aligned}$$
- If q is a receiving state, then we must show that
  $$A(q, \omega) \wedge \omega = a \cdot \omega' \implies q \xrightarrow{?a} q' \wedge A(q', \omega')$$
  which follows from the fact that $\{q'' \mid q \xrightarrow{!}{}^* q''\} = \{q\}$ since q is a *receiving* state. Hence, by definition of $A(q, \omega)$, we have $q \xrightarrow{?a} q' \wedge A(q', \omega')$ as required.
- If q is a final state, and $\omega \neq \epsilon$, we have a contradiction with the definition of $A(q, \omega)$.

$\square$

Lemma 3.1. *Let* $M = (Q, q_0, \delta)$*,* $q \in Q$ *and* $\omega \in \mathbb{A}^*$*. If* $A(q, \omega)$ *and* $\forall(\varphi, q') \in W(q, \omega) : A(q', a)$ *then* $A(q, \omega \cdot a)$*.*

Proof. By induction on the size of ω.

- If $\omega = \epsilon$, then $W(q, \omega) = \{(\epsilon, q)\}$, hence $A(q, a)$, as required.
- Assume the result holds for $|\omega| = n \geqslant 0$ and let us show that it also holds for $|\omega| = n + 1$.
  Take $\omega = b \cdot \omega'$. By $A(q, \omega)$, we have

  $$\forall q' : q \xrightarrow{!}{}^{*} q' : \exists q'', \hat{q} : q' \xrightarrow{!}{}^{*} q'' \xrightarrow{?b} \hat{q} \wedge A(\hat{q}, \omega') \tag{3}$$

  By definition of $W(q, \omega)$ and assumption that $A(q, \omega)$ hold, in particular $q \xrightarrow{!}{}^{*} \xrightarrow{?b} \hat{q}$, we have

  $$\{q_0 \mid (\_, q_0) \in W(\hat{q}, \omega')\} \subseteq \{q_1 \mid (\_, q_1) \in W(q, b \cdot \omega')\} \tag{4}$$

  Finally, from (4) and since, by assumption, we have

  $$\forall(\_, q') \in W(q, \omega) : A(q', a)$$

  we also have

  $$\forall(\_, q') \in W(\hat{q}, \omega') : A(q', a)$$

  which, together with (3), allows us to invoke the induction hypothesis, i.e.,

  $$A(\hat{q}, \omega') \text{ and } \forall(\_, q') \in W(\hat{q}, \omega') : A(q', a) \implies A(\hat{q}, \omega' \cdot a)$$

  Applying weakening in (3), we obtain

  $$\forall q' : q \xrightarrow{!}{}^{*} q' : \exists q'', \hat{q} : q' \xrightarrow{!}{}^{*} q'' \xrightarrow{?b} \hat{q} \wedge A(\hat{q}, \omega' \cdot a) \quad \iff A(q, b \cdot \omega' \cdot a)$$

  i.e., the expected result.

$\square$

Theorem C.1. *If* $M_1 \asymp M_2$*, then* (M_1, M_2) *is* safe*.*

Proof. Direct consequence of Lemmas 3.2 and 3.3. $\square$

Theorem C.2. *If* $M_1 \asymp M_2$*, then* (M_1, M_2) *is an* asynchronous duplex *system.*

Proof. Take $S = (M_1, M_2)$ such that $M_1 \asymp M_2$. By contradiction, assume there is $s = (p, \omega_1, q, \omega_2) \in RS(S)$ such that, $\omega_1 \neq \epsilon$, $\omega_2 \neq \epsilon$ and $\neg\texttt{fin}(p)$ (if $\neg\texttt{fin}(q)$ the proof is similar).

Since S is safe, by Lemma 2.1, there is $s' \in RS(S)$ such that $s \Rightarrow^* s' = (p, \epsilon, q', \omega_2 \cdot \omega_2')$. Hence, $\omega_2 \cdot \omega_2' \blacktriangleright p \asymp q'$ holds by Lemma 3.4. Finally, by Lemma F.1, we must have $\texttt{fin}(p)$, a contradiction. $\square$

Theorem 3.1. *If $M_1 \asymp M_2$, then (M_1, M_2) is a* safe AD *system.*

Proof. By Theorems C.1 and C.2. □

Theorem 3.2. *If $M_1 \asymp_s M_2$, then (M_1, M_2) is a* safe HD *system.*

Proof. Since $\leqslant_{\mathsf{s}} \subseteq \leqslant_{\mathsf{a}}$, the safety part follows from Theorem 3.1. The HD part follows trivially from the definition of $\asymp_s$ (if two machines are sending simultaneously, none of the cases of $\asymp_s$ applies). □

D Proofs for Section 3 (completeness of $\asymp$)

Lemma 3.4. *Let S be* safe *and* AD, *then $\forall (p, \epsilon, q, \omega) \in RS(S) : \omega \blacktriangleright p \asymp q$.*

Proof. We show this result by induction on the k^{th} approximation of $\omega \blacktriangleright p \asymp q$, i.e., $\omega \blacktriangleright p \asymp_k q$.

Base case. If $k = 0$, then we have the result trivially since $\omega \blacktriangleright p \asymp_0 q$, for any p, q, and ω.

Inductive case. Assume that for all $\forall s = (p, \epsilon, q, \omega) \in RS(S)$, we have $\omega \blacktriangleright p \asymp_k q$, let us show that we have $\omega \blacktriangleright p \asymp_{k+1} q$.

1. If $p \not\rightarrow$, then, by definition of safety, we must have $\omega = \epsilon$ (eventual reception) and $q \not\rightarrow$ (no deadlock). Hence, we have $\omega \blacktriangleright p \asymp q$, following Case (1) of Definition 3.1.
2. If $p \xrightarrow{?a}$, then we have two cases, depending on ω being empty or not.
   - If $\omega = \epsilon$, then by safety we must have $q \xrightarrow{!b}$ (otherwise, we have a deadlock).

