Bocchi, Laura, Orchard, Dominic A. and Voinea, Laura ***A theory of protocol composition.*** **Technical report. NA 10.48550/arXiv.2203.02461 <https://doi.org/10.485**
(Unpublished)

A compositional theory of protocol engineering

Laura Bocchi, Dominic Orchard, and Laura Voinea

University of Kent, UK

Abstract. Real-world communication protocols are often built out of a number of simpler protocols that cater for some specific functionality (e.g., banking, authentication). However much of the formal definitions of protocols used for program verification treat protocols as monolithic units. Composition is considered for implementations of a protocol, but not for the protocols themselves as engineering components. We propose primitives and techniques for the modular composition of protocols. Our notion of composition defines an interleaving of two or more protocols in a way that satisfies user-specified context-dependent constraints which serve to explain "contact points" between the protocols. The resulting approach gives a theoretical basis for protocol (re-)engineering based on a process calculus with constraint annotations. We have implemented our approach as a tool for Erlang that supports generation of protocol compositions with formal guarantees, and code generation/extraction.

Keywords: Process-calculi, Distributed protocols, Protocol engineering

1 Introduction

Protocols are everywhere. Whenever two entities need to communicate, a protocol can be used to ensure that both parties effectively exchange information. Protocols can be seen as a *specification* of communication, and as such have been leveraged for the purposes of verification in programming languages, e.g., session types [18,19,8,20], choreographies [10,11,28], typestate [31], behavioural types in general [21,17], and more. There may be many protocols that a program has to conform to, capturing different interactions between different parts of a system. Here we use the term *protocol* to denote a specification of the interaction patterns between different system components. For example, when considering distributed systems, a protocol may describe the causalities and dependencies of the communication between processes. To give a more concrete intuition, an informal specification of a protocol for an e-banking system may be as follows: *The banking server repeatedly offers a menu with three options: (1) request a banking statement, which is sent back by the server, (2) request a payment, after which the client will send payment data, or (3) terminate the session.* We elaborate on this example later, using it as a motivating example.

Much of the work on systematising the process of programming against a specification assumes a monolithic view of protocols: a protocol is often given for the entire system, explaining the communication between all parties involved.

This up-front, single point of definition runs contrary to the human aspects of real-world programming, in which a programmer gradually pieces together their code, perhaps heavily leveraging libraries, to reach their intended goal; programs are gradual *compositions.* A view that is globally defined once does not reflect the real process of software composition. In contrast, a view that defines lots of local protocols or sub-protocols places the burden of configuring their interaction on the programmer: programmers must themselves work in a situation where they have to consider many smaller protocols and work out how they want dependencies between them to be resolved. Instead, a flexible, non-monolithic notion of composition (and possibly recomposition, when a piece of code is refactored and rewritten, or reused) is needed to support the engineering of protocol-dependent code. Ideally, such a notion should support well-founded semi-automated protocol composition and implementation with formal guarantees.

This work lays a foundation for compositional protocol engineering based on a notion of *interleaving composition* of protocols. An interleaving composition of two protocols 'weaves' them together into a single unified protocol, differing from a sequential composition in which one protocol follows the other, or one's inputs are coupled to the other's outputs. We address, in general terms, the question of what a correct protocol composition is, and introduce a syntactic definition of composition that characterises finite sets of *correct* interleaving compositions, each representing a 'good way' to interleave the component protocols with respect to domain-specific user-specified constraints. The resulting approach gives a theoretical basis for protocol (re-)engineering based on a process calculus with constraint annotations. Interleaving composition has the purpose of enhancing the awareness (of engineers and programmers) of what a protocol means, as well as facilitating the reasoning about its properties. We give an algorithmic implementation of interleaving composition supporting the process of defining protocols and inspecting the generated compositions, and code generation for Erlang, producing skeletons of processes following a given protocol (composite or not). Code generation is based on Erlang/OTP `gen_statem` behaviour [1] allowing code to be automatically migrated in subsequent compositions and re-engineering. Correspondence of our protocol language with Finite State Machines (FSM) (via directed graphs) yields a straightforward link between protocols and FSM-structured code.

A related line of work is that of automata compositions. Team Automata, introduced in [7,16], provide several means of composing machines via synchronization on their common actions, and give a formal framework for composition. Unlike Team Automata, we express composition constraints orthogonally to communication: instead of synchronization on common actions, we use 'asserts'/'requires' as asymmetric contact points for composition, and reason about the properties of a composite protocol from the perspective of the application logic. The resulting composition relation given in this work is not characterizable as one of the synchronizations of Team Automata (discussed further in Section 7). Another related line of work defines composition as run-time weaving, for example applying principles of aspect-oriented programming to protocol

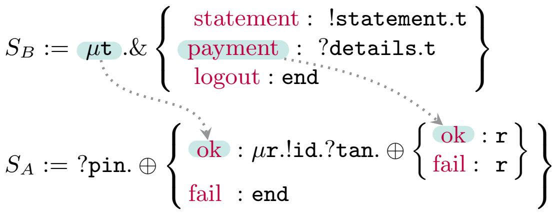

Fig. 1. Banking (S_B) and PIN/TAN authentication (S_A) protocols.

composition [32]. Unlike the aforementioned work, we *statically* derive protocol compositions that enable (human/automated) reasoning and verification of their properties.

Motivating example Consider the banking protocol discussed earlier in this section. The banking protocol can be formally specified as S_B in Figure 1 using a process calculus notation. S_B repeatedly (via a fixed point μt) offers (denoted &) three options: option statement is followed by a send action (denoted !) of a message with the bank statement, option payment is followed by a receive action (denoted ?) with details of the payment, and option logout is followed by termination of the protocol (denoted end). After each of the first two options, the control flow goes back to the initial state (via t).

Assume now that we want to extend S_B with two-level authentication: one level for accessing the service and one additional level for each payment transaction. Concretely, we wish to compose S_B with the PIN/TAN (Personal Identification Number/Transaction Authentication Number) protocol modelled in Figure 1 as S_A which offers two-stage authentication. The first stage is pin authentication: the server receives a pin and decides (⊕) whether to continue (i.e., ok) or terminate (i.e., fail). If ok is chosen, the protocol enters a loop (i.e., μr) that manages multiple TAN authentications, supporting multiple transactions requiring an additional level of security. In the loop, the server sends an identifier id for which the client must send back a tan. The server notifies the client about the correctness of the tan with either ok or fail.

We want to compose the banking and authentication protocols into one single protocol where the actions of the two protocols follow a specific interleaving: access to the banking service requires a PIN authentication, and each payment instance/iteration requires an extra TAN authentication (see dotted arrows in Figure 1). This specific interleaving entails an authorization property, which we later express and ensure by using assertion annotations. Moreover, we want tools that facilitate (re-)engineering of programs implementing interleaving compositions. For example, we want to obtain a skeleton implementation for the banking and PIN/TAN protocol, and in a second stage we want to reuse the code when composing banking with a different multi-factor authentication protocol, e.g., offering other options besides TAN, such as keycard authentication.

Contributions Central to our work is our definition of interleaving composition. We use a process-calculus-based notation for protocols with 'assertions' that specify contact points and constraints between component protocols (Section 2). Interleaving composition is defined relationally as there may be many possible valid interleaved protocols (or even none) (Section 3). In Section 4, we prove that our composition relation returns *correct* interleaving compositions. Correctness comprises three properties: (1) *behaviour preservation* (Theorem 1): interleaving compositions only perform sequences of actions that may be performed by either of the component protocols, (2) *fairness* (Theorem 3): interleaving compositions eventually execute the next available action of each protocol, and (3) *well-assertedness* (Proposition 3): interleaving compositions always satisfy requirements prescribed by the assertions in the protocols being composed. Thus, we establish that the composition relation produces sets of correct-by-construction protocol compositions. In Section 3.1 we provide two less restrictive definitions of interleaving composition with the use of two additional rules, *weak branching* and *correlating branching* that are able to capture a larger number of scenarios but enjoy a weaker fairness property (Theorem 2).

In Section 5, we introduce a tool to aid code reuse and modularisation in Erlang. The tool implements interleaving composition, and provides code generation as well as protocol extraction from existing code. For instance, starting with two `gen_statem` separated implementations of the authentication and banking protocols, we can extract their underlying protocols (together with any assertion constraints they may have), automatically compose the extracted protocols and generate a new Erlang module.

In Section 6, we discuss instantiating our protocol language into well-known formalisms such as CCS [27] and Session Types [19,8,20], and illustrate possible synergies. Section 7 discusses related work.

2 Asserted Protocols

We introduce a simple language of protocols to abstractly capture essential features of sequential computation: sequencing, choice, and looping. Our protocol language somewhat resembles Milner's CCS [27] or the π-calculus [30] (but without parallel composition or name restriction) and has some relation to Kleene algebras [24] but we provide more general patterns of recursion via recursive binders rather than a single closure operator.

Generally, two protocols can be composed in several ways, each reflecting a possible interleaving of the actions of the two protocols. Not all such interleavings are meaningful depending on the scenario or domain. The protocol language therefore includes a notion of 'assertions' which can be used to capture the behavioural constraints of a protocol to guide interleaving composition in a meaningful way; they act as a specification of minimal 'contact points' between protocols akin to pre- and post-conditions.

Following an explanation of the syntax and various examples, we give an operational model to the protocol language which serves to explain both the program semantics which it abstracts, and the meaning of the assertion actions.

Definition 1 (Asserted protocols). *Asserted protocols, or just* protocols *for short, are ranged over by S and are defined as followsing syntax rules:*

$$\begin{array}{rcl}
S & ::= & \begin{array}{ll} p.S & \textit{action prefix} \end{array} \\
 & \mid & \begin{array}{ll} +\{\mathrm{l_i} : S_i\}_{i\in I} & \textit{branching} \end{array} \\
 & \mid & \begin{array}{ll} \mu\mathtt{t}.S & \textit{fixed-point} \end{array} \\
 & \mid & \begin{array}{ll} \mathtt{t} & \textit{recursive variable} \end{array} \\
 & \mid & \begin{array}{ll} \mathtt{end} & \textit{end} \end{array} \\
 & & \left.\begin{array}{lll} \mid & \mathtt{assert}(n).S & \textit{assert (produce)} \\ \mid & \mathtt{require}(n).S & \textit{require} \\ \mid & \mathtt{consume}(n).S & \textit{consume} \end{array}\right\}\text{ assertion fragment}
\end{array}$$

where $p \in \mathcal{P}$ ranges over prefixing actions, $\mathrm{l} \in \mathcal{L}$ ranges over labels used to label each branch of the n-ary branching construct, `t` *ranges over protocol variables for recursive protocol definitions, $n \in \mathcal{N}$ ranges over names of logical atoms used by assertions. The sets of actions $\mathcal{P}$, labels $\mathcal{L}$, and names $\mathcal{N}$ are parameters to the language and thus can be freely chosen. Furthermore $+$ ranges over a set of operators $\mathcal{O}$ used to represent branching choice and thus can also be instantiated.*

The prefixing action provides sequential composition (in the style of process calculi). Branching is n-ary, taking the form of a set of protocol choices with a label $\mathrm{l_i}$ for each choice. Looping behaviour is captured via the recursive protocol variable binding $\mu\mathtt{t}$, which respects the usual rules of binders, and recursion variables `t`. We assume variables to be guarded in the standard way (they only occur under actions or branching). Unless otherwise stated, we consider protocols to be closed with respect to these recursion variables.

Protocols can be annotated with assertions to introduce guarantees $\mathtt{assert}(n)$, requirements $\mathtt{require}(n)$, and linear requirements $\mathtt{consume}(n)$: $\mathtt{assert}(n)$ introduces a true logical atom n into the scope of the following protocol, $\mathtt{require}(n)$ allows the protocol to proceed only if n is in scope, and $\mathtt{consume}(n)$ removes the truth of logical atom n from the scope of the following protocol.

Remark 1 (Language instantiation). The protocol language can be instantiated to model different protocol languages. In the examples we often instantiate the prefixing actions $\mathcal{P}$ to sends $!\mathtt{T}$ and receives $?\mathtt{T}$ capturing interaction with some other concurrent program, i.e., $p \in \{!\mathtt{T}, ?\mathtt{T}\}$ where `T` is a type (e.g., booleans, integers, strings), and instantiate choice $+$ to a pair of *polarised choice* operators: $+ \in \{\oplus, \&\}$, either offering of a choice $\oplus$ or selecting from amongst some choices $\&$. This yields a session types-like syntax like the one of Dardha et al. [14]. Different instantiations are possible, as shown in Section 6.

Examples often colour assertions green and labels purple for readability.

2.1 Assertion examples

Consider a payment process $?\texttt{pay.end}$ that receives a payment and terminates, and a dispatch process $!\texttt{item.end}$ that sends a product link and terminates. We can interleave these two protocols in two ways: $?\texttt{pay}.!\texttt{item.end}$ (payment first) or $!\texttt{item}.?\texttt{pay.end}$ (dispatch first). By using assertions, we can require that payment happens before dispatch: below, I_1 asserts the logical atom *paid* as a post-condition to receiving payment while in I_2 the sending action depends on the logical atom *paid* as a pre-condition, and in doing so consumes it.

$$I_1 = ?\texttt{pay}.\texttt{assert}(\mathit{paid}).\texttt{end} \qquad I_2 = \texttt{consume}(\mathit{paid}).!\texttt{item.end}$$

The only interleaving composition of I_1 and I_2 that satisfies the constraints posed by the assertions is:

$$?\texttt{pay}.\texttt{assert}(\mathit{paid}).\texttt{consume}(\mathit{paid}).!\texttt{item.end}$$

The exact definition of *well-assertedness*, that specified constraints are satisfied in all of the protocol's executions, will be given later in this section (Definition 6).

Linear constraints model guarantees that can be used only once. For example, the scenario "*in a recursive payment/dispatch scenario there is one dispatch for each one payment*" can be modelled by recursive payment $\mu\texttt{t}.?\texttt{pay}.\texttt{assert}(\mathit{paid}).\texttt{t}$ and recursive dispatch $\mu\texttt{r}.\texttt{consume}(\mathit{paid}).!\texttt{item.r}$ protocols. Non-linear constraint $\texttt{require}(n)$ does not consume guarantees. It can express, e.g., that at a prepaid buffet, payment remains valid (hungry) until the meal ends (end):

$$\mu\texttt{t}.\&\{\mathsf{hungry} : \texttt{require}(\mathit{paid}).!\texttt{food.t},\ \mathsf{end} : \texttt{consume}(\mathit{paid}).\texttt{end}\}$$

Example 1 (Asserted banking and authentication protocols). Returning to the banking and PIN/TAN example, the informal requirement discussed in the introduction can be modelled using assertions. An asserted version of the banking protocol, given below as S'_B, uses $\texttt{require}(\mathit{pin})$ to ensure a successful PIN authentication before accessing the banking menu; $\texttt{consume}(\mathit{tan})$ to require one successful TAN authentication for each iteration involving a payment; and $\texttt{consume}(\mathit{pin})$ to remove the PIN guarantee when logging out. Assertions $\texttt{assert}(\mathit{pay})$ and $\texttt{consume}(\mathit{pay})$ ensure TAN authentication only happens in case of payment.

$$S'_B = \texttt{require}(\mathit{pin}).\mu\texttt{t}.\&\left\{\begin{array}{ll}\mathsf{statement}: & !\texttt{statement.t}\\ \mathsf{payment}: & \texttt{assert}(\mathit{pay}).\texttt{consume}(\mathit{tan}).?\texttt{details.t}\\ \mathsf{logout}: & \texttt{consume}(\mathit{pin}).\texttt{end}\end{array}\right\}$$

In the asserted authentication protocol S'_A below, $\texttt{assert}(\mathit{pin})$ and $\texttt{assert}(\mathit{tan})$ provide guarantees of successful PIN and TAN authentication, respectively:

$$S'_A = ?\texttt{pin}.\oplus\left\{\begin{array}{l}\mathsf{ok}\colon\ \texttt{assert}(\mathit{pin}).\mu\texttt{r}.\texttt{consume}(\mathit{pay}).!\texttt{id}.?\texttt{tan}.\oplus\left\{\begin{array}{l}\mathsf{ok}\colon\ \texttt{assert}(\mathit{tan}).\texttt{r}\\ \mathsf{fail}\colon \texttt{r}\end{array}\right\}\\ \mathsf{fail}\colon \texttt{end}\end{array}\right\}$$

2.2 Protocols semantics

The semantics of a protocol is given in Definition 3 in terms of an environment that keeps track of guarantees, and lets protocols progress only if stated guarantees can be met by the environment. The semantics is up to the (standard) structural equivalence rules given in Definition 2 where $S[\mu\mathtt{t}.S/\mathtt{t}]$ is the one-time unfolding of $\mu\mathtt{t}.S$ (all occurrences of $\mathtt{t}$ are substituted with $\mu\mathtt{t}.S$).