     In addition, by eventual reception, we must have $\forall b : q \xrightarrow{!b} q' \implies p \xrightarrow{!b} p'$ and $(p, \epsilon, q, \epsilon)$ must be safe.

     Hence, Case (2a) of Definition 3.1 applies here, since by the induction hypothesis, we must have $\epsilon \blacktriangleright p' \asymp_k q'$ for all such p' and q'.
   - If $\omega = b \cdot \omega'$, then, by safety, we must have $p \xrightarrow{?b} p'$ and each $(p', \epsilon, q, \omega')$ must be safe.

     Hence, Case (2b) of Definition 3.1 applies here since, by the induction hypothesis, we must have $\omega' \blacktriangleright p' \asymp_k q$.
3. If $p \xrightarrow{!a} p'$, we have two cases depending on whether $\texttt{fin}(p) \wedge \texttt{fin}(q)$ holds. We first show that $\omega \neq \epsilon \implies \texttt{fin}(p) \wedge \texttt{fin}(q)$ by contradiction. We have $s \Rightarrow s' = (p', a, q, \omega)$, s' is not a valid configuration of an asynchronous duplex configuration if $\omega \neq \epsilon \wedge \neg(\texttt{fin}(p) \wedge \texttt{fin}(q))$.
   - Case $\neg(\texttt{fin}(p) \wedge \texttt{fin}(q))$, then we must have $\omega = \epsilon$ and $q \xrightarrow{?b}$. Then, by safety, we must have $p \xrightarrow{!a} p' \implies q \xrightarrow{?a} q'$ and $(p', \epsilon, q', \epsilon)$ is safe.

     Hence, Case (3a) of Definition 3.1 applies here since, by the induction hypothesis, we must have $\epsilon \blacktriangleright p' \asymp_k q'$ for all such p' and q'.

– Case $\texttt{fin}(p) \wedge \texttt{fin}(q)$. By safety, it must be the case that:
$\forall q' \in Q_2 \,:\, \forall \pi \in \mathbb{A}^* \,:\, q \xrightarrow{!\pi'} q'$, there exist $\pi'' \in \mathbb{A}^*$ and $q'', q_1 \in Q_2$ such that $q' \xrightarrow{!\pi'} q'' \xrightarrow{?a} q_1$ (i.e., a can always be received from state q) and $(p', \epsilon, q_1, \omega \cdot \pi \cdot \pi')$ is safe.
By induction hypothesis, we have $\omega \cdot \pi \cdot \pi' \blacktriangleright p' \asymp_k q_1$ for each such configuration, hence, Case (3b) of Definition 3.1 applies.

□

Theorem 3.4. *If* (M_1, M_2) *is a* safe HD *system, then* $M_1 \asymp_s M_2$*.*

Proof. The proof is a degenerated case of the proof of Theorem 3.3. We only show the basic idea here. Take $S = (M_1, M_2)$ a safe HD system and $s = (p, \omega_1, q, \omega_2) \in RS(S)$. By definition of HD: $\omega_1 = \epsilon$ or $\omega_2 = \epsilon$. Take $\omega_1 = \epsilon$ (the other case is similar). Since S is safe, we can use Lemma 2.1 and the HD assumption (if M_1 could send a message while ω_2 is not yet empty, it would contradict the HD hypothesis) and to reach $s' = (p', \epsilon, q, \epsilon)$ such that $s \Rightarrow^* s'$. Then we show that $p' \asymp_s q$ holds as in the proof of Theorem 3.3. □

E Proofs for Section 4 (undecidability)

Theorem 4.1 (Undecidability of $\asymp$). *Given two machines* M_1 *and* M_2*, it is generally undecidable whether* $M_1 \asymp M_2$ *holds.*

Proof. We prove that following statements are equivalent: (1) TM accepts ω, (2) $S(TM, \omega) = (A_1, A_2)$ is *not* safe, and (3) $\neg(A_1 \asymp A_2)$.

(1) ⇒ (2): We show the contrapositive, i.e., if $S(TM, \omega)$ is safe, then it does not halt (i.e., ω is not accepted). By Lemma 2.1 and the fact that neither A_i contains final states each reachable configuration of $S(TM, \omega)$ has a successor, hence it does not halt (thus ω is not accepted).

(2) ⇒ (1): We show the contrapositive: if TM does not accepts ω, then $S(TM, \omega)$ is safe. In other words, if TM does not halt, then $S(TM, \omega)$ is safe. Assume by contradiction that TM does not halt and $S(TM, \omega)$ is *not* safe. Then by definition of safety, there must be $s \in RS(S(TM, \omega))$ such that either s is a deadlock or s does not satisfy eventual reception.

(*i*) If s is a deadlock, then it clearly contradicts the fact that TM does not halt (from [17] we now that $S(TM, \omega)$ simulates TM).

(*ii*) Assume s does not satisfy eventual reception. Without loss of generality, take $s = (q_1, a \cdot \omega_1, q_2, \omega_2)$ with q'_2 such that $q_2 \xrightarrow{!}{}^* q'_2$ and $\neg(q'_2 \xrightarrow{?a})$. Then we have $s \rightarrow^* s' = (q_1, a \cdot \omega_1, q'_2, \omega_2 \cdot \omega'_2)$ for some ω'_2 such that $q_2 \xrightarrow{!\omega'_2} q'_2$. Since $\neg(q'_2 \xrightarrow{?a})$, machine A_2 is stuck in q'_2; hence all further moves must be done by A_1 only. Clearly, the size of the input queue of A_1 is also stuck with maximal content $\omega_2 \cdot \omega'_2$. Since each cycle in A_1 contains at least one reception, its input queue will eventually be emptied and A_1 will be stuck as well, contradicting the fact that $S(TM, \omega)$ does not halt.

(2) $\Leftrightarrow$ (3): By Lemma 4.1, $S(TM, \omega)$ is an asynchronous duplex system. Hence the result follows from Theorems 3.1 and 3.3 (equivalence between safety and $\asymp$ for asynchronous duplex systems).

We have reduced the halting problem for Turing machines to the problem of deciding whether $M_1 \asymp M_2$ holds, hence checking $M_1 \asymp M_2$ is undecidable. $\square$

F Proofs for Section 4.1 (decidable sub-classes)

In order to give an algorithm for checking whether $M_1 \asymp M_2$ holds, we adapt the notion of expansion tree [22, 26] to our setting.

Definition F.1. *The function succ* : $(\mathbb{A}^* \times Q_1 \times Q_2) \rightarrow \mathcal{P}(\mathbb{A}^* \times Q_1 \times Q_2)$ *is defined as follows:*

1. $\{(\epsilon, p', q') \mid p \xrightarrow{?b} p' \wedge q \xrightarrow{!b} q'\}$ *if* $p \xrightarrow{?a}, q \xrightarrow{!b}, \pi = \epsilon,$ *and* $q \xrightarrow{!b} \implies p \xrightarrow{?b}$
2. $\{(\pi', p', q)\}$ *if* $\pi = a \cdot \pi'$ *and* $p \xrightarrow{?a} p'$
3. $\{(\epsilon, p', q') \mid p \xrightarrow{!b} p' \wedge q \xrightarrow{?b} q'\}$ *if* $\neg(\mathtt{fin}(p) \wedge \mathtt{fin}(q))$ *and* $p \xrightarrow{!a}, q \xrightarrow{?b}, \pi = \epsilon,$ *and* $p \xrightarrow{!a} \implies q \xrightarrow{?a}$
4. $\{(\pi \cdot \pi', p', q') \mid p \xrightarrow{!a} p' \wedge q \xrightarrow{!\pi'}\xrightarrow{?a} q'\}$ *if* $\mathtt{fin}(p) \wedge \mathtt{fin}(q)$*,* $p \xrightarrow{!a}$*, and* $p \xrightarrow{!a} \implies \forall q' \in Q_2 : q \xrightarrow{!}^{*} q' \implies q' \xrightarrow{!}^{*} \xrightarrow{?a}$
5. $\varnothing$ *otherwise.*

The derivation tree of $\pi \blacktriangleright p \asymp q$ is a tree whose nodes are (labelled by) triples of the form $\mathtt{c} = (\pi_i, p_i, q_i)$, in which the children of a node are precisely the (finitely many) set of nodes in *succ*($\mathtt{c}$). The root of the tree is $\mathtt{c}_0 = (\pi, p, q)$. A leaf (π, p, q) is deemed successful only if $\pi = \epsilon$ and both p and q are final states. All other leaves are deemed unsuccessful. We say that a branch (a full path) is *successful* iff it is infinite or finishes with a successful node; otherwise it is unsuccessful (it finishes with an unsuccessful node). It is clear from Definition F.1 that $\pi \blacktriangleright p \asymp q$ holds if and only if all branches of the derivation tree of $\pi \blacktriangleright p \asymp q$ are successful.

Lemma 4.2. *The derivation tree of* $\pi \blacktriangleright p \asymp q$ *is finitely branching.*

Proof. The only interesting case is case (4) of Definition 4. The finite number of children is due to the requirements that $\mathtt{fin}(q)$ must hold, which guarantees that there are finitely many lists π' such that $q \xrightarrow{!\pi'} q'$. $\square$

Lemma F.1. *Let* $M_i = (Q_i, q_{0_i}, \delta_i)$*,* $i \in \{1, 2\}$*. If* $M_1 \asymp M_2$*, then for every node* $(a \cdot \pi, p, q)$ *which is a child of* $(\epsilon, q_{0_1}, q_{0_2})$ *in the derivation tree of* $\pi \blacktriangleright q_{0_1} \asymp q_{0_2}$*,* $\mathtt{fin}(p)$ *holds.*

Proof. This follows directly from Definition 3.1. The π part of the relation only grows in the second sub-case of (3), where $\mathtt{fin}(p)$ is required to hold. Once $\pi \neq \epsilon$ it can only (*i*) grow again, in which case $\mathtt{fin}(p)$ must still hold, or (*ii*) decrease in which case $p \xrightarrow{?a}$, therefore $\mathtt{fin}(p)$ holds trivially. $\square$

Theorem F.1. *If* (M_1, M_2) *is* half-duplex*, then* $M_1 \asymp M_2$ *is decidable.*

Proof. If (M_1, M_2) is half-duplex, case (3b) of Definition 3.1 never applies since it cannot be the case that both machines are simultaneously in a sending state. Hence, for any node $\pi \blacktriangleright p \asymp q$ in the derivation tree, we have $\pi = \epsilon$; thus the derivation tree is finite-state. $\square$

Theorem 4.2. *If* M_1 *and* M_2 *are* alternating*, then* $M_1 \asymp M_2$ *is decidable.*

Proof. We show that the π part of the relation is bounded by 1 by induction on the depth of the derivation tree. We show only the interesting case here. Let $M_i = (Q_i, q_{0_i}, \delta_i)$ such that $M_1 \asymp M_2$ Take $p \in Q_1$, and $q \in Q_2$ such that $p \xrightarrow{!a} p'$ and $q \xrightarrow{!b} q'$. The successors of $\mathtt{c} = \pi \blacktriangleright p \asymp q$ in the derivation tree have the form $\mathtt{c}' = \pi \cdot b \blacktriangleright p' \asymp q''$, taking q'' such that $q' \xrightarrow{?a} q''$ since $M_1 \asymp M_2$ and M_2 is alternating. The (unique) successor of $\mathtt{c}'$ must be $\pi' \cdot b \blacktriangleright p'' \asymp q''$, with $\pi = c \cdot \pi'$ and taking p'' such that $p' \xrightarrow{?c} p''$ since M_1 is alternating. $\square$

Lemma F.2. *Let* $M_i = (Q_i, q_{0_i}, \delta_i)$*,* $i \in \{1, 2\}$*, be two machines such that at least one of them is* not branching *and* $M_1 \asymp M_2$ *holds, then the derivation tree of* $\epsilon \blacktriangleright q_{0_1} \asymp q_{0_2}$ *(resp.* $\epsilon \blacktriangleright q_{0_2} \asymp q_{0_1}$*) has at most one branch.*

Proof. Take a node $\mathtt{c} = (\pi, p, q)$ in the derivation tree of $\epsilon \blacktriangleright q_{0_1} \asymp q_{0_2}$, we make a case analysis on the type of p, following Definition F.1, we show: $|succ(\mathtt{c})| \leqslant 1$.