Definition 2 (Structural congruence). *Protocols have a structural congruence relation $\equiv$ (used in reasoning and the proofs for this paper) where:*

$$\mu\mathtt{t}.\mu\mathtt{t}'.S \equiv \mu\mathtt{t}'.\mu\mathtt{t}.S \qquad \mu\mathtt{t}.S \equiv S \ (\textit{where } \mathtt{t} \notin \mathsf{fv}(S)) \qquad \mu\mathtt{t}.S \equiv S[\mu\mathtt{t}.S/\mathtt{t}]$$

Definition 3 (Operational semantics). *The semantics of protocols is defined by a labelled-transition system (LTS) over configurations of the form (A, S) where A ranges over 'environments': sets of logical atoms (i.e., $A \subseteq \mathcal{N}$), with transition labels $\ell ::= p \mid +\mathrm{l} \mid \mathtt{assert}(n) \mid \mathtt{require}(n) \mid \mathtt{consume}(n)$ and the transition rules below:*

$$(A, p.S) \xrightarrow{p} (A, S) \qquad \langle\mathtt{Inter}\rangle$$

$$(A, +\{\mathrm{l_i} : S_i\}_{i\in I}) \xrightarrow{+\mathrm{l_j}} (A, S_j) \qquad (j \in I) \qquad \langle\mathtt{Branch}\rangle$$

$$(A, \mathtt{assert}(n).S) \xrightarrow{\mathtt{assert}(n)} (A \cup \{n\}, S) \qquad \langle\mathtt{Assert}\rangle$$

$$(A, \mathtt{require}(n).S) \xrightarrow{\mathtt{require}(n)} (A, S) \qquad (n \in A) \qquad \langle\mathtt{Require}\rangle$$

$$(A, \mathtt{consume}(n).S) \xrightarrow{\mathtt{consume}(n)} (A \setminus \{n\}, S) \qquad (n \in A) \qquad \langle\mathtt{Consume}\rangle$$

$$\frac{(A, S) \xrightarrow{\ell} (A', S')}{(A, \mu\mathtt{t}.S) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}.S/\mathtt{t}])} \qquad \langle\mathtt{Rec}\rangle$$

Rules $\langle\mathtt{Inter}\rangle$ and $\langle\mathtt{Branch}\rangle$ always allow a protocol to proceed with some action, resulting in the appropriate continuation, without any effect to the environment. Rule $\langle\mathtt{Assert}\rangle$ adds atom n to the environment. Rules $\langle\mathtt{Require}\rangle$ and $\langle\mathtt{Consume}\rangle$ both require the presence of atom n in the environment for the protocol to continue. Although $\langle\mathtt{Require}\rangle$ leaves the environment unchanged, $\langle\mathtt{Consume}\rangle$ consumes the atom n from the environment. In $\langle\mathtt{Rec}\rangle$, $S'[\mu\mathtt{t}.S/\mathtt{t}]$ means that the recursive protocol is unfolded by substituting $\mu\mathtt{t}.S$ for $\mathtt{t}$ in S'.

We write: $(A, S) \not\rightarrow$ if $(A, S) \xrightarrow{\ell} (A', S')$ for no ℓ, A', S'; $(A, S) \xrightarrow{\boldsymbol{\ell}} (A', S')$ for a vector $\boldsymbol{\ell} = \ell_1, \ldots, \ell_n$ if $(A, S) \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_n} (A', S')$. We say that (A', S') is *reachable* from (A, S) if $(A, S) = (A', S')$ or $(A, S) \xrightarrow{\boldsymbol{\ell}} (A', S')$ for a vector $\boldsymbol{\ell}$. We omit labels and target states where immaterial.

Definition 4 (Stuck state). *State (A, S) is* stuck *if $S \not\equiv \mathtt{end}$ and $(A, S) \not\rightarrow$.*

Definition 5 (Progress). *A protocol S enjoys* progress *if every state (A', S') reachable from $(\emptyset, S)$ is not stuck.*

A protocol may reach a stuck state when it does not have sufficient preconditions in its environment A. In Example 1, S'_B does not enjoy progress because the pre-condition expressed by $\mathtt{require}(\mathit{pin})$ cannot be met; similarly, S'_A does not enjoy progress because of unmet pre-condition $\mathtt{consume}(\mathit{pay})$.

2.3 Well-assertedness

The assertions are key to generating meaningful compositions of protocols. Following the labelled transitions semantics, we define a judgment which captures the pre- and post-conditions of a protocol implied by its assertions. We use the notation $A\ \{S\}\ A'$ reminiscent of a Hoare triple where A and A' are pre- and post- conditions of S respectively.

Definition 6 (Well-assertedness). *Let A be a set of names.* Well-assertedness *of a protocol S with respect to A is defined below, as an inference system on judgments of the form $A\ \{S\}\ A'$, where A' is the set of names (logical atoms) resulting after the execution of S given the set of names A.*

$$\frac{A\ \{S\}\ A'}{A\ \{p.S\}\ A'}\,[\text{act}] \qquad \frac{\forall i \in I.\ A\ \{S_i\}\ A_i}{A\ \{+\{\mathrm{l_i} : S_i\}_{i\in I}\}\ \bigcap_{i\in I} A_i}\,[\text{bra}] \qquad \frac{A \cup \{n\}\ \{S\}\ A'}{A\ \{\mathtt{assert}(n).S\}\ A'}\,[\text{assert}]$$

$$\frac{A \cup \{n\}\ \{S\}\ A'}{A \cup \{n\}\ \{\mathtt{require}(n).S\}\ A'}\,[\text{require}] \qquad \frac{A \setminus \{n\}\ \{S\}\ A' \quad n \in A}{A\ \{\mathtt{consume}(n).S\}\ A'}\,[\text{consume}]$$

$$\frac{A\ \{S\}\ A \cup A'}{A\ \{\mu\mathtt{t}.S\}\ A \cup A'}\,[\text{rec}] \qquad \frac{-}{A\ \{\mathtt{end}\}\ A}\,[\text{end}] \qquad \frac{-}{A\ \{\mathtt{t}\}\ A}\,[\text{call}]$$

We write $A\ \{S\}$ when $A\ \{S\}\ A'$ for some A' (i.e., when the post-condition is not of interest). We say that S is very-well-asserted *if $\emptyset\ \{S\}$. We say that a state (A, S) is well-asserted if S is well-asserted with respect to A.*

Protocols S'_A and S'_B in Example 1 are not very-well-asserted but they are well-asserted with respect to $\{pin, tan\}$ and $\{pay\}$, respectively.

We now consider some properties of well-asserted protocols. Proofs are in Appendix C. Firstly, protocols that do not contain assertions are very-well-asserted:

Proposition 1 (Very-well-assertedness) *If S is generated by the grammar in Definition 1 without the assertion fragment then it is very-well-asserted.*

Next, well-asserted protocols can have their environment weakened, akin to pre-condition weakening in Hoare logic:

Proposition 2 (Environment weakening) *If $A\ \{S\}$ and $A \subseteq A'$ then $A'\ \{S\}$. Hence, $\emptyset\ \{S\}$ implies $A\ \{S\}$ for all A.*

Next, Lemma 1 states that the redux of a well-asserted state is well-asserted, moreover the postconditions are not weakened by reduction:

Lemma 1 (Reduction preserves well-assertedness). *If $A\ \{S\}\ A'$ and there is a reduction $(A, S) \xrightarrow{\ell} (A'', S')$ then $\exists A''' \supseteq A'.\ A''\ \{S'\}\ A'''$.*

Lemma 2 (Well-asserted protocols are not stuck). *If $A\ \{S\}$ and S is closed with respect to recursion variables ($\mathsf{fv}(S) = \emptyset$) then (A, S) is not stuck.*

Next, Lemma 3 shows that if a protocol “gets stuck”, this is because it does not have enough preconditions to proceed. Thus, the protocol needs assumptions that may be provided by other protocols it could be composed with.

Lemma 3 (Progress of very-well-asserted protocols). *If S is very-well-asserted (i.e., $\emptyset\ \{S\}$) and closed then it exhibits* progress*.*

Lemma 3 follows by induction on the length of a protocol’s execution, combined with Lemmas 1 and 2.

We next introduce protocol composition, which produces protocols that are meaningful with respect to their assertions (i.e., that exhibit progress).

3 Interleaving Compositions

We compose protocols by computing syntactic interleavings. We derive the ‘interleaving composition’ of two protocols S_1 and S_2 via a relation with judgments of the form: $T_L, T_R, A, S_1 \circ S_2 \vdash S$ where S is the resulting composed protocol, and A is the set of names (i.e., assertions) provided by the environment to S. Environments T_L and T_R are sets of protocol variables that are free in S_1 and S_2 respectively, and are used to handle composition of recursive protocols. The composition relation is illustrated by examples after its definition in Definition 7.

Definition 7 (Interleaving composition). *Let* $\mathtt{Top}(\mu\mathtt{t}.S) = \{\mathtt{t}\}$ *and* $\mathtt{Top}(S) = \emptyset$ *for all other cases of S. Interleaving composition is defined as follows.*

$$\frac{T_L, T_R, A, S_1 \circ S_2 \vdash S}{T_L, T_R, A, p.S_1 \circ S_2 \vdash p.S} \qquad \frac{T_R, T_L, A, S_2 \circ S_1 \vdash S}{T_L, T_R, A, S_1 \circ S_2 \vdash S} \qquad \text{[act/sym]}$$

$$\frac{T_L, T_R, A \cup \{n\}, S_1 \circ S_2 \vdash S}{T_L, T_R, A \cup \{n\}, \mathtt{require}(n).S_1 \circ S_2 \vdash \mathtt{require}(n).S} \qquad \text{[require]}$$

$$\frac{T_L, T_R, A \setminus \{n\}, S_1 \circ S_2 \vdash S \qquad n \in A}{T_L, T_R, A, \mathtt{consume}(n).S_1 \circ S_2 \vdash \mathtt{consume}(n).S} \qquad \text{[consume]}$$

$$\frac{T_L, T_R, A \cup \{n\}, S_1 \circ S_2 \vdash S}{T_L, T_R, A, \mathtt{assert}(n).S_1 \circ S_2 \vdash \mathtt{assert}(n).S} \qquad \text{[assert]}$$

$$\frac{\forall i \in I \quad T_L, T_R, A, S_i \circ S_2 \vdash S'_i}{T_L, T_R, A, +\{\mathrm{l_i} : S_i\}_{i \in I} \circ S_2 \vdash +\{\mathrm{l_i} : S'_i\}_{i \in I}} \qquad \text{[bra]}$$

$$\frac{T_L \cup \{\mathtt{t}_1\}, T_R, A, S_1 \circ \mu\mathtt{t}_2.S_2 \vdash S \qquad A\ \{\mu\mathtt{t}_1.S\} \qquad \mathtt{Top}(S_1) = \emptyset}{T_L, T_R, A, \mu\mathtt{t}_1.S_1 \circ \mu\mathtt{t}_2.S_2 \vdash \mu\mathtt{t}_1.S} \qquad \text{[rec1]}$$

$$\frac{T_L, T_R, A, S_1[\mathtt{t}/\mathtt{t}_1] \circ S_2 \vdash S \qquad \mathtt{t} \in T_R \qquad \mathtt{Top}(S_1) = \emptyset}{T_L, T_R, A, \mu\mathtt{t}_1.S_1 \circ S_2 \vdash S} \qquad \text{[rec2]}$$

$$\frac{A\ \{\mu\mathtt{t}.S\} \qquad \mathsf{fv}(\mu\mathtt{t}.S) = \emptyset}{T_L, T_R, A, \mu\mathtt{t}.S \circ \mathtt{end} \vdash \mu\mathtt{t}.S} \qquad \text{[rec3]}$$

$$\frac{\mathtt{t} \in T_L \vee \mathtt{t} \in T_R}{T_L, T_R, A, \mathtt{t} \circ \mathtt{t} \vdash \mathtt{t}} \qquad \frac{-}{T_L, T_R, A, \mathtt{end} \circ \mathtt{end} \vdash \mathtt{end}} \qquad \text{[call/end]}$$

Rule [act] is for prefixes, [sym] is the commutativity rule, and [end] handles a terminated protocol. By combining [act] and [sym] one can obtain all interleavings of two sequences of actions. Rule [require] includes the continuation of a protocol only if a required assertion n is provided by the environment. Rule [consume] is similar except the assertion is removed in the precondition's environment. Conversely, [assert] adds assertion n to the environment of the precondition. Rules [require], [assume], and [consume] enforce a particular order in actions of an interleaving, seen in the next example.

Example 2 (Composition with assertions). Section 2.1 informally discussed the composition of $I_1 = ?\mathtt{pay}.\mathtt{assert}(p).\mathtt{end}$ and $I_2 = \mathtt{consume}(p).!\mathtt{item}.\mathtt{end}$ which produces only one possible interleaving given by the derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{end}\circ\mathtt{end}\vdash\mathtt{end}}\,[\text{end}]}{\emptyset,\emptyset,\ \emptyset,\ !\mathtt{item}.\mathtt{end}\circ\mathtt{end}\vdash !\mathtt{item}.\mathtt{end}}\,[\text{act}]}{\emptyset,\emptyset,\ \{p\},\ \mathtt{consume}(p).!\mathtt{item}.\mathtt{end}\circ\mathtt{end}\vdash\mathtt{consume}(p).!\mathtt{item}.\mathtt{end}}\,[\text{consume}]}{\emptyset,\emptyset,\ \{p\},\ \mathtt{end}\circ I_2\vdash\mathtt{consume}(p).!\mathtt{item}.\mathtt{end}}\,[\text{sym}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{assert}(p).\mathtt{end}\circ I_2\vdash\mathtt{assert}(p).\mathtt{consume}(p).!\mathtt{item}.\mathtt{end}}\,[\text{assert}]}{\emptyset,\emptyset,\ \emptyset,\ ?\mathtt{pay}.\mathtt{assert}(p).\mathtt{end}\circ I_2\vdash ?\mathtt{pay}.\mathtt{assert}(p).\mathtt{consume}(p).!\mathtt{item}.\mathtt{end}}\,[\text{act}]$$

Rule [bra] is similar to [act] but the continuations are composed with each branch. For example the composition $+\{\mathrm{l}_1 : \mathtt{end},\ \mathrm{l}_2 : \mathtt{end}\} \circ !\mathtt{Int}.\mathtt{end}$ with initially empty environment produces the following two interleavings:

$+\{\mathrm{l}_1 : !\mathtt{Int}.\mathtt{end},\ \mathrm{l}_2 : !\mathtt{Int}.\mathtt{end}\}$ (applying [bra], [sym], [act], [end])
$!\mathtt{Int}. + \{\mathrm{l}_1 : \mathtt{end},\ \mathrm{l}_2 : \mathtt{end}\}$ (applying [sym], [act], [act], [sym], [bra], [end])

Rule [end] requires both protocols to be terminated. Rules [rec1] and [rec2] allow two recursive protocols to be composed. When composing two recursive protocols, say $\mu\mathtt{t}_1.S_1$ and $\mu\mathtt{t}_2.S_2$, rules [rec1] and [rec2] both contribute to merging the two recursion bodies into one, associated to one protocol variable, either $\mathtt{t}_1$ or $\mathtt{t}_2$. More precisely, rule [rec1] picks $\mathtt{t}_1$ as name for the interleaving composition, records $\mathtt{t}_1$ into environment T_L and continues with the composition of the recursion body S_1 with $\mu\mathtt{t}_2.S_2$. Assumption $\mathtt{Top}(S_1) = \emptyset$ rules out protocols of the form $\mu\mathtt{t}.\mu\mathtt{t}'.S$.[1] The premise $A\ \{\mu\mathtt{t}_1.S\}$ ensures well-assertedness of the arbitrary repetition of S, that is $\mu\mathtt{t}_1.S$ (the composition rules would only check that S is well-asserted). Rule [rec2] completes the merge of two recursions, with $\mathtt{t}_1$ here being a second recursion variable merged to $\mathtt{t} \in T_R$. In the premise, all calls to $\mathtt{t}_1$ are redirected to $\mathtt{t}$ (via a substitution). Again, for simplicity and with no loss of generality $\mathtt{Top}(S_1) = \emptyset$. To understand [rec1], [rec2], consider a derivation of two recursive protocols $\mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1$ and $\mu\mathtt{t}_2.!\mathtt{p}_2.\mathtt{t}_2$:

[1] This assumption simplifies the theory with no loss of generality, as $\mu\mathtt{t}.\mu\mathtt{t}'.S$ is behaviourally equivalent to $\mu\mathtt{t}.S[\mathtt{t}/\mathtt{t}']$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{t}_1 \in \{\mathtt{t}_1\}}{\emptyset, \{\mathtt{t}_1\}, \emptyset,\ \mathtt{t}_1 \circ \mathtt{t}_1 \vdash \mathtt{t}_1}\,[\text{call}]}{\emptyset, \{\mathtt{t}_1\}, \emptyset,\ !\mathtt{p}_2.\mathtt{t}_1 \circ \mathtt{t}_1 \vdash\ !\mathtt{p}_2.\mathtt{t}_1}\,[\text{act}]}{\emptyset, \{\mathtt{t}_1\}, \emptyset,\ \mu\mathtt{t}_2.!\mathtt{p}_2.\mathtt{t}_2 \circ \mathtt{t}_1 \vdash\ !\mathtt{p}_2.\mathtt{t}_1}\,[\text{rec2}]}{\{\mathtt{t}_1\}, \emptyset, \emptyset,\ \mathtt{t}_1 \circ \mu\mathtt{t}_2.!\mathtt{p}_2.\mathtt{t}_2 \vdash\ !\mathtt{p}_2.\mathtt{t}_1}\,[\text{sym}]}{\{\mathtt{t}_1\}, \emptyset, \emptyset,\ !\mathtt{p}_1.\mathtt{t}_1 \circ \mu\mathtt{t}_2.!\mathtt{p}_2.\mathtt{t}_2 \vdash\ !\mathtt{p}_1.!\mathtt{p}_2.\mathtt{t}_1}\,[\text{act}]}{\emptyset, \emptyset, \emptyset,\ \mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1 \circ \mu\mathtt{t}_2.!\mathtt{p}_2.\mathtt{t}_2 \vdash \mu\mathtt{t}_1.!\mathtt{p}_1.!\mathtt{p}_2.\mathtt{t}_1}\,[\text{rec1}]$$

Thus the end result is a protocol with just one recursion.

The composition of a recursive protocol with a non-recursive one is delicate. An approach to composition that is too permissive could generate interleaving compositions that violate behaviour preservation and fairness. These properties will be formally introduced later, but we offer an intuition here so that the reader can understand some of our design choices for Definition 7.

The intuition of behaviour preservation is: *'All executions allowed by an interleaving composition preserve the interaction structures of each component protocol that comprises it'*. When composing a recursive protocol with a non-recursive one e.g., $S_1 = \mu\mathtt{t}.!\mathtt{p}_1.\mathtt{t}$ with $S_2' = !\mathtt{p}_2.\mathtt{end}$, then we would *not* want to derive the following protocol: $S = \mu\mathtt{t}.!\mathtt{p}_1.!\mathtt{p}_2.\mathtt{t}$. This allows e.g., execution $!\mathtt{p}_1, !\mathtt{p}_2, !\mathtt{p}_1, !\mathtt{p}_2$ where action $!\mathtt{p}_2$ is repeatedly executed (while S_2 only prescribes one instance of $!\mathtt{p}_2$) hence violating behaviour preservation (Theorem 1). Our rules do not allow S above to be derived from S_1 and S_2', thanks to rule [call] checking that the protocols being composed are indeed recursive protocols that have been correctly merged (i.e., share a recursion variable $\mathtt{t}$). Below, for illustration purposes, we use ☹ to denote 'non-derivable protocol':

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\mathtt{t}\}, \emptyset, \emptyset,\ \mathtt{end} \circ \mathtt{t} \vdash \frownie}{\emptyset, \{\mathtt{t}\}, \emptyset,\ !\mathtt{p}_2.\mathtt{end} \circ \mathtt{t} \vdash\ !\mathtt{p}_2.\frownie}\,[\text{act}]}{\{\mathtt{t}\}, \emptyset, \emptyset,\ \mathtt{t} \circ !\mathtt{p}_2.\mathtt{end} \vdash\ !\mathtt{p}_2.\frownie}\,[\text{sym}]}{\{\mathtt{t}\}, \emptyset, \emptyset,\ !\mathtt{p}_1.\mathtt{t} \circ !\mathtt{p}_2.\mathtt{end} \vdash\ !\mathtt{p}_1.!\mathtt{p}_2.\frownie}\,[\text{act}]}{\emptyset, \emptyset, \emptyset,\ \mu\mathtt{t}.!\mathtt{p}_1.\mathtt{t} \circ !\mathtt{p}_2.\mathtt{end} \vdash \mu\mathtt{t}.!\mathtt{p}_1.!\mathtt{p}_2.\frownie}\,[\text{rec1}]$$

Another consideration when composing recursive and non-recursive protocols is fairness. The intuition of fairness is: *'In all executions allowed by an interleaving composition, each component protocol can proceed until it terminates'*. For the two protocols S_1 and S_2' given above, if we interleave them so that S_1 'comes first' we may obtain the following interleaving composition $\mu\mathtt{t}.!\mathtt{p}_1.\mathtt{t}$ that morally represents the protocol that behaves as S_2' after an infinite loop. Such a protocol clearly violates fairness. Similarly, an interleaving $\mu\mathtt{t}.!\mathtt{p}_1.!\mathtt{p}_2.\mathtt{end}$ would violate fairness by preventing S_1 from proceeding until it terminates (i.e., forever) again compromising fairness (Theorem 2). A composition that satisfies fairness is one in which the terminating protocol 'comes first': $!\mathtt{p}_2.\mu\mathtt{t}.!\mathtt{p}_1.\mathtt{t}$. Such a composition is obtained via [rec3]. Crucially, [rec3] only allows a recursive protocol to be introduced in an interleaving composition when the non-recursive component has already been all merged (i.e., it is $\mathtt{end}$). Thus, we can derive:

$$\cfrac{\cfrac{\cfrac{\emptyset\ \{\mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1\} \quad \mathsf{fv}(\mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1) = \emptyset}{\emptyset, \emptyset,\ \emptyset,\ \mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1\ \circ\ \mathtt{end} \vdash \mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1}\,[\text{rec3}]}{\emptyset, \emptyset,\ \emptyset,\ \mathtt{end}\ \circ\ \mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1 \vdash \mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1}\,[\text{sym}]}{\emptyset, \emptyset,\ \emptyset,\ !\mathtt{p}_2.\mathtt{end}\ \circ\ \mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1 \vdash !\mathtt{p}_2.\mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1}\,[\text{act}]$$

The premise $\mathsf{fv}(\mu\mathtt{t}_1.!\mathtt{p}_1.\mathtt{t}_1) = \emptyset$ prevents [rec3] being used inappropriately in case of nested recursion, e.g., to *prevent* the composition of $\mu\mathtt{t}_1.!\mathtt{p}_1.\mu\mathtt{t}_3.!\mathtt{p}_1.\mathtt{t}_1$ and $!\mathtt{p}_2.\mathtt{end}$ to produce (with some applications of [rec1], [act], [sym], and finally [rec3]) $\mu\mathtt{t}_1.!\mathtt{p}_1.!\mathtt{p}_2.\mu\mathtt{t}_3.!\mathtt{p}_1.\mathtt{t}_1$, which would violate behaviour preservation.

3.1 Variations on the branching rule

The branching rule of interleaving composition can be viewed as a *distributivity* property: sequential composition after a control-flow branch can be distributed inside the branches. Algebraically, we can informally describe this distributivity exhibited by the branching rule as follows, for a 2-way branch (sans labelling): $(S_1 + S_2) \circ T \equiv (S_1 \circ T) + (S_2 \circ T)$. Such a property is familiar in Kleene algebra models of programs and program reasoning [24] and monotone dataflow frameworks in static analysis [22]. Since interleaving composition generates a set of possible protocols it would be more accurate to express this property in terms of set membership rather than equality (for simplicity of the analogy, this elides the fact that each composition $\circ$ is itself a set):

$$(S_1 + S_2) \circ T \ \ni\ (S_1 \circ T) + (S_2 \circ T) \qquad \text{(distributivity)}$$

In this section we consider two variants of this distributive behaviour for composition called (1) 'weak branching' and (2) 'interchange branching' which can be summarised via the algebraic analogy as variants of distributivity, respectively:

$$(S_1 + S_2) \circ T \ \ni\ (S_1 \circ T) + S_2 \quad \wedge \quad (S_1 + S_2) \circ T \ \ni\ S_1 + (S_2 \circ T) \qquad \text{(weak)}$$

$$(S_1 + S_2) \circ (T_1 + T_2) \ \ni\ (S_1 \circ T_1) + (S_2 \circ T_2) \qquad \text{(interchange)}$$

In (weak), composition distributes inside one branch but not the other. In (interchange)[2], composing branches with branches has a 'merging' effect on the branches rather than distributing within.

We motivate and discuss each variation in term from the protocol perspective. In the rest of this section we introduce two additional composition rules: [wbra] for weak branching, and [cbra] for interchange branching (which we will refer to as *correlating branching* as it better reflects the effects of the rule on the protocols). Note that these two variations grow the set of possible interleavings, rather than shrinking it: they provide more general composition behaviours but do not exclude the more specialised behaviours. For generality of the theory, the derivation of interleaving composition can apply any branching ([bra], [wbra], [cbra]). For practicality, our tool allows engineers to choose the kind of branching to use in any specific scenario (as shown in Section 5).

[2] The naming of algebraic properties of this form is common in category theory [23].

Weak branching *Weak branching* allows for partial execution of some of the protocols being composed if there are not sufficient assertions to continue, as long as all protocols are completely executed in some execution path.

Example 3 (Branching with "asymmetric" guarantees). Protocol S_B below needs assertion n to proceed. Assume we want to compose S_B with a protocol S_A, which can provide n in only one of its branches ok. S_A may be an authentication server, granting or blocking access to S_B depending on a password `pwd`:

$$S_A ::= ?\texttt{pwd}.\oplus\{\mathsf{ok} : \texttt{assert}(n).\ \texttt{end},\ \mathsf{ko} :\ \texttt{end}\} \qquad S_B ::= \texttt{require}(n).S'$$

for some S'. Since we want the actions of S_B not to be executed after selection of label ko, we want interleaving composition to generate the following protocol:

$$S_{AB} = ?\texttt{pwd}.\oplus\{\mathsf{ok} :\ \texttt{assert}(n).\texttt{require}(n).S',\ \mathsf{ko} :\ \texttt{end}\}$$

Interleaving composition S_{AB} is not attainable using the rules of Definition 7: the derivation blocks when composing $\texttt{require}(n).S'$ with the second branch's `end` in the empty environment. [3]

Example 3 illustrates that asymmetric composition is a reasonable characterization of some scenarios. Definition 8 introduces a 'weak branching' composition rule [wbra] to allow for asymmetric guarantees.

Definition 8 (Weak branching). Weak branching composition *of protocols is derived using the judgements in Definition 7 and the* additional *rule [wbra]:*

$$\frac{\begin{array}{l} I = I_A \cup I_B \quad I_A \cap I_B = \emptyset \quad I_A \neq \emptyset \\ \forall i \in I_A.\quad T_L,\, T_R,\, A,\ S_i \circ S \vdash S'_i \\ \forall i \in I_B.\quad T_L,\, T_R,\, A,\ S_i \circ S \nvdash \ \wedge \ \ A\ \{S_i\} \end{array}}{T_L,\, T_R,\, A,\ +\{\mathrm{l_i} : S_i\}_{i\in I} \circ S \vdash +\{\mathrm{l_i} : S'_i\}_{i\in I_A} \cup \{\mathrm{l_i} : S_i\}_{i\in I_B}}\ [\text{wbra}]$$

Precondition $I_A \neq \emptyset$ ensures that each protocol's actions are executed in at least one execution path, and is key to the fairness property introduced in *Section* 4.1. Hereafter we denote with $\vdash_s$ derivations obtained using the judgements in Definition 7 only, and $\vdash_w$ for derivations with the additional rule [wbra].

Example 4 (Weak interleaving composition of banking and PIN/TAN). Consider the banking and PIN/TAN protocols in Example 1. Interleaving composition of S_A and S_B using $\vdash_s$ returns an empty set. When using $\vdash_w$, instead, we can derive the following interleaving composition modelling a banking/authentication protocol that satisfies the requirements specified in Section 1.

$$S_{BA} = ?\texttt{pin}.\oplus\left\{\begin{array}{l} \mathsf{ok} :\ \texttt{assert}(pin).\texttt{require}(pin).\mu\texttt{r}.\&\left\{\begin{array}{l} \mathsf{payment} :\ S_{TAN}, \\ \mathsf{statement} : !\texttt{statement}.\texttt{r}, \\ \mathsf{logout} :\ \texttt{consume}(pin).\ \texttt{end} \end{array}\right\} \\ \\ \mathsf{fail} :\ \texttt{end} \end{array}\right\}$$

[3] If we start from a non-empty environment $\{n\}$ we can derive $?\texttt{pwd}.\oplus\{\mathsf{ok} : \texttt{assert}(n).\texttt{require}(n).S', \mathsf{ko} : \texttt{require}(n).S'\}$. However, initial assumption $\{n\}$ means that access to S_B is granted regardless of the authentication outcome.

$$S_{TAN} = \texttt{assert}(pay).\texttt{consume}(pay).!\texttt{id}.?\texttt{tan}.\oplus\left\{\begin{array}{l}\mathsf{ok} : \texttt{assert}(tan).\,\texttt{consume}(tan).\\ \qquad ?\texttt{details}.\,\texttt{r},\\ \mathsf{fail} : \texttt{r}\end{array}\right\}$$

Correlating branching *Correlating branching* allows two protocols to be composed by 'correlating' each branch of one with at least one branch of the other.

Example 5 (Correlating branching). Consider two branching protocols: S_1 offering two services $\mathsf{s1}$ and $\mathsf{s2}$, and S_2 offering two kinds of payment $\mathsf{p1}$ and $\mathsf{p2}$. When composing S_1 and S_2, we want to correlate $\mathsf{s1}$ with $\mathsf{p1}$, and $\mathsf{s2}$ with $\mathsf{p2}$. We use assertions to model the desired correlation, as shown below:

$$\begin{array}{l} S_1 = \oplus\{\mathsf{s1} : \texttt{assert}(one).\texttt{end},\ \ \mathsf{s2} : \texttt{assert}(two).\texttt{end}\}\\ S_2 = \oplus\{\mathsf{p1} : \texttt{consume}(one).\texttt{end},\ \ \mathsf{p2} : \texttt{consume}(two).\texttt{end}\}\end{array}$$

We would like to obtain the following composition:

$$S_{12} = \oplus\left\{\begin{array}{l}\mathsf{s1} : \texttt{assert}(one).\oplus\{\mathsf{p1} : \texttt{consume}(one).\texttt{end}\},\\ \mathsf{s2} : \texttt{assert}(two).\oplus\{\mathsf{p2} : \texttt{consume}(two).\texttt{end}\}\end{array}\right\}$$

Composition rule [bra] is too strict and returns an empty set for S_1 and S_2. Weak branching [wbra] is also not useful in this case, producing the interleaving below, which does not capture the intended correlation:

$$\oplus\left\{\begin{array}{l}\mathsf{p1} : \oplus\left\{\mathsf{s1} : \texttt{assert}(one).\texttt{consume}(one).\texttt{end}, \mathsf{s2} : \texttt{assert}(two).\texttt{end}\right\},\\ \mathsf{p2} : \oplus\left\{\mathsf{s1} : \texttt{assert}(one).\texttt{end}, \mathsf{s2} : \texttt{assert}(two).\texttt{consume}(two).\texttt{end}\right\}\end{array}\right\}$$

Definition 9 introduces a further rule [cbra], to allow for correlating compositions.