1. If $p \nrightarrow$ then $succ(\mathtt{c}) = \varnothing$
2. If $p \xrightarrow{?a}$,
   (a) If $\pi = \epsilon$. We know that there is a unique transition such that $q \xrightarrow{!b} q'$, and we must have $p \xrightarrow{?b} p'$, hence $succ(\mathtt{c}) = \{(\epsilon, p', q')\}$.
   (b) If $\pi = b \cdot \pi'$, then $|succ(\mathtt{c})| = 1$ by Definition F.1.
3. If $p \xrightarrow{?a}$,
   (a) If $\pi = \epsilon$ then there must be a unique transition such that $p \xrightarrow{!b} p'$, since there is a unique receive action fireable from q (no branching). Hence we must have $q \xrightarrow{?b} q'$ and $succ(\mathtt{c}) = \{(\epsilon, p', q')\}$.
   (b) If $\mathtt{fin}(p) \wedge \mathtt{fin}(q)$. As above, there must be a unique transition such that $p \xrightarrow{!b} p'$, since there is no branching in M_2. Also there must be a unique state q' such that $q \xrightarrow{!}{}^{*} \xrightarrow{?a} q'$ since M_2 does not contain any branching. Take π' such that $q \xrightarrow{!\pi'} \xrightarrow{?a} q'$, we have $succ(\mathtt{c}) = \{(\pi \cdot \pi', p', q')\}$.

The reasoning is similar to show that the derivation tree of $\epsilon \blacktriangleright q_{0_2} \asymp q_{0_1}$ has at most one branch, we show only the interesting case, i.e., take $\mathtt{c} = (\pi, p, q)$ in the derivation tree of $\epsilon \blacktriangleright q_{0_2} \asymp q_{0_1}$ such that $p \xrightarrow{!a} p'$ and $\mathtt{fin}(p) \wedge \mathtt{fin}(q)$.

Take $\pi = \epsilon$ and assume by contradiction that $q \xrightarrow{!b} \xrightarrow{?a} q'$ and $q \xrightarrow{!c} \xrightarrow{?a} q''$. Then both $b \blacktriangleright p' \asymp q'$ and $c \blacktriangleright p' \asymp q''$ must hold. Which leads to a contradiction since M_1 is not branching (i.e., it cannot consume both b and c). $\square$

Given $\psi \in Act^*$, we define

$$snd(\psi) \stackrel{\text{def}}{=} \begin{cases} a \cdot snd(\psi') & \text{if } \psi = !a \cdot \psi' \\ snd(\psi') & \text{if } \psi = ?a \cdot \psi' \\ \epsilon & \text{otherwise} \end{cases} \qquad rcv(\psi) \stackrel{\text{def}}{=} \begin{cases} a \cdot rcv(\psi') & \text{if } \psi = ?a \cdot \psi' \\ rcv(\psi') & \text{if } \psi = !a \cdot \psi' \\ \epsilon & \text{otherwise} \end{cases}$$

Theorem 4.3. *Let M_1 and M_2 be two machines such that at least one of them is* non-branching*, then $M_1 \asymp M_2$ is decidable.*

Proof. Take $M_2 = (Q_2, q_0, \delta_2)$ such that M_2 is not branching. We show that the problem of checking $M_1 \asymp M_2$ is decidable, observing that if it is M_1 that is not branching, then we can check $M_2 \asymp M_1$ (which implies $M_1 \asymp M_2$ by Theorem 3.5).

From Lemma 4.2, we know that the case $\neg(M_1 \asymp M_2)$ is semi-decidable. It suffices to find the first unsuccessful leaf in the derivation tree [26].

We call a (possibly infinite) sequence $\mathsf{c}_i \cdots \mathsf{c}_j \cdots$ *strictly positive* if $\forall k \geqslant i : \mathsf{c}_k = (\pi_k, p', q') \implies \pi_k \neq \epsilon$.

If a branch is finite, then we can always decide whether or not it is successful, similarly if it is finite state (i.e., if the length of the π component of each triple is bounded). Hence, we focus on infinite branch over an infinite set of triples, i.e., where the π component increases infinitely. It is easy to see that such infinite branches must have a suffix that is a strictly positive sequence, hence we only address strictly increasing sequences.

Part 1. Consider a branch:

$$\mathsf{c}_0 \cdot \mathsf{c}_1 \cdots \mathsf{c}_i \cdots \mathsf{c}_j \cdots \tag{5}$$

we first show that if there is $0 \leqslant i < j$ such that

1. $\mathsf{c}_i = (\omega^n, p, q)$, and $\mathsf{c}_j = (\omega^m, p, q)$, with $n \leqslant m$, and
2. $\mathsf{c}_i \cdots \mathsf{c}_j$ is a strictly positive sequence.

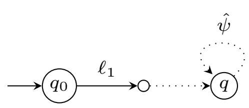
then branch (5) is *infinite* (i.e., successful).

Since there is a cycle in p with the message list non-empty, some of ω^n must be consumed between the two configurations. Hence, we first observe that we must have $\mathsf{c}_i = (\omega^{l_1} \cdot \omega^{l_2}, p, q)$, with $l_1 + l_2 = n$, and $\mathsf{c}_j = (\omega^{l_2} \cdot \omega^{l_3}, p, q)$, with $l_2 + l_3 = m \geqslant n = l_1 + l_2$. Note that since $m \geqslant n$, we must have $l_3 \geqslant l_1$ (i.e., more is added to the list than is removed).

Since $l_1 \leqslant m$, we can rewrite: $\mathsf{c}_j = (\omega^m, p, q) = (\omega^{l_1} \cdot \omega^{m-l_1}, p, q)$, hence c_j must have a successor $\mathsf{c} = (\omega^{m-l_1} \cdot \omega^{l_3}, p, q)$, since c_j can simulate the first steps of c_i (without going through a triple where the message list is empty).

Since $l_3 \geqslant l_1$, we have $m - l_1 + l_3 \geqslant m$, hence we can repeat this reasoning on c_j and c, and build an infinite sequence.

Part 2. We show that for any strictly positive infinite sequence, we can find a pair of node as in (5) in Part 1. Since we have assumed that M_2 is not branching, M_2 must be of the form



Note that $\mathtt{fin}(q)$ holds since only cases (2) and (4) of Definition F.1 apply in a strictly positive branch. The latter case requires $\mathtt{fin}(q)$ while, the former does not change q. Without loss of generality, pose $q = q_0$. Since M_2 is not branching, there is at most one elementary cycle between q_0 and q_0, let ψ_0 be that path, i.e., $q_0 \xrightarrow{\psi_0} q_0$ and pose $\omega_0 = \mathit{snd}(\psi_0)$. For any state q_i along the cycle $\xrightarrow{\psi_0}$, we assume $q_i \xrightarrow{\psi_i} q_i$ and pose $\omega_i = \mathit{snd}(\psi_i)$. Then, deep enough in the tree each node is of the form $\mathtt{c} = (\pi, \_, q_i)$ with $\pi \in \mathit{suffixes}(\omega_i) \cdot (\omega_i)^*$. We show that for any node $\mathtt{c} = (\pi, \_, q_i)$ with $\pi \in \mathit{suffixes}(\omega_i) \cdot (\omega_i)^*$, $\mathtt{c}$ has a successor of the form $(\hat{\pi}, \_, q_i)$ with $\hat{\pi} \in (\omega_i)^*$.

Assume we have

$$q_0 \xrightarrow{!\pi_0}\xrightarrow{?a_1} q_1 \xrightarrow{!\pi_1}\xrightarrow{?a_2} q_2 \xrightarrow{!\pi_2} \cdots \xrightarrow{?a_k} q_k \xrightarrow{!\pi_k}\xrightarrow{?a_0} q_0$$

Then any node (deep enough in the tree) is of the form, for $n > 1$ and $j \geqslant i$, $(\pi_j \cdots \pi_{i-1} \cdot (\pi_i \cdots \pi_{i-1})^n, p, q)$ if $i > 0$, or $(\pi_j \cdots \pi_k \cdot (\pi_i \cdots \pi_k)^n, p, q)$ if $i = 0$.

Assume an environment E for any state q_i return the size of the "best" prefix encountered so far (i.e., the shortest sequence $\pi_j \cdots \pi_{i-1}$). Then for each application of Definition F.1, size associated to a state decreases or state constant.

Consider $\mathtt{c} = (\pi_j \cdots \pi_{i-1} \cdot (\pi_i \cdots \pi_{i-1})^n, p, q_i)$ and let j the start of the best prefix so far each q_i.

1. If $j = i$, then $\pi_j \cdots \pi_{i-1} \cdot (\pi_i \cdots \pi_{i-1})^n = (\pi_i \cdots \pi_{i-1})^{n+1}$, and we have found a good configuration.
2. If $j > 1$, then
   (a) If $p \xrightarrow{?a}$, pose $\mathit{succ}(\mathtt{c}) = \{(\hat{\pi}, p', q_i)\}$, then we have $E(q_i)$ strictly decreases since we have consumed a message from the prefix.
   (b) If $p \xrightarrow{!a}$, pose $\mathit{succ}(\mathtt{c}) = \{(\pi_j \cdots \pi_{i-1} \cdot (\pi_i \cdots \pi_{i-1})^n \cdot \pi_i, p', q_{i+1})\}$, then $E(q_i)$ is unchanged while $E(q_{i+1})$ is assigned the length of $\pi_j \cdots \pi_{i-1} \cdot \pi_i$ which cannot be worse than the previous step since $j > i$ and therefore $j \geqslant i + 1$. Hence if $j = i + 1$, we have found a configuration, otherwise the invariant is preserved.

Since $\mathtt{fin}(p)$ holds (strictly positive branch), there must be a receive action at most every $|Q_1|$ step, hence step (2a) must be executed infinitely often. Since there is finitely many states in Q_2, the procedure above must terminate. □

G Proofs for Section 5 (Equivalence between $\leqslant_{\mathsf{a}}$ and $\asymp$)

The (finite) LTS of a (closed) session type is given by the rules below.

Definition G.1 (LTS of session types).

$$\frac{j \in I}{\bigoplus_{i\in I} !a_i.\, T_i \xrightarrow{!a_j} T_j} \qquad \frac{j \in I}{\&_{i\in I} ?a_i.\, T_i \xrightarrow{?a_j} T_j} \qquad \frac{T[\mathtt{rec}\,\mathtt{x}.T/\mathtt{x}] \xrightarrow{\dagger a} T'}{\mathtt{rec}\,\mathtt{x}.T \xrightarrow{\dagger a} T'}$$

We first introduce a dual relation of $\leqslant_{\mathsf{a}}$, for which one easily sees that $T_1 \leqslant_{\mathsf{a}} T_2 \iff T_1 \leqslant_{\mathsf{c}} \overline{T_2}$. All the results of Section 5 follow from the results below, via the following proposition.