Definition 9 (Correlating branching). *The* correlating branching composition *of two protocols is derived using the judgement in Definition 7 with the addition of rule [cbra] below:*

$$\frac{\begin{array}{lll}\forall i \in I \quad J_i \neq \emptyset\ \wedge \bigcup_{i\in I} J_i = J, & \forall j \in J_i & T_L,\ T_R,\ A,\ S_i \circ S'_j \vdash S_{ij},\\ & \forall j \in J \setminus J_i & T_L,\ T_R,\ A,\ S_i \circ S'_j \nvdash\end{array}}{T_L,\ T_R,\ A,\ +\{\mathsf{l_i} : S_i\}_{i\in I} \circ +'\{\mathsf{l'_j} : S'_j\}_{j\in J} \vdash +\{\mathsf{l_i} : +'\{\mathsf{l'_j} : S_{ij}\}_{j\in J_i}\}_{i\in I}}\ [\text{cbra}]$$

The first premise requires that: (1) each branch of the first protocol can be correlated with at least one branch of the second protocol ($J_i \neq \emptyset$), and (2) each branch of the second protocol can be correlated with at least one branch of the first protocol ($\bigcup_{i\in I} J_i = J$). This precondition is critical to ensure the fairness property we introduce in Section 4.1. Rule [cbra] allows us to obtain S_{12} as the interleaving composition of S_1 and S_2 above, modelling the intended correlation.

Hereafter we denote with $\vdash_c$ (resp. $\vdash_{wc}$) derivations obtained using the judgments in Definition 9 with the addition of rule [cbra] (resp. [cbra] and [wbra]). The inclusion relation between the different kinds of judgment is shown of the right (with $\vdash_s$ and $\vdash_{wc}$ being the most and less strict, respectively).

$$\begin{tikzcd} & \vdash_w \arrow[dr, phantom, "\subset"] & \\ \vdash_s \arrow[ur, phantom, "\subset"] \arrow[dr, phantom, "\subset"] & & \vdash_{wc} \\ & \vdash_c \arrow[ur, phantom, "\subset"] & \end{tikzcd}$$

4 Properties of interleaving composition

In this section, we give the main properties of interleaving compositions. First, we give some general properties of well-assertedness and algebraic/scoping properties (i.e., sanity checks). Then, in Section 4.1 we give behaviour preservation and fairness, both formulated using a semantics of 'protocol ensembles' (a semantic counterpart of syntactic composition). Hereafter, we will denote with $\vdash$ any kind of judgment in $\{\vdash_s, \vdash_w, \vdash_c, \vdash_{wc}\}$.

Well-assertedness of compositions Critical for the validity of our approach is that interleaving compositions preserve the constraints of assertions:

Proposition 3 (Validity) *If $T_L,\ T_R,\ \emptyset,\ S_1 \circ S_2 \vdash S$ then S is very-well-asserted.*

Appendix D details the proof. A corollary of Proposition 3 and Lemma 3 (progress of very-well-asserted protocols) is that interleaving compositions enjoy progress:

Corollary 1 (Progress) *If $T_L,\ T_R,\ \emptyset,\ S_1 \circ S_2 \vdash S$ then S enjoys progress.*

Algebraic and scoping properties Protocols may contain recursion variables, thus we consider notions of open and closed protocol with respect to recursion variables. Observe that, when composing two closed recursive protocols we only obtain closed protocols. This property is a corollary of a more general property, that free variables are preserved by interleaving composition:

Proposition 4 *If $T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash S$ then $\mathsf{fv}(S_1) \cup \mathsf{fv}(S_2) = \mathsf{fv}(S)$.*

That is, the free variable set of a composed protocol is exactly the union of the free variables of the protocols being composed. Hence, composing protocols with no free variables necessarily obtains a closed protocol:

Corollary 2 (Composition preserves closedness) *For all A, S and closed protocols S_1, S_2, if $T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash S$ then S is a closed protocol.*

A useful algebraic property is that composition has `end` protocols as units:

Proposition 5 (Interleaving composition has left- and right-units)

$$A\ \{S\} \wedge \mathsf{fv}(S) = \emptyset \implies T_L,\ T_R,\ A,\ S \circ \mathtt{end} \vdash S \ \wedge\ T_L,\ T_R,\ A,\ \mathtt{end} \circ S \vdash S$$

Appendix E details the proofs of the above results.

4.1 Behaviour preservation and fairness of protocol ensembles

To formalise a notion of behaviour preservation for interleaving composition, we first define an intermediate notion of *protocol ensembles.*

Protocol ensembles In Section 3, we gave a syntactic definition of *interleaving composition*. Interleaving composition makes the dependencies between protocols explicit, and provides a blue-print of an implementation. In this section, we consider 'protocol ensembles', which can be understood as the *semantic compositions* of two asserted protocols. Semantic compositions have a behaviour that is similar to parallel composition (e.g., as in CCS), but unlike parallel composition the two asserted protocols cannot communicate with each other, i.e., there are no internal τ actions. All interactions in a semantic composition are directed towards other endpoints (i.e., communication co-parties). Semantic composition provides a more general and somewhat familiar notion of composition, which we will use as a reference to analyze the properties of interleaving compositions.

Protocol ensembles, ranged over by C, are defined as follows:

$$\begin{array}{rll} C ::= & S & \text{(asserted protocol)} \\ \mid & S \,||\, S & \text{(semantic composition)} \end{array}$$

By defining protocol ensembles C as either asserted protocols (which may be compositions) or semantic compositions, we obtain a common LTS for comparing the behaviour of interleaving compositions and semantic compositions. For simplicity of presentation, we limit the theory to the composition $S_1 \,||\, S_2$ of two protocols S_1 and S_2. The extension to n parallel protocols is straightforward although possibly verbose e.g., based on labelling each protocol, as well as its actions, with a unique identifier.

The LTS for protocol ensembles extends the LTS for asserted protocols: it is defined over states of the form (A, C), transition labels L (as for asserted protocols), and by the rules in Definition 3 plus the following two rules:

$$\frac{(A, S_1) \xrightarrow{\ell} (A', S_1')}{(A, S_1 \,||\, S_2) \xrightarrow{\ell} (A', S_1' \,||\, S_2)} \langle\texttt{Com1}\rangle \qquad \frac{(A, S_2) \xrightarrow{\ell} (A', S_2')}{(A, S_1 \,||\, S_2) \xrightarrow{\ell} (A', S_1 \,||\, S_2')} \langle\texttt{Com2}\rangle$$

We write $(A, C) \rightarrow$ if $(A, C) \xrightarrow{\ell} (A', C')$ for some ℓ, A', C'. Protocols in C do not communicate internally, but they may affect each other by adding, checking, or removing assertions in A.

Behaviour preservation Fix an LTS for protocol ensembles $(Q, L, \rightarrow)$ defined on the set Q of states $\mathbf{s}$ of the form (A, C). We use the standard notion of *simulation* [30] to compare protocols of interleaving compositions and protocol ensembles, using protocol ensembles as a correct general model to which interleaving compositions need to adhere.

Definition 10 (Simulation). *A (strong)* simulation *is a relation* $\mathcal{R} \subseteq Q \times Q$ *such that, whenever* $\mathbf{s}_1 \mathcal{R} \mathbf{s}_2$*:* $\forall \ell \in L, \mathbf{s}_1' : \mathbf{s}_1 \xrightarrow{\ell} \mathbf{s}_1'$ *implies* $\exists \mathbf{s}_2' : \mathbf{s}_2 \xrightarrow{\ell} \mathbf{s}_2'$ *and* $\mathbf{s}_1' \,\mathcal{R}\, \mathbf{s}_2'$.

We call 'similarity' the largest simulation relation. We write $\mathbf{s}_1 \lesssim \mathbf{s}_2$ when there exists $\mathcal{R}$ such that $\mathbf{s}_1 \mathcal{R} \mathbf{s}_2$.

Definition 11 (Behaviour preservation). *We say that C_1 preserves the behaviour of C_2 with respect to A if $(A, C_1) \lesssim (A, C_2)$.*

Theorem 1 (Behaviour preservation of compositions - closed).

$$\emptyset, \emptyset, A, S_1 \circ S_2 \vdash S \quad \Rightarrow \quad (A, S) \lesssim (A, S_1 \,\|\, S_2)$$

Therefore, interleaving compositions will only show behaviour that would be allowed by a protocol ensemble. Clearly, protocol ensembles allow more possible executions than an interleaving composition, which is only one of the possible interleavings. The proof of Theorem 1 is by induction on the derivation of S and, although the statement assumes closed protocols, some inductive hypotheses in the proof (e.g., premises of [rec1] or [rec2]) require reasoning about open protocols. The proof hence relies on a property (Lemma F6 – appendix) on open protocols: (roughly) given two protocols and one of their interleaving compositions, any action of the interleaving composition is matched by an action of the ensemble of the two protocols, and this property is preserved upon transition. Note that, while environments T_L and T_R are trivially empty in Theorem 1 (closed protocols), they have a key role in proving Lemma F6 (open protocols): they include the variables of each component protocol that are "morally" bound in a derivation, and give critical information of the scope and structure of the original component protocols in that derivation. The proof focusses on proving $\vdash_{wc}$ which is the most general case; the other cases can be obtained by omitting the cases for rules not used by that kind of composition (e.g., for $\vdash_s$ omit the [wbra] and [cbra] case). Appendix F details the full proof.

Fairness The fairness enjoyed by interleaving compositions depends on the set of composition rules used to derive that composition.

Given the ensemble of two protocols $S_0 \,\|\, S_1$, and any of their interleaving compositions S, each action of $S_0 \circ S_1$ (and hence of any one of the two protocols S_0 and S_1 being composed) can be observed in at least one execution of the interleaving composition S, possibly after a finite sequence of other actions by the other protocol being composed. For readability, in the following definition we will write $(\_, S)$ to denote (A, S) when A is not relevant to the definition.

Definition 12 (Fairness). *S is* fair *w.r.t. S_0 and S_1 on A, if $\forall i \in \{0, 1\}$ and any transition $(\_, S_i) \xrightarrow{\ell} (\_, S'_i)$ there exists $\boldsymbol{r}$ such that:* ***1)*** *$(A, S_{|1-i|}) \xrightarrow{\boldsymbol{r}} (\_, S'_{|1-i|})$,* ***2)*** *$(A, S) \xrightarrow{\boldsymbol{r}\ell} (A', S')$, and* ***3)*** *$S'$ is fair with respect to S'_i and $S'_{|1-i|}$ on A'.*

Theorem 2 (Fairness of compositions with $\vdash$). *If $\emptyset, \emptyset, A, S_0 \circ S_1 \vdash S$ then S is* fair *w.r.t. S_0 and S_1 on A.*

One key characteristic of fairness in Definition 12 is that first we fix ℓ, and then we require at least one execution in which ℓ is eventually executed by S. This implies that although not all possible future branches include all parts of the protocols being composed, some will. This is illustrated for $\vdash_c$ in Example 6, and for $\vdash_w$ later in the section in Example 7.

Example 6 (Fairness and correlating branching). We illustrate fairness on S_1, S_2, and their interleaving composition S_{12} with correlating branching given in Example 5. If S_1 makes a transition, then it is trivial to show that S_{12} is a fair composition by taking $\boldsymbol{r}$ as the empty vector. The interesting case is the one where S_2 makes a step, picking one of the two possible labels. Assume S_2 transitions with $\oplus\,\mathrm{p1}$. Given $\ell = \oplus\,\mathrm{p1}$, there exists $\boldsymbol{r} = \oplus\,\mathrm{s1}, \mathtt{assert}(1)$ such that $(\emptyset, S_1) \xrightarrow{\boldsymbol{r}}$ and $(\emptyset, S_{12}) \xrightarrow{\boldsymbol{r}} (\{1\}, \oplus\{\mathrm{p1} : \mathtt{consume}(1).\mathtt{end}\}) \xrightarrow{\oplus\,\mathrm{p1}}$ as required. The case for $\oplus\,\mathrm{p2}$ is similar.

In Theorem 3 we give a stronger fairness result for compositions using only [bra] (i.e., holding only for $\vdash_s$ judgments). Each action of the protocol ensemble $S_0 \circ S_1$ (and hence of one of the protocols in the ensemble) can be observed in any execution of the interleaving composition, possibly after a finite sequence of other actions by the other protocol in the ensemble composed.

Definition 13 (Strong fairness). *S is* strongly fair *w.r.t. S_0 and S_1 on A, if any $i \in \{0, 1\}$ and all transitions $(\_, S_i) \xrightarrow{\ell} (\_, S_i')$ and $(A, S_{|1-i|}) \xrightarrow{\boldsymbol{r}}$, there exist $\boldsymbol{r'}$, $\boldsymbol{r''}$ with $(A, S_{|1-i|}) \xrightarrow{\boldsymbol{r'}} (\_, S'_{|1-i|})$ and either:*
1) $\boldsymbol{r'r''} = \boldsymbol{r}$ (i.e., $\boldsymbol{r'}$ is a prefix of $\boldsymbol{r}$), or
2) $\boldsymbol{r'} = \boldsymbol{rr''}$ (i.e., $\boldsymbol{r}$ is an ex prefix of $\boldsymbol{r'}$)

such that $(A, S) \xrightarrow{\boldsymbol{r'}\ell} (A', S')$ and S' is strongly fair w.r.t. S_i' and $S'_{|1-i|}$ on A'.

By Definition 13 any action of a composition can be matched by an action of the protocols being composed, and this property is preserved by transition. Vectors $\boldsymbol{r}$, $\boldsymbol{r'}$, and $\boldsymbol{r''}$ are used to universally quantify on $\boldsymbol{r}$ and yet allow for the cases where ℓ comes before (1) or after (2) $\boldsymbol{r}$ in the composition.

Theorem 3 (Strong fairness of compositions with $\vdash_s$). *Assume*

$$\emptyset,\ \emptyset,\ A,\ S_0 \circ S_1 \vdash_s S$$

then S is strongly fair *with respect to S_0 and S_1 on A.*

Appendix G details the proofs.

Example 7 (Fairness and weak branching). Consider a simpler variant of the protocols in Example 3 (omitting password exchange and continuation):

$$\begin{aligned} S_A &= \oplus\{\mathrm{ok} : \mathtt{assert}(n).\mathtt{end},\ \mathrm{ko} : \mathtt{end}\} \qquad S_B = \mathtt{require}(n).\ \mathtt{end} \\ S_{AB} &= \oplus\{\mathrm{ok} : \mathtt{assert}(n).\mathtt{require}(n).\mathtt{end},\ \mathrm{ko} : \mathtt{end}\} \end{aligned}$$

Observe $\emptyset, \emptyset, \emptyset$, $S_A \circ S_B \not\vdash_s$ and $\emptyset, \emptyset, \emptyset$, $S_A \circ S_B \vdash_w S_{AB}$. We show that S_{AB} is a fair composition w.r.t. S_A and S_B on $\emptyset$, but it is not a *strongly* fair one.

First focus on fairness. S_A can move with either label $\oplus\,\mathrm{ok}$ or $\oplus\,\mathrm{ko}$. In either cases $(\emptyset, S_{AB})$ can immediately make a corresponding step with $\boldsymbol{r}$ empty. If S_B moves, that is by label $\mathtt{require}(n)$, then for some environment $\{n\}$:

$$(\{n\}, S_B) \xrightarrow{\mathtt{require}(n)} (\emptyset, \mathtt{end}) \tag{1}$$

There exists a sequence of transitions with labels $\boldsymbol{r} = \oplus\,\mathrm{ok}, \mathtt{assert}(n)$ such that

$$(\emptyset, S_B) \xrightarrow{\oplus\,\mathrm{ok},\mathtt{assert}(n)} (\{n\}, \mathtt{end})$$

$$(\emptyset, S) \xrightarrow{\oplus\,\mathrm{ok},\mathtt{assert}(n)} (\{n\}, \mathtt{require}(n).\mathtt{end}) \xrightarrow{\mathtt{require}(n)} (\emptyset, \mathtt{end})$$

and $\emptyset, \emptyset, \emptyset, \mathtt{end} \circ \mathtt{end} \vdash_w \mathtt{end}$. In the case above, we could select a 'good' path of S_A and S_{AB} that allows the transition with label $\mathtt{require}(n)$ to happen.

Focus now on strong fairness and again, consider the step in Equation (1) by S_B. Now we can pick an arbitrary $\boldsymbol{r}$, say, $\oplus\,\mathrm{ok}$, such that $(\emptyset, S_B) \xrightarrow{\oplus\,\mathrm{ko}} (\emptyset, \mathtt{end})$. Looking at S_{AB}, there is no prefix nor extension of $\boldsymbol{r} = \oplus\,\mathrm{ok}$ that allows a $\mathtt{require}(n)$ step by S_{AB} once the branch ko is taken. Therefore, S_{AB} is not strongly fair with respect to S_A and S_B on $\emptyset$.

5 Implementation

We have implemented our approach as a tool for Erlang that offers functionality for *protocol composition*, *code generation*, and *protocol extraction*.

Interleaving composition. Interleaving composition is defined as a function producing 0 or more possible protocol compositions, giving an algorithmic implementation of the relation in Definition 7. Following the variations on the branching rule, the tool offers strong, weak, correlating, and weak/correlating compositions. The user can select the kind of branching to use. Looking at the running Example 1, we can give as input the banking and authentication protocols, and opt for strong composition, which returns an empty set as expected. When opting for weak composition instead, the tool outputs two possible interleaving compositions, from which the user can choose the desired protocol. The banking and authentication protocols are composed into a single protocol where the actions of the two protocols follow a specific interleaving dictated by the assertions. The resulting interleaving composition, equivalent to example 4, is:

```
bank_pt() -> {act,r_pin,{branch,
  [{ok,{assert,pin,{require,pin,{rec,r,{branch,
     [{payment,{assert,pay,{consume,pay,{act,s_id,{act,r_tan,
        {branch,[{ok,{assert,tan,{consume,tan,
                      {act,r_details,{rvar,r}}}}}},
                  {fail,{rvar,r}}]}}}}}}},
      {statement,{act,s_statement,{rvar,r}}},
      {logout,{consume,pin,endP}}]}}}}},
  {fail,endP}]}}
```

Listing 1.1. PIN/TAN Banking Protocol

Offering all of the four different options of composition options (and not only the less restrictive weak/correlating branching) improves relevance of the compositions returned, and hence facilitates analysing and choosing between them. For example, as observed in Example 5, using [wbra] in a context where we need to correlate branching likely returns irrelevant compositions.[4] Another way to

[4] Annotations specifying which branching rule to use at specific points in the protocol would further increase relevance of the returned results. This is a further work.

Protocols	Strong	Weak	Correlating	Weak/Correlating
`service()`, `login()`	0	1	0	1
`services()`, `payments()`	3	3	3	3
`payment()`, `dispatch()`	1	1	1	1
`http()`, `aws_auth()`	1	1	1	1
`login()`, `booking()`	0	1	0	1
`pin()`, `tan()`	0	1	0	1
`pintan()`, `bank()`	0	2	0	2
`resource()`, `server()`	1	1	1	2
`userAgent()`, `agentInstrument()`	0	0	2	2
`bankauthsimple()`, `keycard()`	0	1	0	1

Table 1. Number of Compositions for Variations on the Branching Rule; running example highlighted in grey.

reduce the number of returned (irrelevant) compositions is to introduce more relevant assertions. In fact, one of the aims of the tool is to support step-wise understanding of the protocol via progressive insertion of assertions. For the main examples, presented in this paper, by selecting the most appropriate kind of composition and most appropriate assertions, the tool returns at most two interleaving compositions; this mostly happens when composing two recursive protocols as [sym] allows the recursion variable of each component to be used to denote the interleaved recursion. Table 1 shows the number of interleaving compositions obtained for each variation of the branching rule for a suite of examples used in this paper or from literature. For additional details and examples, see our repository with the complete benchmark (`https://anonymous.4open.science/r/protocol-reengineering-implementation-2DC5/`). For most examples, with appropriate assertions, the tool returns a small number of compositions. Thus, it is not hard for the user to assess and choose the most suitable for their domain. However, if for instance, several branches of a protocol offer the same assertion, when composed with another with that particular requirement, the number of compositions will be higher.

Code generation. Code generation takes a protocol definition and produces an Erlang stub. Protocol structures (action, sequence, choice) can be represented as a directed graph and then as finite state machines that transition based on the messages received. The finite state machines are used to generate a stub that uses the Erlang/OTP `gen_statem` [1], a generic abstraction which supports the implementation of finite state machine modules. Not only is it convenient to represent the protocol as a state machine, but `gen_statem` offers some useful features. Internal events from the state machine to itself are a good way of representing branches that make a selection among some choices. Co-located callback code for each state enables the use of non-atomic states, e.g., complex states or even hierarchical states. ‘Postponing events’ and timeouts provide functionality for further implementation of the generate code stubs.

Action and *branch* are represented as events that trigger a state transition. We use function declarations to represent incoming events, and function applications to represent outgoing events. Each state has its own handler function used to send an event to the state machine. When the event is received the corresponding state function is called and the transition to the next state is made. The default generated event is an asynchronous communication (called a ‘cast’ in Erlang/OTP parlance). For sending actions and selecting branches, the event type is internal, an event from your state machine to itself. *End* is represented by the terminate function of a `gen_statem` module, whilst the *fixed-point* and the *recursive variables* dictate the control flow of the state machine. State variables must be declared by including them in a record definition — `Data`.

Following the example of [13], we represent *assertions* as specially formatted comments. For example: `{assert, pay}` is represented as an Erlang comment `%assert pay`. These comments are positioned before code that implements the state to which this assertion acts as a pre-condition in the protocol.

Below is an excerpt of the code generated for the PIN/TAN Banking protocol, `bank_pt()`, showing the states generated for the first action and branch:

```
state2(cast, fail, Data) -> {stop, normal, Data}.
%assert pay
%consume pay
state3(cast, payment, Data) -> {next_state, state4, Data};
state3(cast, statement,Data) ->{next_state, state10,Data};
%consume pin
state3(cast, logout, Data) -> {stop, normal, Data}.
```

Listing 1.2. PIN/TAN Banking State Machine

Protocol extraction and migration. Protocol extraction generates protocols from code via a static analysis of Erlang modules implemented as state machines using either `gen_statem`, or `gen_fsm` behaviour. When assertions are expressed using the comments illustrated above, they are also extracted. The obtained protocol can be annotated with extra assertions as necessary and composed with another to obtain a more complex protocol. The extraction option preserves preexisting local code that can be automatically migrated when generating a new stub. For example, starting out from an existing implementations of banking, we can use the tool to extract the protocol S_B (in this case manual introduction of assertions may be needed), obtain an interleaving composition with S_A, and generate a new implementation where pre-existing code for the banking code can be migrated.

If we wanted to re-engineer the banking/authentication server to include an option for keycard authentication (in addition to TAN authentication) we could further compose the PIN/TAN Banking Protocol with a keycard option protocol as the one below. Assertions ensure that the branching for choosing TAN or keycard authentication is plugged in (using assertion `keyp`) to the payment option of the PIN/TAN protocol, and that TAN authentication in PIN/TAN protocol is plugged only in the `tan` branch of the keycard protocol (using assertion `otp`):

```
keycard() -> {rec, y, {require, keyp,
                   {branch, [{tan, {assert, otp, {rvar, y}}},
                             {keycard, {rvar, y}}]}}}.
```

Listing 1.3. Keycard Option Protocol

By adding an assertion of `keyp` and a consume `otp` at the beginning of the branch `payment` of the PIN/TAN Protocol one would obtain the desired extension as interleaving composition, using the weak composition option. The tool can be used to generate a stub for the extended protocol and migrate reusable code from the implementation of the PIN/TAN Banking Protocol to the new implementation.

Together these features satisfy the requirements laid out in Section 1, facilitating program re-engineering. We can obtain skeleton implementations for banking and PIN/TAN protocol, extract the protocol and reuse the code when composing with a different protocol.

6 Instantiation to known protocol languages

Our protocol language (Section 2) is parametric on the set of action/branching prefixes and branching labels and our results hold regardless of the specific instantiation used. Instantiation allows us to apply our framework to different scenarios. Many of the examples in this paper are in the context of concurrency or distribution, yet protocols are pervasive in monolithic sequential code, e.g., for interacting with an operating system or libraries. A classic example is the stateful protocol of file handling in which files must be opened and closed, and only read and written to according to their permissions between those two actions. Our protocol language can easily model such situations, with interleaving composition providing a range of choices to a developer about how to combine and interact multiple stateful protocols.

In this section we provide a more concrete discussion of a few use cases focussing on interaction and communication, providing hints at possible synergies between our framework and techniques already studied for specific formalisms.

6.1 Protocols for communicating processes

Interaction structures of a communicating process can be characterised using process calculi such as CCS [27]. For example, a CCS process $S = \mathit{com}.S + \overline{\mathit{out}}.0$ can be understood as a protocol prescribing the interactions supported by a server: repeatedly receive commands (*com* is a receive action) or decide to logout (*out* is a send action) and terminate. The CCS notation can be expressed in our protocol language by instantiating

$$\mathcal{P} = \{a, \overline{a} \mid a \in \mathit{Names}\} \quad + \in \{+\} \quad \mathcal{L} = \mathcal{P}$$

where a (resp. $\overline{a}$) models a receive (resp. send) action on channel a, and *Names* is a set of channel names. Hence, S above can be represented in our framework as the process below $\mu\mathtt{t}.+\{\mathit{com} : \mathtt{t},\ \overline{\mathit{out}} : \mathtt{end}\}$. In many examples in this paper we used the following instantiation:

$$\mathcal{P} = \{!\mathtt{T}, ?\mathtt{T} \mid T \in \mathcal{T}\} \quad + \in \{\oplus, \&\} \quad \mathcal{L} \subset \mathbb{N}$$

where $\mathcal{T}$ is a set of datatypes for messages. The above instantiation yields a session types-like syntax in the style of [14]. Session types [18,33,19] represent types of sessions through communication channels, describing the type of data that can be sent or received on a channel (e.g., `!T` and `?T` where `T` is a datatype) and the patterns of message exchanges (e.g., sequential composition, recursion). This syntax is more restrictive than CCS (e.g., it is not possible to model mixed choices as in the CCS process above, where S is in a state in which it can either receive or send a message). The syntactic restrictions of session types contribute to their effectiveness for verification.

6.2 Protocols as session types

As mentioned in Remark 1 we often used a session types-like syntax in our examples. In this section we show how to use our framework in combination with session types techniques.

Recall, in Example 4, we have used interleaving composition to generate a banking/authentication protocol S_{BA} using a session types-like syntax, and in Section 5 we have shown that our tool can generate a stub implementation of S_{BA}, which one can then extend with local (i.e., non communication-related) behaviour. Assume this implementation, say `bank_pt.erl`, is published as a web API. S_{BA} can be published as a *behavioural* API.

So far, our framework has provided assurances on the relationship between component protocols and their interleaving composition (which pertains to the engineering within one node in a distributed system – in this case the banking/authentication server). Session types serve an orthogonal purpose: to provide assurances about the *inter-relations* of the protocols implemented in different nodes, e.g., given a well-defined banking/authentication server, how to derive a *suitable* client?

Anyone willing to develop a client for the banking/authentication service can use S_{BA} to algorithmically derive a client protocol by using the notion of *duality* of Session Types [18,33,19]. The dual of a protocol is obtained alghorytmically, by swapping the 'direction' of each action and branching: ! with ?, $\oplus$ with &, and vice-versa. We let $\overline{S_{BA}}$ denote the dual of S_{BA}. Our tool can be used again to generate a stub for $\overline{S_{BA}}$ (e.g., file `co_bank_pt.erl`). Session types duality and communication structuring (e.g., determinism, no mixed choices) yields a *safe* distributed system, resulting from, say, the concurrent and possibly distributed execution of `bank_pt.erl` and `co_bank_pt.erl`. In this context, *safe* means no deadlock and no communication mismatches even when communications are asynchronous [12] as in Erlang.

The protocol language can be instantiated as multiparty session types (e.g., local types in [9]) by setting $\mathcal{P} = \{ab\#\mathtt{T} \mid a, b \in \mathit{Roles},\ \# \in \{!, ?\},\ T \in \mathcal{T}\}$ with $+ \in \{ab\# \mid a, b \in \mathit{Roles},\ \# \in \{\oplus, \&\}\}$ and $\mathcal{L} \subset \mathbb{N}$. Safety can then be ensured using existing multiparty compatibility [15] and verification techniques [26]. See Appendix A.2 for an example of interleaving composition based on the multiparty session types instantiation.

7 Related Work and Conclusion

There is a vast literature on protocol specification (both theory and practice, e.g. [25]). Most techniques provide a monolithic, closed view of a system: a protocol is given for the entire system and all components are assumed to be present. We instead studied composition in open, general terms, defining composition as the interleaving of protocols, using 'assertions' to specify contact points and constraints between the protocols. We have given correctness in terms of behaviour preservation, fairness and well-assertedness, and shown that all compositions enjoy it. There are three main lines of research that relate to our work.

Firstly, *software adaptors* give typed protocol interfaces between software components [34]. The idea is similar to the structured view of communication in session types [19], with the notion of *duality* capturing when opposite sides of a protocol are compatible. Composition in these works is really about *parallel composition* along a protocol interface, guaranteeing sound communication. Instead, we study a sequential interleaving composition of protocols from the perspective of a single party (the common situation of engineering a single component of a larger system). Our assertions provide a kind of compatibility, but at the level of the protocol's (possibly interacting) application logics.

Secondly, composition has been studied as the run-time 'weaving' of component actions. Barbanera et al. study such a composition in the setting of asynchronous FIFO communicating finite state machines, while guaranteeing lock freedom [2,3]. Participants in two communicating systems can be transformed into coupled 'gateways', forming a composite system. Messages sent to one of the gateways are forwarded to the other, which in turn sends it to the other system. A compatibility relation is based on dual behaviour of the two gateways. Safety of the resulting system is by this compatibility, along with conditions of 'no mixed states' and determinism for sends and receives. Building from this idea, later work looks at composition in a setting of synchronous CFSMs, and replaces the two coupled gateways with a single one [5]. [4] look at direct composition and decomposition on global types in the setting of multiparty session types. Inspired by aspect-oriented programming, [32] support protocol extensions with 'aspectual' session types, that allow messages in session types to be matched and consequently introduce new behaviour in addition to, or in place of, the matched messages. [29] look at composition in the setting of choreographies. Composition relies on the use of partial choreographies, which can mix global descriptions with communication among external peers. Unlike the above approaches, we focus on a syntactic, statically derivable notion of composition. Concretely, we use process calculi to model protocols as simple syntactic objects that can be used to reason about the desired application logic and generate/engineer modular code. Again, we differ in that our protocols are understood as being enacted by a single process (within a larger system).

The third pertinent thread in the literature defines syntactic compositions in the form of Team Automata [16,7,6] or related calculi [7]. These works define different ways of composing machines, primarily based on synchronising machines via common actions. They consider a general framework that is parametric in choices about synchronisation in an n-party context. In contrast, our means of composition is via assertions (orthogonal to actions) which express directional (i.e., rely-guarantee-style) dependencies. Our use of assertions aims to reflect programming practice. Assertions are also kept in our generated code and can be used to enable protocol extraction and re-engineering (as well as understanding and code documentation). Thus we focus on protocol compositions that are sensitive to the application logic of the protocols being composed. Our composition cannot capture the whole range of synchronizations offered by Team Automata. Conversely, Team Automata cannot capture the range of compositions possible in our approach. One can encode some interleaving compositions as Team Automata, by modelling each $\texttt{assert}(n)$-$\texttt{require}(n)$ or $\texttt{assert}(n)$-$\texttt{consume}(n)$ pair as a common synchronization action. However, the options offered by Team Automata (e.g., 'free', 'state indispensable', or 'action indispensable') do not capture our requirement that synchronization (i) always happens on assertion-actions and (ii) never happens on communication actions (these are a separate syntactic entity). Furthermore, our assertions do not imply immediate synchronisation: an $\texttt{assert}(n)$ can occur in a protocol some way before a $\texttt{require}(n)$. Thus an attempted encoding of Team Automata into our protocols, encoding synchronization actions as unique $\texttt{assert}(n)$-$\texttt{consume}(n)$ pairs, would not preserve the behaviour of Team Automata for all possible compositions (just the ones where such 'annihilating' pairs appeared contiguously). Thus, Team Automata and our approach overlap in some synchronising behaviours, but not all. A formal study of the class of overlapping compositions between our approach and Team Automata is further work.