Proposition G.1. $T \leqslant_{\mathsf{c}} U \iff T \leqslant_{\mathsf{a}} \overline{U}$

Definition G.2 (Asynchronous Context [9]).

$$\mathcal{A} \coloneqq [\,]^n \mid \oplus_{i \in I} ! a_i . \mathcal{A}_i$$

We write $\mathcal{A}[\,]^{n \in N}$ to denote a context with holes indexed by elements of N and $\mathcal{A}[T_n]^{n \in N}$ to denote the same context when the hole $[\,]^n$ has been filled with T_n.

Definition G.3 ($\leqslant_{\mathsf{c}}$-Relation [9]). $\leqslant_{\mathsf{c}}$ *is the* largest *relation that contains the rules:*

$$\frac{\forall i \in I \,:\, T_i \leqslant_{\mathsf{c}} U_i}{\oplus_{i \in I} ! a_i . T_i \leqslant_{\mathsf{c}} \&_{i \in I \cup J} ? a_i . U_i}\ [\textsc{sel}] \qquad \frac{\forall i \in I \,:\, T_i \leqslant_{\mathsf{c}} U_i}{\&_{i \in I \cup J} ? a_i . T_i \leqslant_{\mathsf{c}} \oplus_{i \in I} ! a_i . U_i}\ [\textsc{bra}]$$

$$\frac{\forall i \in I \,:\, T_i \leqslant_{\mathsf{c}} \mathcal{A}[U_i^n]^{n \in N} \quad \& \in T_i}{\oplus_{i \in I} ! a_i . T_i \leqslant_{\mathsf{c}} \mathcal{A}[\&_{i \in I \cup J_n} ? a_i . U_i^n]^{n \in N}}\ [\textsc{async}] \qquad \frac{}{\mathtt{end} \leqslant_{\mathsf{c}} \mathtt{end}}\ [\textsc{end}]$$

The double line in the rules indicates that the rules should be interpreted coinductively*. We are assuming an equi-recursive view of types.*

The predicate below is also adapted from [9].

Definition G.4. *The predicate $\& \in T$ holds if it can be derived from the following rules:*

$$\frac{}{\& \in \&_{i \in I} ? a_i . T_i} \qquad \frac{\forall i \in I \,:\, \& \in T_i}{\& \in \oplus_{i \in I} ! a_i . T_i} \qquad \frac{\& \in T}{\& \in \mathtt{rec}\,\mathtt{x}.T}$$

Lemma G.1. *Let $M = (Q, q_0, \delta)$ and T be a session type.*

1. *For each $q \in Q$ if $\mathtt{fin}(q)$, then $\& \in \mathcal{T}((Q, q, \delta))$.*
2. *If $\& \in T$ and $\mathcal{M}(T) = (\hat{Q}, q, \hat{\delta})$, then $\mathtt{fin}(q)$.*
3. *If $T = \mathcal{A}[\&_{i \in I} ? a_i . U_i^n]^{n \in N}$ then $\& \in T$.*

Proof. By Lemmas G.2, G.3, and G.4. □

Lemma G.2. *Let $M = (Q, q_0, \delta)$, for all $q \in Q$ if $\mathtt{fin}(q)$, then $\& \in \mathcal{T}((Q, q, \delta))$.*

Proof. We prove

$$\forall q \in Q \,:\, \forall R \subseteq Q \,:\, \mathtt{fin}(q, R) \implies \& \in \mathcal{T}((Q, q, \delta))$$

by induction on the (increasing) number of states in R. Note that $R \subset Q$, hence the definition of $\mathtt{fin}(q)$ is indeed well-founded.

Assume $\texttt{fin}(q, R)$.

If $q \nrightarrow$ then $\neg\texttt{fin}(q)$ by definition of $\texttt{fin}(\_)$.

If $q \xrightarrow{?a_i} q_i$, then we have $\mathcal{T}((Q, q, \delta)) = \&_{i\in I}?a_i.\,\mathcal{T}((Q, q_i, \delta))$ and we have the result by definition of $\texttt{fin}(\_)$ and Definition G.4.

If $q \xrightarrow{!a_i} q_i$, then we have $\mathcal{T}((Q, q, \delta)) = \bigoplus_{i\in I}!a_i.\,\mathcal{T}((Q, q_i, \delta))$, and either

- $q \in R$, in which case $\neg\texttt{fin}(q, R)$.
- $q \notin R$, hence by induction hypothesise, we have

$$\texttt{fin}(q_i, R \cup \{q\}) \implies \& \in \mathcal{T}((Q, q_i, \delta)) \qquad \text{Note that } R \subset R \cup \{q\}$$

  and the result follows straightforwardly from Definition G.4. □

We write $fv(T)$ for the set of free variables in T. In the proof below, we abuse the notations slightly and identify recursion variables in type T with states in $\mathcal{M}(T)$, i.e., assuming that a $T = \texttt{rec}\,\texttt{x}.T'$ induces the machine $(Q, \texttt{x}, \delta)$. This way, write, e.g., $R = fv(T)$ for the set of states corresponding the free variables of T.

Lemma G.3. *If* $\& \in T$ *and* $\mathcal{M}(T) = (Q, q_0, \delta)$*, then* $\texttt{fin}(q_0)$.

Proof. We show the following by structural induction on T.

$$\& \in T \ \wedge\ R = fv(T) \subseteq Q \ \wedge\ \mathcal{M}(T) = (Q, q, \delta) \quad \implies \quad \texttt{fin}(q, R)$$

Take T such that $\& \in T$, $R = fv(T)$, and $\mathcal{M}(T) = (Q, q, \delta)$

- If $T = \texttt{end}$, then $\neg(\& \in T)$.
- If $T = \&_{i\in I}?a_i.\,T_i$, then we have $q \xrightarrow{?a_i} q_i$, and thus $\texttt{fin}(q)$ holds, by definition.
- If $T = \texttt{rec}\,\texttt{x}.T'$, then we have $\& \in T'$ by Definition G.4 and there are two sub-cases, either
  - $T' = \&_{i\in I}?a_i.\,T_i$ and we have the result as above, or
  - $T' = \bigoplus_{i\in I}!a_i.\,T_i$ and thus $\& \in T_i$ for each $i \in I$ by Definition G.4. By induction hypothesis, for each $i \in I$, we have:

$$\& \in T_i \ \wedge\ R' = R \cup \{\texttt{x}\} \ \wedge\ \mathcal{M}(T_i) = (Q, q_i, \delta) \quad \implies \quad \texttt{fin}(q_i, R')$$