Unlike applications of team automata for safe communication [6], and other works discussed above, we do not focus on safe communications as such, which is an orthogonal concern for us. Our focus is on correct representation of the application logic via a notion of composition steered by assertions. However, the specific session-type-like notation we use, following [14], would allow us to inherit communication-safety properties from session types, even with asynchronous communications [12] and multiparty sessions [9], as discussed in Section 6.2. We can model higher order messages by instantiating prefixes to incorporate the entire protocol language itself, preventing use of delegated channels using assertions.

Future work For the theory, we are currently working on a factorisation function that decomposes protocols, as a kind of algebraic inverse to composition. This would allow us to 'close the loop', factorizing (possibly extracted) protocols into simple components for later (re)composition. Interestingly, our weak composition still retains sufficient information (existential quantification on the branches in Definition 8) about the component protocols for them to be correctly factorized. We also plan to extend recursion to model quantified recursion and assertion environments as multisets (e.g., to quantify on rely and guarantees).

References

1. AB, E.: Stdlib, reference manual. https://erlang.org/doc/man/gen_statem.html (2021), version 3.15.1
2. Barbanera, F., de'Liguoro, U., Hennicker, R.: Global types for open systems. In: Bartoletti, M., Knight, S. (eds.) Proceedings 11th Interaction and Concurrency Experience, ICE 2018, Madrid, Spain, June 20-21, 2018. EPTCS, vol. 279, pp. 4–20 (2018). https://doi.org/10.4204/EPTCS.279.4, https://doi.org/10.4204/EPTCS.279.4
3. Barbanera, F., Dezani-Ciancaglini, M.: Open multiparty sessions. In: Bartoletti, M., Henrio, L., Mavridou, A., Scalas, A. (eds.) Proceedings 12th Interaction and Concurrency Experience, ICE 2019, Copenhagen, Denmark, 20-21 June 2019. EPTCS, vol. 304, pp. 77–96 (2019). https://doi.org/10.4204/EPTCS.304.6, https://doi.org/10.4204/EPTCS.304.6
4. Barbanera, F., Dezani-Ciancaglini, M., Lanese, I., Tuosto, E.: Composition and decomposition of multiparty sessions. Journal of Logical and Algebraic Methods in Programming **119**, 100620 (2021). https://doi.org/https://doi.org/10.1016/j.jlamp.2020.100620, http://www.sciencedirect.com/science/article/pii/S235222082030105X
5. Barbanera, F., Lanese, I., Tuosto, E.: Composing communicating systems, synchronously. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12476, pp. 39–59. Springer (2020). https://doi.org/10.1007/978-3-030-61362-4_3, https://doi.org/10.1007/978-3-030-61362-4_3
6. ter Beek, M.H., Hennicker, R., Kleijn, J.: Team automata@work: On safe communication. In: Bliudze, S., Bocchi, L. (eds.) Coordination Models and Languages. pp. 77–85. Springer International Publishing, Cham (2020)
7. ter Beek, M.H., Kleijn, J.: Team automata satisfying compositionality. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003: Formal Methods. pp. 381–400. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
8. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5201, pp. 418–433. Springer (2008). https://doi.org/10.1007/978-3-540-85361-9_33, https://doi.org/10.1007/978-3-540-85361-9_33
9. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5201, pp. 418–433. Springer (2008). https://doi.org/10.1007/978-3-540-85361-9_33, https://doi.org/10.1007/978-3-540-85361-9_33
10. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: European Symposium on Programming. pp. 2–17. Springer (2007)

11. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. ACM SIGPLAN Notices **48**(1), 263–274 (2013)
12. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N.: Asynchronous session types and progress for object oriented languages. In: Bonsangue, M.M., Johnsen, E.B. (eds.) Formal Methods for Open Object-Based Distributed Systems, 9th IFIP WG 6.1 International Conference, FMOODS 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4468, pp. 1–31. Springer (2007). https://doi.org/10.1007/978-3-540-72952-5_1, `https://doi.org/10.1007/978-3-540-72952-5_1`
13. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: A Software Analysis Perspective. In: International conference on software engineering and formal methods. pp. 233–247. Springer (2012)
14. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. Inf. Comput. **256**, 253–286 (2017). https://doi.org/10.1016/j.ic.2017.06.002, `https://doi.org/10.1016/j.ic.2017.06.002`
15. Deniélou, P.M., Yoshida, N.: Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) Automata, Languages, and Programming. pp. 174–186. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
16. Ellis, C.: Team automata for groupware systems. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge. p. 415?424. GROUP '97, Association for Computing Machinery, New York, NY, USA (1997). https://doi.org/10.1145/266838.267363, `https://doi.org/10.1145/266838.267363`
17. Gay, S., Ravara, A.: Behavioural Types: From Theory to Tools. River Publishers (2017)
18. Honda, K.: Types for dyadic interaction. In: Best, E. (ed.) CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 715, pp. 509–523. Springer (1993). https://doi.org/10.1007/3-540-57208-2_35, `https://doi.org/10.1007/3-540-57208-2_35`
19. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1381, pp. 122–138. Springer (1998). https://doi.org/10.1007/BFb0053567, `https://doi.org/10.1007/BFb0053567`
20. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Necula, G.C., Wadler, P. (eds.) Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008. pp. 273–284. ACM (2008). https://doi.org/10.1145/1328438.1328472, `https://doi.org/10.1145/1328438.1328472`
21. Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1), 3:1–3:36 (2016). https://doi.org/10.1145/2873052, `https://doi.org/10.1145/2873052`
22. Kam, J.B., Ullman, J.D.: Monotone data flow analysis frameworks. Acta Informatica **7**(3), 305–317 (1977)

23. Kock, J.: Note on commutativity in double semigroups and two-fold monoidal categories. arXiv preprint math/0608452 (2006)
24. Kozen, D.: On kleene algebras and closed semirings. In: International Symposium on Mathematical Foundations of Computer Science. pp. 26–47. Springer (1990)
25. Lai, R.: A survey of communication protocol testing. Journal of Systems and Software **62**(1), 21–46 (2002)
26. Lange, J., Yoshida, N.: Verifying asynchronous interactions via communicating session automata. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 97–117. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_6, https://doi.org/10.1007/978-3-030-25540-4_6
27. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3, https://doi.org/10.1007/3-540-10235-3
28. Montesi, F.: Choreographic programming. IT-Universitetet i København (2014)
29. Montesi, F., Yoshida, N.: Compositional choreographies. In: D'Argenio, P.R., Melgratti, H.C. (eds.) CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8052, pp. 425–439. Springer (2013). https://doi.org/10.1007/978-3-642-40184-8_30, https://doi.org/10.1007/978-3-642-40184-8_30
30. Sangiorgi, D., Walker, D.: The Pi-Calculus - a theory of mobile processes. Cambridge University Press (2001)
31. Strom, R.E., Yemini, S.: Typestate: A programming language concept for enhancing software reliability. IEEE Transactions on Software Engineering (1), 157–171 (1986)
32. Tabareau, N., Südholt, M., Tanter, É.: Aspectual session types. In: Binder, W., Ernst, E., Peternier, A., Hirschfeld, R. (eds.) 13th International Conference on Modularity, MODULARITY '14, Lugano, Switzerland, April 22-26, 2014. pp. 193–204. ACM (2014). https://doi.org/10.1145/2577080.2577085, https://doi.org/10.1145/2577080.2577085
33. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D.G., Philokyprou, G., Theodoridis, S. (eds.) PARLE '94: Parallel Architectures and Languages Europe, 6th International PARLE Conference, Athens, Greece, July 4-8, 1994, Proceedings. Lecture Notes in Computer Science, vol. 817, pp. 398–413. Springer (1994). https://doi.org/10.1007/3-540-58184-7_118, https://doi.org/10.1007/3-540-58184-7_118
34. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Transactions on Programming Languages and Systems (TOPLAS) **19**(2), 292–333 (1997)

A Additional Examples

A.1 Interleaving Composition

Consider the protocols `!Int.end` and `!String.end`. By combining [act] and [sym] one can obtain all interleavings of two sequences of actions, `!Int.!String.end` and `!String.!Int.end`, as shown with the two example derivations below:

Example 8 (Composition with [act] and [sym] rules).

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{end}\circ\mathtt{end}\vdash\mathtt{end}}\,[\text{end}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{!String.end}\circ\mathtt{end}\vdash\mathtt{!String.end}}\,[\text{act}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{end}\circ\mathtt{!String.end}\vdash\mathtt{!String.end}}\,[\text{sym}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{!Int.end}\circ\mathtt{!String.end}\vdash\mathtt{!Int.!String.end}}\,[\text{act}]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{end}\circ\mathtt{end}\vdash\mathtt{end}}\,[\text{end}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{!Int.end}\circ\mathtt{end}\vdash\mathtt{!Int.end}}\,[\text{act}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{end}\circ\mathtt{!Int.end}\vdash\mathtt{!Int.end}}\,[\text{sym}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{!String.end}\circ\mathtt{!Int.end}\vdash\mathtt{!String.!Int.end}}\,[\text{act}]}{\emptyset,\emptyset,\ \emptyset,\ \mathtt{!Int.end}\circ\mathtt{!String.end}\vdash\mathtt{!String.!Int.end}}\,[\text{sym}]$$

A.2 Interleaving Composition and Multiparty Session Types

Consider a protocol, modelled using the session types syntax, that specifies the possible interactions between a user U and a remote instrument I (e.g., a camera). Below, S_I is the protocol specified from the perspective of I, which offers a menu with two choices: set or get. In case of set, I receives coordinates to update its own state, and in case of get, I sends a picture from the current coordinates. Protocol $\overline{S_U}$ is the dual of S_I (i.e., specified from the perspective of U).

$$S_I = \mu\mathtt{t}.\&\,\{\mathsf{set}\colon\mathtt{?coord.t},\mathsf{get}\colon\mathtt{!snap.t}\} \qquad \overline{S_U} = \mu\mathtt{t}.\oplus\{\mathsf{set}\colon\mathtt{!coord.t},\mathsf{get}\colon\mathtt{?snap.t}\}$$

The protocols above may have been designed top-down or extracted using our tool out of an existing system. Assume we need to modify the scenario above by introducing a proxy agent A, so that U and I will only interact via A. We need to: (1) express the protocols so that it is clear which roles are involved in each interaction, and (2) define the protocol for A. We address (1) by using a different instantiation of the protocol language to make roles explicit. For example, by fixing a set *Roles* of protocol roles, a set $\mathcal{T}$ of datatypes, and letting

$$\begin{aligned}\mathcal{P} = \{&ab\#\mathtt{T} \mid a,b \in \mathit{Roles}, \# \in \{!,?\}, T \in \mathcal{T}\}\\ &+ \in \{ab\# \mid a,b \in \mathit{Roles}, \# \in \{\oplus,\&\}\} \quad \mathcal{L} \subset \mathbb{N}\end{aligned}$$

Then we obtain protocols in a multiparty session types syntax (e.g., local types in [9]), where $ab!$ (resp. $ab?$) denotes a send action by a (resp. receive action by b) in an asynchronous interaction from a to b. Similarly for branching and selection. This instantiation allows us to model the following multiparty versions of S_I and S_U, respectively:

$$S_{AI} = \mu\mathtt{t}.AI\,\&\begin{cases}\mathsf{set}: & AI\,?\mathtt{coord.t},\\ \mathsf{get}: & IA\,!\mathtt{snap.t}\end{cases} \qquad S_{UA} = \mu\mathtt{t}.UA\oplus\begin{cases}\mathsf{set}: & UA\,!\mathtt{coord.t},\\ \mathsf{get}: & AU\,?\mathtt{snap.t}\end{cases}$$

We would like A to act as a server for U and as a client for I. Concretely, we could generate the protocol for A as a specific 'forwarding' interleaving of the dual of S_{AI} and the dual of S_{UA}. We use assertions to ensure that A sends I a menu choice only after having received one from U (and this must be the same choice received by U), and then A must reflect the behaviour of I following the given choice. The asserted protocols to be composed into the protocol for A are given below, where $\mathtt{assert}(set)/\mathtt{consume}(set)$ and $\mathtt{assert}(get)/\mathtt{consume}(get)$ express the desired correlation between branches, and $\mathtt{assert}(f)/\mathtt{consume}(f)$ model the forwarding pattern:

$$\mu\mathtt{t}.\ AI\ \oplus\left\{\begin{array}{l}\mathsf{set}:\mathtt{consume}(set).\,\mathtt{consume}(f).\ AI\,!\mathtt{coord.t},\\ \ \mathsf{get}:\ \mathtt{consume}(get).\ AI\,?\mathtt{snap.assert}(f).\,\mathtt{t}\end{array}\right\}$$

$$\mu\mathtt{t}.\ UA\,\&\left\{\begin{array}{l}\mathsf{set}:\mathtt{assert}(set).\,UA\,?\mathtt{coord.assert}(f).\,\mathtt{t},\\ \ \mathsf{get}:\mathtt{assert}(get).\,\mathtt{consume}(f).\ AU\,!\mathtt{snap.t}\end{array}\right\}$$

Using interleaving composition with correlating branching (Section 3.1) on the two protocols above we obtain the following interleaving composition specifying the protocol for A, where we omit the assertions for readability:

$$S_{UAI} = \mu\mathtt{t}.\,UA\,\&\begin{cases}\mathsf{set}: AI\ \oplus\{\mathsf{set}: UA\,?\mathtt{coord}.\ AI\,!\mathtt{coord}.\,\mathtt{t}\},\\ \mathsf{set}: AI\ \oplus\{\mathsf{get}: IA\,?\mathtt{snap}.\ AU\,!\mathtt{snap}.\,\mathtt{t}\}\end{cases}$$

Now that we have the protocols of the extended multiparty system we can use our tool to generate code for S_{UAI}, S_{AI}, and S_{UA}. Thanks to the extraction/migration functionality of our tool, pre-existing local code for U and I can be reused in the new code for U and I, where the specific endpoints for communication can be added manually. The tool does not yet support syntax for specifying roles, so in the case of an agent communicating with several parties such as S_{UAI} the direction of the communication would need to be specified manually.

In the multiparty scenario, correctness of the interaction structures can be checked using multiparty compatibility [15] and verification techniques [26].

B Basic properties of protocols

We first define some additional results in this appendix which are used for some of the key lemmas of this section. We have the following set of inversion lemmas on well-assertedness:

Lemma B1 (Prefix well-asserted inversion) $\forall A, A', S.\ A\ \{p.S\}\ A' \implies A\ \{S\}\ A'$.

Proof. There is only one rule that provides the well-assertedness of prefixing:

$$\frac{A\ \{S\}\ A'}{A\ \{p.S\}\ A'}$$

Lemma B2 (Branch well-asserted inversion) $\forall A, A', I, \{S_i\}_{i \in I}.$

$$A\ \{+\{\mathrm{l}_\mathrm{i} : S_i\}_{i \in I}\}\ A' \implies A' \equiv \exists \{A_i\}_{i \in I}. \bigcap_{i \in I} A_i \ \wedge\ \forall i \in I.\ A\ \{S_i\}\ A_i$$

Proof. There is only one rule that provides the well-assertedness of branching:

$$\frac{\forall i \in I.\ B\ \{S_i\}\ B_i}{B\ \{+\{\mathrm{l}_i : S_i\}_{i \in I}\}\ \bigcap_{i \in I} B_i)}$$

Given the antecedent of this lemma, we then have that $\{A_i\}_{i \in I} = \{B_i\}_{i \in I}$ and $A' = \bigcap_{i \in I} B_i$ then the premise provides the consequent of the lemma, $\forall i \in I.\ A\ \{S_i\}\ A_i$.

Lemma B3 (Assert well-asserted inversion) $\forall A, A', n, S.\ A\ \{\mathtt{assert}(n).S\}\ A' \implies A \cup \{n\}\ \{S\}\ A'$.

Proof. There is only one rule that provides the well-assertedness of assert:

$$\frac{A \cup \{n\}\ \{S\}\ A'}{A\ \{\mathtt{assert}(n).S\}\ A'}$$

Lemma B4 (Require well-asserted inversion) $\forall A, A', n, S.\ A\ \{\mathtt{require}(n).S\}\ A' \implies A\ \{S\}\ A' \wedge n \in A$.

Proof. There is only one rule that provides the well-assertedness of require:

$$\frac{B \cup \{n\}\ \{S\}\ B'}{B \cup \{n\}\ \{\mathtt{require}(n).S\}\ B'}$$

Thus we have that $A = B \cup \{n\}$ and so $n \in A$ and $A' = B'$ thus the premise provides the consequent of this lemma.

Lemma B5 (Consume well-asserted inversion) $\forall A, A', n, S.\ A\ \{\mathtt{consume}(n).S\}\ A' \implies n \in A \wedge\ A \setminus \{n\}\ \{S\}\ A'$.

Proof. There is only one rule that provides the well-assertedness of consume:

$$\frac{B\ \{S\}\ B' \qquad n \in (B \cup \{n\})}{B \cup \{n\}\ \{\mathtt{consume}(n).S\}\ B'}$$

Thus let $A = B \cup \{n\}$ and $A' = B'$ and therefore $n \in A$. The (first) premise of this rule then provides the consequent of this lemma.

Lemma B6 (Recursion well-asserted inversion) $\forall A, A', n, S.\ A\ \{\mu\mathtt{t}.S\}\ A' \implies A\ \{S\}\ A' \wedge A \subseteq A'$.

Proof. There is only one rule that provides the well-assertedness of recursion:

$$\frac{B\ \{S\}\ B \cup B'}{B\ \{\mu\mathtt{t}.S\}\ B \cup B'}$$

Thus we let $A = B$ and $A' = B \cup B'$ yielding $(A \subseteq A')$ and then premise provides the consequent of this lemma.

Lemma B7 (Well-asserted unfolding extension) *For all*

$$A\ \{S[\mu\mathtt{t}.S/\mathtt{t}]\}\ A' \Rightarrow A\ \{S[\mu\mathtt{t}.e.S/\mathtt{t}]\}\ A'$$

where e ranges over p, $\mathtt{require}(n), \mathtt{assert}(n), \mathtt{consume}(n), \mu\mathtt{t}'$ (in the last case then . becomesa scoping rather than a prefixing, by overloading).

Proof.
- (act) $S = p.S'$. Assuming $A\ \{p.S'[\mu\mathtt{t}.p.S'/\mathtt{t}]\}\ A'$ then by inversion (Lemma B1) this yields $A\ \{S'[\mu\mathtt{t}.p.S'/\mathtt{t}]\}\ A'$.
  By induction then $A\ \{S'[\mu\mathtt{t}.e.p.S'/\mathtt{t}]\}\ A'$, which then allows us to derive:

  $$\frac{A\ \{S'[\mu\mathtt{t}.e.p.S'/\mathtt{t}]\}\ A'}{A\ \{p.S'[\mu\mathtt{t}.e.p.S'/\mathtt{t}]\}\ A'}[\text{act}]$$

  which equals our goal

  $$A\ \{(p.S')[\mu\mathtt{t}.e.p.S'/\mathtt{t}]\}\ A'$$

  by the definition of syntactic substitution.
- (bra) $S = +\{\mathrm{l}_i : S_i\}_{i \in I}$ then by inversion (Lemma B2) this yields $A\ \{S_i\}\ A_i$ for all $i \in I$. with $A' \equiv \exists\{A_i\}_{i\in I}.\bigcap_{i \in I} A_i$.
  By induction on each then we have $A\ \{S_i[\mu\mathtt{t}.e.+\{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}\ A'_i$ allowing us to re-derive branching well-assertedness:

  $$\frac{A\ \{S_i[\mu\mathtt{t}.e. + \{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}\ A'_i}{A\ \{+\{\mathrm{l}_i : S_i[\mu\mathtt{t}.e. + \{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}_{i\in I}\}\ A'}[\text{bra}]$$

  which equals our goal by the definition of syntactic substitution:

  $$A\ \{(+\{\mathrm{l}_i : S_i\}_{i\in I})[\mu\mathtt{t}.e. + \{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}\ A'$$
- (assert) $S = \mathtt{assert}(n).S'$. Assuming $A\ \{\mathtt{assert}(n).S'[\mu\mathtt{t}.\mathtt{assert}(n).S'/\mathtt{t}]\}\ A'$ then by inversion (Lemma B3) this yields $A \cup \{n\}\ \{S'[\mu\mathtt{t}.\mathtt{assert}(n).S'/\mathtt{t}]\}\ A'$.
  By induction then $A \cup \{n\}\ \{S'[\mu\mathtt{t}.e.\mathtt{assert}(n).S'/\mathtt{t}]\}\ A'$, which then allows us to derive:

  $$\frac{A \cup \{n\}\ \{S'[\mu\mathtt{t}.e.\mathtt{assert}(n).S'/\mathtt{t}]\}\ A'}{A\ \{\mathtt{assert}(n).S'[\mu\mathtt{t}.e..S'/\mathtt{t}]\}\ A'}[\text{assert}]$$

  which equals our goal

  $$A\ \{(\mathtt{assert}(n).S')[\mu\mathtt{t}.e.\mathtt{assert}(n).S'/\mathtt{t}]\}\ A'$$

  by the definition of syntactic substitution.

- (require) $S = \texttt{require}(n).S'$. Assuming $A\ \{\texttt{require}(n).S'[\mu\texttt{t}.\texttt{require}(n).S'/\texttt{t}]\}\ A'$ then by inversion (Lemma B4) this yields $A\ \{S'[\mu\texttt{t}.\texttt{require}(n).S'/\texttt{t}]\}\ A'$ (with $n \in A$).
  By induction then $A\ \{S'[\mu\texttt{t}.e.\texttt{require}(n).S'/\texttt{t}]\}\ A'$, which then allows us to derive:

  $$\frac{A\ \{S'[\mu\texttt{t}.e.\texttt{require}(n).S'/\texttt{t}]\}\ A' \qquad n \in A}{A\ \{\texttt{require}(n).S'[\mu\texttt{t}.e.\texttt{require}(n).S'/\texttt{t}]\}\ A'}\,[\text{require}]$$

  which equals our goal

  $$A\ \{(\texttt{require}(n).S')[\mu\texttt{t}.e.\texttt{require}(n).S'/\texttt{t}]\}\ A'$$

  by the definition of syntactic substitution.
- (consume) $S = \texttt{consume}(n).S'$. Assuming $A\ \{\texttt{consume}(n).S'[\mu\texttt{t}.\texttt{consume}(n).S'/\texttt{t}]\}\ A'$ then by inversion (Lemma B5) this yields $A\backslash\{n\}\ \{S'[\mu\texttt{t}.\texttt{consume}(n).S'/\texttt{t}]\}\ A'$ (with $n \in A$).
  By induction then $A\backslash\{n\}\ \{S'[\mu\texttt{t}.e.\texttt{consume}(n).S'/\texttt{t}]\}\ A'$, which then allows us to derive:

  $$\frac{A \setminus \{n\}\ \{S'[\mu\texttt{t}.e.\texttt{consume}(n).S'/\texttt{t}]\}\ A' \qquad n \in A}{A\ \{\texttt{consume}(n).S'[\mu\texttt{t}.e.\texttt{consume}(n).S'/\texttt{t}]\}\ A'}\,[\text{consume}]$$

  which equals our goal

  $$A\ \{(\texttt{consume}(n).S')[\mu\texttt{t}.e.\texttt{consume}(n).S'/\texttt{t}]\}\ A'$$

  by the definition of syntactic substitution.
- (rec) $S = \mu\texttt{t}'.S'$ Assuming $A\ \{(\mu\texttt{t}'.S')[\mu\texttt{t}.(\mu\texttt{t}'.S')/\texttt{t}]\}\ A'$ then by inversion (Lemma B6) this yields $A\ \{S'[\mu\texttt{t}.(\mu\texttt{t}'.S')/\texttt{t}]\}\ A'$ (with $A \subseteq A'$).
  By induction then $A\ \{S'[\mu\texttt{t}.e.(\mu\texttt{t}'.S')/\texttt{t}]\}\ A'$, which then allows us to derive:

  $$\frac{A\ \{S'[\mu\texttt{t}.e.(\mu\texttt{t}'.S')/\texttt{t}]\}\ A'}{A\ \{\mu\texttt{t}'.S'[\mu\texttt{t}.\mu\texttt{t}'.S'/\texttt{t}]\}\ A'}\,[\text{rec}]$$

  (note the post-condition here satisfies $\exists A''.A \cup A'' = A'$ since $A \subseteq A'$).
  The conclusion equals our goal

  $$A\ \{(\mu\texttt{t}'.S')[\mu\texttt{t}.\mu\texttt{t}'.S'/\texttt{t}]\}\ A'$$

  by the definition of syntactic substitution and uniquness of binders property.
- (end) $S = \texttt{end}$ Assuming $A\ \{\texttt{end}[\mu\texttt{t}.\texttt{end}/\texttt{t}]\}\ A'$.
  Since $\texttt{end}[\mu\texttt{t}.\texttt{end}/\texttt{t}] = [\mu\texttt{t}.e.\texttt{end}/\texttt{t}]$ then this holds trivially from the assumption.
- (call) $S = \texttt{t}'$. Assuming $A\ \{\texttt{t}'[\mu\texttt{t}.\texttt{t}'/\texttt{t}]\}\ A'$.
  - $\texttt{t} = \texttt{t}'$ then $\texttt{t}'[\mu\texttt{t}.\texttt{t}'/\texttt{t}] = \texttt{t}[\mu\texttt{t}.\texttt{t}/\texttt{t}] = \mu\texttt{t}.\texttt{t}$ Such a protocol is not allowed by the syntactic guardness requirement, so this case is trivial (ex falso quodlibet).
  - $\texttt{t} \neq \texttt{t}'$ then $\texttt{t}'[\mu\texttt{t}.\texttt{t}'/\texttt{t}] = \texttt{t}'$ Then the goal hoalds trivially here as from the assumption we get $A\ \{\texttt{t}'[\mu\texttt{t}.e.\texttt{t}'/\texttt{t}]\}\ A'$.

Lemma B8 (Well-asserted unfolding extension under branch) *For all*

$$A\ \{S_i\}\ A' \wedge A\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}\}\ A' \Rightarrow A\ \{S_i[+\{\mathrm{l}_i : \mu\mathtt{t}.S_i\}_{i\in I}/\mathtt{t}]\}\ A'$$

Proof. By induction over the structure of S_i.

In each case, we proceed by induction, rebuilding well-assertedness (exactly as in the proof of Lemma B7). The key case is when we have a recursion variable that we are substituting into, $\mathtt{t}'$.

- $\mathtt{t}' \equiv \mathtt{t}$ then we substitute here: $\mathtt{t}'[+\{\mathrm{l}_i : \mu\mathtt{t}.S_i\}_{i\in I}/\mathtt{t}] = +\{\mathrm{l}_i : \mu\mathtt{t}.S_i\}_{i\in I}$ and so well-assertedness holds by the second conunct of the premise.
- $\mathtt{t}' \neq \mathtt{t}'$ then trivially $A\ \{\mathtt{t}'\}\ A'$ since $\mathtt{t}'[+\{\mathrm{l}_i : \mu\mathtt{t}.S_i\}_{i\in I}/\mathtt{t}] = \mathtt{t}'$

Lemma B9 (Well-asserted unfolding) *For all sets of names* A, A' *and protocols* S*, then:*

$$A\ \{S\}\ A' \implies A\ \{S[\mu\mathtt{t}.S/\mathtt{t}]\}\ A'$$

Proof. By induction on the structure of terms S:

- (act) $S = p.S'$ with assumption $A\ \{p.S'\}\ A'$. By Lemma B1 (inversion) we then have $A\ \{S'\}\ A'$.
  By induction on this judgment we have that $A\ \{S'[\mu\mathtt{t}.S'/\mathtt{t}]\}\ A'$.
  By Lemma B7 then this gives us $A\ \{S'[\mu\mathtt{t}.p.S'/\mathtt{t}]\}\ A'$ which we can use to build the well-assertedness derivation:

  $$\frac{A\ \{S'[\mu\mathtt{t}.p.S'/\mathtt{t}]\}\ A'}{A\ \{p.S'[\mu\mathtt{t}.p.S'/\mathtt{t}]\}\ A'}\text{[act]}$$

  which yields our goal by the definition of syntactic substitution.

- (bra) $S = +\{\mathrm{l}_i : S_i\}_{i\in I}$ with assumption $A\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}\}\ A'$.
  By inversion (Lemma B2) this yields $A\ \{S_i\}\ A_i$ for all $i \in I$. with $A' \equiv \exists\{A_i\}_{i\in I}.\bigcap_{i\in I} A_i$.
  By induction on each $i \in I$ then we have that $A\ \{S_i[\mu\mathtt{t}.S_i/\mathtt{t}]\}\ A'$. Applying Lemma B8 on each then this give us: $A\ \{S_i[+\{\mathrm{l}_i : \mu\mathtt{t}.S_i\}_{i\in I}/\mathtt{t}]\}\ A'$ which we can then use to build the well-assertedness derivation:

  $$\frac{\forall i \in I.\ A\ \{S_i[\mu\mathtt{t}. + \{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}\ A_i}{A\ \{+\{\mathrm{l}_i : S_i[\mu\mathtt{t}. + \{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}_{i\in I}\}\ \bigcap_{i\in I} A_i}$$

  which by the definition of syntactic substitution yields the goal:

  $$A\ \{(+\{\mathrm{l}_i : S_i\}_{i\in I})[\mu\mathtt{t}. + \{\mathrm{l}_i : S_i\}_{i\in I}/\mathtt{t}]\}\ \bigcap_{i\in I} A_i\ \text{[bra]}$$

- (require) $S = \texttt{require}(n).S'$ with assumption $A\ \{\texttt{require}(n).S'\}\ A'$.
  By Lemma B4 (inversion) we then have $A\ \{S'\}\ A'$ and $n \in A$.
  By induction on this judgment we have that $A\ \{S'[\mu\texttt{t}.S'/\texttt{t}]\}\ A'$.
  By Lemma B7 then this gives us $A\ \{S'[\mu\texttt{t}.\texttt{require}(n).S'/\texttt{t}]\}\ A'$ which we can use to build the well-assertedness derivation:

  $$\frac{A\ \{S'[\mu\texttt{t}.\texttt{require}(n).S'/\texttt{t}]\}\ A'}{A\ \{\texttt{require}(n).S'[\mu\texttt{t}.\texttt{require}(n).S'/\texttt{t}]\}\ A'}\text{[require]}$$

  (where $\exists A''.A = A'' \cup \{n\}$ sinch $n \in A$). which yields our goal by the definition of syntactic substitution.

- (consume) $S = \texttt{consume}(n).S'$ with assumption $A\ \{\texttt{consume}(n).S'\}\ A'$.
  By Lemma B5 (inversion) we then have $A \setminus \{n\}\ \{S'\}\ A'$ and $n \in A$.
  By induction on this judgment we have that $A \setminus \{n\}\ \{S'[\mu\texttt{t}.S'/\texttt{t}]\}\ A'$.
  By Lemma B7 then this gives us $A \setminus \{n\}\ \{S'[\mu\texttt{t}.\texttt{consume}(n).S'/\texttt{t}]\}\ A'$ which we can use to build the well-assertedness derivation:

  $$\frac{A \setminus \{n\}\ \{S'[\mu\texttt{t}.\texttt{consume}(n).S'/\texttt{t}]\}\ A' \qquad n \in A}{A\ \{\texttt{consume}(n).S'[\mu\texttt{t}.\texttt{consume}(n).S'/\texttt{t}]\}\ A'}\text{[consume]}$$

  which yields our goal by the definition of syntactic substitution.

- (assert) $S = \texttt{assert}(n).S'$ with assumption $A\ \{\texttt{assert}(n).S'\}\ A'$.
  By Lemma B3 (inversion) we then have $A \setminus \{n\}\ \{S'\}\ A'$ and $n \in A$.
  By induction on this judgment we have that $A \cup \{n\}\ \{S'[\mu\texttt{t}.S'/\texttt{t}]\}\ A'$.
  By Lemma B7 then this gives us $A \cup \{n\}\ \{S'[\mu\texttt{t}.\texttt{assert}(n).S'/\texttt{t}]\}\ A'$ which we can use to build the well-assertedness derivation:

  $$\frac{A \setminus \{n\}\ \{S'[\mu\texttt{t}.\texttt{assert}(n).S'/\texttt{t}]\}\ A'}{A\ \{\texttt{assert}(n).S'[\mu\texttt{t}.\texttt{assert}(n).S'/\texttt{t}]\}\ A'}\text{[assert]}$$

  which yields our goal by the definition of syntactic substitution.