    Hence, we have $\bigwedge_{\{q_i \,|\, q \xrightarrow{!a_i} q_i\}} \texttt{fin}(q_i, R \cup \{\texttt{x}\})$ as required.
- If $T = \bigoplus_{i\in I}!a_i.\,T_i$, $\& \in T_i$ for each $i \in I$ by Definition G.4. Since we have that $fv(T) = fv(T_i)$ for each $i \in I$ as well, we obtain $\bigwedge_{\{q_i \,|\, q \xrightarrow{!a_i} q_i\}} \texttt{fin}(q_i, R)$ from the induction hypothesis.

□

Lemma G.4. *If* $T = \mathcal{A}[\&_{i\in I}?a_i.\,U_i^n]^{n\in N}$ *and* $\mathcal{A} \neq [\,]$ *then* $\& \in T$.

Proof. Follows from Definition G.4 and the fact that by definition each branch of $\mathcal{A}$ is finite and each branch ends with a hole filled with a $\&_{i\in I}?a_i.T_i$ type. ◻

Lemma G.5. *Let T and U be two session types, such that $\mathcal{M}(T) = (Q^T, q_0^T, \delta^T)$ and $\mathcal{M}(U) = (Q^U, q_0^U, \delta^U)$, the following holds*

$$T \leqslant_{\mathsf{c}} \mathcal{A}[U] \implies \forall \pi \in \mathcal{A} : \pi \blacktriangleright q_0^T \asymp q_0^U$$

Proof. We show the proof by coinduction on $\blacktriangleright\asymp$.

1. If $T = \texttt{end}$, then $\mathcal{A}[U] = \texttt{end}$ and, hence q_0^T and q_0^U are final states and $\pi = \epsilon$. Thus we have $\epsilon \blacktriangleright q_0^T \asymp q_0^U$ by Definition 3.1.
2. If $T = \oplus_{i\in I}!a_i.T_i$, there are two cases depending on the structure of $\mathcal{A}[U]$.
   - If $\mathcal{A}[U] = \&_{i\in I\cup J}?a_i.U_i$, we pose $\mathcal{A} = []$ and $U = \&_{i\in I\cup J}?a_i.U_i$. By definition of $\mathcal{M}(\_)$, we then have that $q_0^T \xrightarrow{!a_i} q_i^T \implies q_0^U \xrightarrow{!a_i} q_i^U$ and we have $\epsilon \blacktriangleright q_i^T \asymp q_i^U$ follows by coinduction hypothesis.
   - If $\mathcal{A}[U] = \mathcal{A}[\&_{i\in I\cup J_n}?a_i.U_i^n]^{n\in N}$, then we have to show that case (3b) of Definition 3.1 applies.
     (a) We have $\texttt{fin}(q_0^T)$ since $T = \oplus_{i\in I}!a_i.T_i$ and $\& \in T_i$, by Lemma G.3.
     (b) We have $\texttt{fin}(q_0^U)$ by using Lemma G.4 (the $\mathcal{A}$ context must be finite) then Lemma G.3.
     (c) By definition of $\mathcal{M}(\_)$, we have $q_0^T \xrightarrow{!a_i} q_i^T$ and $q_0^U \xrightarrow{!a_i} q_i^U$, for all $i \in I$.
     By coinduction hypothesis, we have:
     $$\forall i \in I : T_i \leqslant_{\mathsf{c}} \mathcal{A}[U_i^n]^{n\in N} \implies \forall \hat{\pi} \in \mathcal{A} : \hat{\pi} \blacktriangleright q_i^T \asymp q_i^U$$
     Thus, we see that the structure of U guarantees that its corresponding machine satisfies the property that $q_0^T \xrightarrow{!a_i} q_i^T$ implies that $\forall q_1^U \in Q^U : \forall \pi \in \mathbb{A}^* : q_0^U \xrightarrow{!\pi} q_1^U$, there exist $\pi' \in \mathbb{A}^*$ and $q_i^U \in Q^U$ such that $q_1^U \xrightarrow{!\pi'}\xrightarrow{?a} q_i^U$ and $\pi \cdot \pi' \blacktriangleright q_i^T \asymp q_i^U$; where each $\pi \cdot \pi' \in \mathcal{A}$.
3. If $T = \&_{i\in I\cup J}?a_i.T_i$, then $\mathcal{A}[U] = \oplus_{i\in I}!a_i.U_i$. We can pose $\mathcal{A} = []$ without loss of generality.
   By definition of $\mathcal{M}(\_)$, we then have that $q_0^U \xrightarrow{!a_i} q_i^U \implies q_0^T \xrightarrow{!a_i} q_i^T$ and $\epsilon \blacktriangleright q_i^T \asymp q_i^U$ follows by coinduction hypothesis.

◻

Lemma G.6. *Let $M_i = (Q_i, q_{0_i}, \delta_i)$, $i \in \{1,2\}$ and $\pi = a_1 \cdots a_k \in \mathbb{A}^*$, for all $p \in Q_1$ and $q \in Q_2$, the following holds:*

$$\pi \blacktriangleright p \asymp q \implies \mathcal{T}((Q_1, p, \delta_1)) \leqslant_{\mathsf{c}} !a_1 \cdots !a_k.[\mathcal{T}((Q_2, q, \delta_2))]$$

Proof. By coinduction on the rules of $\leqslant_{\mathsf{c}}$.

1. If $p \not\rightarrow$, then $q \not\rightarrow$ and $\pi = \epsilon$, hence we have $\mathcal{T}((Q_1, p, \delta_1)) = \texttt{end}$ and $\pi[\mathcal{T}((Q_2, q, \delta_2))] = \texttt{end}$.

2. If $p \xrightarrow{?a}$, then there are two cases:
   - if $\pi = \epsilon$, then we have $q \xrightarrow{!b}$ and $\forall a_i \in \mathbb{A} : q \xrightarrow{!a_i} q_i \implies (p \xrightarrow{?a_i} p_i \wedge \epsilon \blacktriangleright p_i \asymp q_i)$ hence we have

$$\mathcal{T}((Q_1, p, \delta_1)) = \&_{i \in I \cup J_n} ?a_i . \mathcal{T}((Q_1, p_i, \delta_1))$$

and

$$\pi[\mathcal{T}((Q_2, q, \delta_2))] = \mathcal{T}((Q_2, q, \delta_2)) = \oplus_{i \in I} !a_i . \mathcal{T}((Q_2, q_i, \delta_2))$$

with $I = \{i \mid q \xrightarrow{!a_i} q_i\}$.
We have the final result by using the coinduction hypothesis, i.e.,

$$\forall i \in I : \epsilon \blacktriangleright p_i \asymp q_i \implies \mathcal{T}((Q_1, p_i, \delta_1)) \leqslant_{\mathsf{c}} \pi[\mathcal{T}((Q_2, q_i, \delta_2))]$$

   - if $\pi = b \cdot \pi'$, then $\exists p_1 \in Q_1 : p \xrightarrow{?b} p_1 \wedge \pi' \blacktriangleright p_1 \asymp q_1$. Hence we have

$$\mathcal{T}((Q_1, p, \delta_1)) = \&_{i \in I} ?a_i . \mathcal{T}((Q_1, p_i, \delta_1)) \quad \text{such that } a_1 = b$$

and

$$\pi[\mathcal{T}((Q_2, q, \delta_2))] = !b . \pi'[\mathcal{T}((Q_2, q, \delta_2))]$$

We have the final result by using the coinduction hypothesis, i.e.,

$$\pi' \blacktriangleright p_1 \asymp q_1 \implies \mathcal{T}((Q_1, p_1, \delta_1)) \leqslant_{\mathsf{c}} \pi'[\mathcal{T}((Q_2, q, \delta_2))]$$

3. If $p \xrightarrow{!a}$, then there are two cases:
   - If $\pi = \epsilon$, and $\forall a_i : p \xrightarrow{!a_i} p_i : \exists q_i \in Q_2 : q \xrightarrow{?a_i} q_i \wedge \epsilon \blacktriangleright p_i \asymp q_i$ hence we have

$$\mathcal{T}((Q_1, p, \delta_1)) = \oplus_{i \in I} !a_i . \mathcal{T}((Q_1, p_i, \delta_1))$$

and

$$\pi[\mathcal{T}((Q_2, q, \delta_2))] = \mathcal{T}((Q_2, q, \delta_2)) = \&_{i \in I \cup J_n} ?a_i . \mathcal{T}((Q_2, q_i, \delta_2))$$

with $I = \{i \mid p \xrightarrow{!a_i} p_i\}$ and the rest follows naturally by coinduction hypothesis.
   - Otherwise, we have $\mathtt{fin}(p)$ and $\mathtt{fin}(q)$, and $\forall a_i : p \xrightarrow{!a_i} p_i : \forall q' \in Q_2 : \forall \pi' \in \mathbb{A}^* : q \xrightarrow{!\pi'} q'$, there exist $\pi'' \in \mathbb{A}^*$ and $q_i \in Q_2$ such that $q' \xrightarrow{!\pi''} \xrightarrow{?a} q_i$ and $\pi \cdot \pi' \cdot \pi'' \blacktriangleright p_i \asymp q_i$.
Since $\mathtt{fin}(q)$ holds, it is possible to build a *finite* context $\mathcal{A}$, i.e., a tree consisting of all the paths $\pi \cdot \pi' \cdot \pi''$ as described above (there is only a finite number of such paths).
Hence, we have

$$\mathcal{T}((Q_1, p, \delta_1)) = \oplus_{i \in I} !a_i . \mathcal{T}((Q_1, p_i, \delta_1))$$

and

$$\pi[\mathcal{T}((Q_2, q, \delta_2))] = \mathcal{A}[\&_{i \in I \cup J_n} ?a_i.\, \mathcal{T}((Q_2, q_i, \delta_2))]$$

By coinduction hypothesis, we have

$$\forall i \in I \,:\, \forall \hat{\pi} \,:\, q \xrightarrow{!\hat{\pi}} q_i \,:\, \mathcal{T}((Q_1, p_i, \delta_1)) \leqslant_{\mathsf{c}} \hat{\pi}[\mathcal{T}((Q_2, q_i, \delta_2))]$$

Since $\mathtt{fin}(p)$, we have $\& \in \mathcal{M}((Q_1, p, \delta_1))$ by Lemma G.2.

□

Theorem 5.4. *The relations $\asymp_s$ and $\leqslant_{\mathsf{s}}$ are equivalent in the following sense:*

1. *Let T_1 and T_2 be two session types, then $T_1 \leqslant_{\mathsf{s}} \overline{T_2} \implies \mathcal{M}(T_1) \asymp_s \overline{\mathcal{M}(T_2)}$.*
2. *Let M_1 and M_2 be two machines, then $M_1 \asymp_s M_2 \implies \mathcal{T}(M_1) \leqslant_{\mathsf{s}} \overline{\mathcal{T}(M_2)}$.*

Proof. The proof (1) and (2) is a sub-case of the proof of Theorem 5.1. □

Theorem 5.5. *If $M_1 \asymp_s \overline{M}$ and $M \asymp_s M_2$, then $M_1 \asymp_s M_2$.*

Proof. Same as the proof of Theorem 5.2, $\leqslant_{\mathsf{s}}$ is transitive, using Theorem 5.4. □