- (rec) $S = \mu\texttt{t}'.S'$ with assumption $A\ \{\mu\texttt{t}'.S'\}\ A'$.
  By Lemma B6 (inversion) we then have $A\ \{S'\}\ A'$ and $A \subseteq A'$.
  By induction on this judgment we have that $A\ \{S'[\mu\texttt{t}.S'/\texttt{t}]\}\ A'$.
  By Lemma B7 then this gives us $A\ \{S'[(\mu\texttt{t}.(\mu\texttt{t}'.S'))/\texttt{t}]\}\ A'$ which we can use to build the well-assertedness derivation:

  $$\frac{A\ \{S'[\mu\texttt{t}.\mu\texttt{t}'.S'/\texttt{t}]\}\ A'}{A\ \{\mu\texttt{t}'.S'[\mu\texttt{t}.\mu\texttt{t}'.S'/\texttt{t}]\}\ A'}\text{[rec]}$$

  (leveraging $A \subseteq A'$) which yields our goal by the definition of syntactic substitution.

- (end) $\dfrac{-}{A\ \{\texttt{end}\}\ A}$ Trivial since $\texttt{end}[\mu\texttt{t}.\texttt{end}/\texttt{t}] = \texttt{end}$.

- (call) $S = \mathtt{t}'$ with assumption $A\ \{\mathtt{t}'\}\ A'$ thus $A \equiv A'$.
  Case
  - $\mathtt{t} = \mathtt{t}'$ thus $\mathtt{t}'[\mu\mathtt{t}.\mathtt{t}'/\mathtt{t}] = \mu\mathtt{t}.\mathtt{t}$. Then we can apply construct well-assertedness by the derivation:

$$\frac{A\ \{\mathtt{t}\}\ A}{A\ \{\mu\mathtt{t}.\mathtt{t}\}\ A}\,[\text{rec}]$$

  - $\mathtt{t} \neq \mathtt{t}'$ then $\mathtt{t}'[\mu\mathtt{t}.\mathtt{t}'/\mathtt{t}] = \mathtt{t}'$ therefore using the assumption we have $A\ \{\mathtt{t}[\mu\mathtt{t}.\mathtt{t}'/\mathtt{t}]\}\ A'$.

C Proof of Lemmas 1, 2, 3 on well-assertedness and progress

Lemma 1 (Reduction preserves well-assertedness). *If* $A\ \{S\}\ A'$ *and there is a reduction* $(A, S) \xrightarrow{\ell} (A'', S')$ *then* $\exists A''' \supseteq A'.\ A''\ \{S'\}\ A'''$.

Proof. By induction on the structure of $A\ \{S\}\ A'$.

- (act)

$$\frac{A\ \{S\}\ A'}{A\ \{p.S\}\ A'}$$

  Then the only possible reduction is $\langle\mathtt{Inter}\rangle$:

$$(A, p.S) \xrightarrow{p} (A, S)$$

  Therefore we can conclude with the premise of $A\ \{S\}\ A'$ which shows that S is well-asserted (and trivially $A \supseteq A$).
- (bra)

$$\frac{\forall i \in I.\ A\ \{S_i\}\ A_i}{A\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}\}\ \bigcap_{i\in I} A_i}$$

  Then the only possible reduction is $\langle\mathtt{Branch}\rangle$:

$$(A, +\{\mathrm{l_i} : S_i\}_{i\in I}) \xrightarrow{+\mathrm{l_j}} (A, S_j)\ \ (j \in I)$$

  Therefore we can conclude with the premise $A\ \{S_j\}\ A_j$ which shows that S_j is well-asserted (and $A_j \supseteq \bigcap_{i\in I} A_i$ since $j \in I$).
- (require)

$$\frac{A \cup \{n\}\ \{S\}\ A'}{A \cup \{n\}\ \{\mathtt{require}(n).S\}\ A'}$$

  Then the only possible reduction is $\langle\mathtt{Require}\rangle$ with:

$$(A \cup \{n\}, \mathtt{require}(n).S) \xrightarrow{\mathtt{require}(n)} (A \cup \{n\}, S)\ \ (n \in (A \cup \{n\}))$$

  (note the trivial satisfaction of the side condition here). Therefore we can conclude with the premise $A\cup\{n\}\ \{S\}\ A'$ which shows that S is well-asserted (and trivially $A' \supseteq A'$).

- (consume)

  $$\frac{A\ \{S\}\ A'}{A \cup \{n\}\ \{\texttt{consume}(n).S\}\ A'}$$

  Then the only possible reduction is $\langle\texttt{Consume}\rangle$:

  $$(A \cup \{n\}, \texttt{consume}(n).S) \xrightarrow{\texttt{consume}(n)} ((A \cup \{n\}) \setminus \{n\}, S) \ \ (n \in (A \cup \{n\}))$$

  Thus since $(A \cup \{n\}) \setminus \{n\} = A$ we can conclude with the premise $A\ \{S\}\ A'$ showing that S is well-asserted (and trivially $A' \supseteq A'$).
- (assert)

  $$\frac{A \cup \{n\}\ \{S\}\ A'}{A\ \{\texttt{assert}(n).S\}\ A'}$$

  Then the only possible reduction is $\langle\texttt{Assert}\rangle$:

  $$(A, \texttt{assert}(n).S) \xrightarrow{\texttt{assert}(n)} (A \cup \{n\}, S)$$

  Thus we can conclude with the premise $A \cup \{n\}\ \{S\}\ A'$ showing that S is well-asserted (and trivially $A' \supseteq A'$).
- (rec)

  $$\frac{A\ \{S\}\ A \cup A'}{A\ \{\mu\texttt{t}.S\}\ A \cup A'}$$

  Then the only possible reduction is $\langle\texttt{Rec}\rangle$:

  $$\frac{(A, S) \xrightarrow{\ell} (A'', S')}{(A, \mu\texttt{t}.S) \xrightarrow{\ell} (A'', S'[\mu\texttt{t}.S/\texttt{t}])} \qquad (*)$$

  We now proceed by an inner induction on the structure of S to prove that $\exists A''' \supseteq A \cup A'.\ A''\ \{S'[\mu\texttt{t}.S/\texttt{t}]\}\ A'''$.
  - (prefix) $S = p.S_1$ thus $A\ \{p.S_1\}\ A \cup A'$, and thus $(*)$ must be the reduction:

    $$\frac{\dfrac{}{(A, p.S_1) \xrightarrow{p} (A, S_1)}\langle\texttt{Inter}\rangle}{(A, \mu\texttt{t}.p.S_1) \xrightarrow{p} (A, S_1[\mu\texttt{t}.p.S_1/\texttt{t}])}\langle\texttt{Rec}\rangle$$

    thus $A'' = A$.
    By lemma B9 on $A\ \{p.S_1\}\ A \cup A'$ then $A\ \{p.S_1[\mu\texttt{t}.p.S_1/\texttt{t}]\}\ A \cup A'$ (**)
    Then by the definition of substitution and lemma B1 (inversion of prefix well-assertedness) on (**) we get: $A\ \{S_1[\mu\texttt{t}.p.S_1/\texttt{t}]\}\ A \cup A'$ providing the goal with $A''' = A \cup A'$.
  - (branch) $S = +\{\mathrm{l_i} : S_i\}_{i \in I}$ thus $A\ \{+\{\mathrm{l_i} : S_i\}_{i \in I}\}\ A \cup A'$, and thus $(*)$ must be the reduction:

    $$\frac{\dfrac{}{(A, +\{\mathrm{l_i} : S_i\}_{i \in I}) \xrightarrow{+\mathrm{l_j}} (A, S_j) \qquad (j \in I)}\langle\texttt{Branch}\rangle}{(A, \mu\texttt{t}.p. + \{\mathrm{l_i} : S_i\}_{i \in I}) \xrightarrow{p} (A, S_j[\mu\texttt{t}. + \{\mathrm{l_i} : S_i\}_{i \in I}/\texttt{t}])}\langle\texttt{Rec}\rangle$$

thus $A'' = A$.
By lemma B9 on $A\ \{+\{\mathrm{l_i} : S_i\}_{i\in I}\}\ A\cup A'$ and unfolding the definition of syntactic substitution then $A\ \{+\{\mathrm{l_i} : S_i[\mu\mathtt{t}.+\{\mathrm{l_i} : S_i\}_{i\in I}/\mathtt{t}]\}_{i\in I}\}\ A\cup A'$ (**)
Then by the definition of substitution and lemma B2 (inversion of branch well-formendess) on (**) we get: $\exists\{A_i\}_{i\in I}.A\cup A' \equiv \bigcap_{i\in I} A_i \ \ \wedge\ \ \forall i \in I.\ A\ \{S_i[\mu\mathtt{t}.+\{\mathrm{l_i} : S_i\}_{i\in I}/\mathtt{t}]\}\ A_i$.

Then taking $i = j$ we get $A\ \{S_j[\mu\mathtt{t}.+\{\mathrm{l_i} : S_i\}_{i\in I}/\mathtt{t}]\}\ A_j$ providing the goal of this lemma with $A''' = A_j$ and $A_j \supseteq A\cup A' = \bigcap_{i\in I} A_i$ since $j \in I$.

- (assert) $S = \mathtt{assert}(n).S_1$ thus $A\ \{\mathtt{assert}(n).S_1\}\ A\cup A'$, and thus $(*)$ must be the reduction:

  $$\dfrac{\dfrac{}{(A, \mathtt{assert}(n).S_1) \xrightarrow{\mathtt{assert}(n)} (A\cup\{n\}, S_1)}\langle\mathtt{Assert}\rangle}{(A, \mu\mathtt{t}.\mathtt{assert}(n).S_1) \xrightarrow{\mathtt{assert}(n)} (A\cup\{n\}, S_1[\mu\mathtt{t}.\mathtt{assert}(n).S_1/\mathtt{t}])}\langle\mathtt{Rec}\rangle$$

  thus $A'' = A\cup\{n\}$.
  By lemma B9 on $A\ \{\mathtt{assert}(n).S_1\}\ A\cup A'$ then (unfolding substitution) $A\ \{\mathtt{assert}(n).S_1[\mu\mathtt{t}.\mathtt{assert}(n).S_1/\mathtt{t}]\}\ A\cup A'$ (**)
  Then by the definition of substitution and lemma B3 (inversion of assert well-assertedness) on (**) we get: $A\cup\{n\}\ \{S_1[\mu\mathtt{t}.\mathtt{assert}(n).S_1/\mathtt{t}]\}\ A\cup A'$ providing the goal with $A''' = A\cup A'$.
- (require) $S = \mathtt{require}(n).S_1$ thus $A\ \{\mathtt{require}(n).S_1\}\ A\cup A'$, and thus $(*)$ must be the reduction:

  $$\dfrac{\dfrac{}{(A, \mathtt{require}(n).S_1) \xrightarrow{\mathtt{require}(n)} (A, S_1)}\langle\mathtt{Require}\rangle\ \ (n\in A)}{(A, \mu\mathtt{t}.\mathtt{require}(n).S_1) \xrightarrow{\mathtt{require}(n)} (A, S_1[\mu\mathtt{t}.\mathtt{require}(n).S_1/\mathtt{t}])}\langle\mathtt{Rec}\rangle$$

  and thus $A'' = A$.
  By lemma B9 on $A\ \{\mathtt{require}(n).S_1\}\ A\cup A'$ then (unfolding substitution) $A\ \{\mathtt{require}(n).S_1[\mu\mathtt{t}.\mathtt{require}(n).S_1/\mathtt{t}]\}\ A\cup A'$ (**)
  Then by the definition of substitution and lemma B4 (inversion of require well-assertedness) on (**) with $n \in A$ we get:
  $A\ \{S_1[\mu\mathtt{t}.\mathtt{require}(n).S_1/\mathtt{t}]\}\ A\cup A'$ matching the goal with $A''' = A\cup A'$.
- (consume) $S = \mathtt{consume}(n).S_1$ thus $A\ \{\mathtt{consume}(n).S_1\}\ A\cup A'$, and thus $(*)$ must be the reduction:

  $$\dfrac{\dfrac{}{(A, \mathtt{consume}(n).S_1) \xrightarrow{\mathtt{consume}(n)} (A\setminus\{n\}, S_1)}\langle\mathtt{Consume}\rangle\ \ (n\in A)}{(A, \mu\mathtt{t}.\mathtt{consume}(n).S_1) \xrightarrow{\mathtt{consume}(n)} (A\setminus\{n\}, S_1[\mu\mathtt{t}.\mathtt{consume}(n).S_1/\mathtt{t}])}\langle\mathtt{Rec}\rangle$$

  and thus $A'' = A\setminus\{n\}$.
  By lemma B9 on $A\ \{\mathtt{consume}(n).S_1\}\ A\cup A'$ then (unfolding substitution) $A\ \{\mathtt{consume}(n).S_1[\mu\mathtt{t}.\mathtt{consume}(n).S_1/\mathtt{t}]\}\ A\cup A'$ (**)

Then by the definition of substitution and lemma B5 (inversion of consume well-assertedness) on (**) with $n \in A$ we get:
$A \setminus \{n\}\ \{S_1[\mu\mathtt{t}.\mathtt{consume}(n).S_1/\mathtt{t}]\}\ A \cup A'$ matching the goal with $A''' = A \cup A'$.

- (rec) $S = \mu\mathtt{t}_1.S_1$ thus $A\ \{\mu\mathtt{t}_1.S_1\}\ A \cup A'$, and thus $(*)$ must be the reduction:

$$\dfrac{\dfrac{(A, S_1) \xrightarrow{\ell} (A_1, S_1')}{(A, \mu\mathtt{t}_1.S_1) \xrightarrow{\ell} (A_1, S_1'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1])}}{(A, \mu\mathtt{t}.\mu\mathtt{t}_1.S_1) \xrightarrow{\mathtt{consume}(n)} (A_1, S_1'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1][\mu\mathtt{t}.\mu\mathtt{t}_1.S_1/\mathtt{t}])}\langle\mathtt{Rec}\rangle$$

and thus $A'' = A_1$.
By lemma B9 on $A\ \{\mu\mathtt{t}_1.S_1\}\ A \cup A'$ then (unfolding substitution and since $\mathtt{t} \neq \mathtt{t}_1$)
$A\ \{\mu\mathtt{t}_1.S_1[\mu\mathtt{t}.\mu\mathtt{t}_1.S_1/\mathtt{t}]\}\ A \cup A'$ (**)
Then by the definition of substitution and lemma B6 (inversion of consume well-assertedness) on (**) with $A \subseteq A \cup A'$ by usual set theory laws, then we get:
$A\ \{S_1[\mu\mathtt{t}.\mu\mathtt{t}_1.S_1/\mathtt{t}]\}\ A \cup A'$ matching the goal with $A''' = A \cup A'$.

Thus by this sublemma we conclude that $\exists A''' \supseteq A \cup A'.\ A''\ \{S'[\mu\mathtt{t}.S/\mathtt{t}]\}\ A'''$.

- (call)

$$\dfrac{-}{A\ \{\mathtt{t}\}\ A}$$

A recursive variable on its own cannot reduce thus this case holds trivially since the premise is false.

- (end)

$$\dfrac{-}{A\ \{\mathtt{end}\}\ A}$$

The terminated process `end` cannot reduce thus this case holds trivially since the premise is false.

Lemma 2 (Well-asserted protocols are not stuck). *If $A\ \{S\}$ and S is closed with respect to recursion variables ($\mathsf{fv}(S) = \emptyset$) then (A, S) is not stuck.*

Proof. We proceed by structural induction on the derivation of well-assertedness $A\ \{S\}\ A'$ (and thus simultaneously on the structure of S since every syntactic construction has one well-assertedness rule):

- $S = \mathtt{end}$ there are no further reductions possible and the thesis trivially holds: we conclude with the first conjunct of the definition 4.
- $S = \mathtt{t}$ then progress is trivial since the premise is that the S is closed.
- $S = p.S'$ with:

$$\dfrac{A\ \{S'\}\ A'}{A\ \{p.S'\}\ A'}\quad[\text{act}]$$

Thus S can reduce by $\langle\mathtt{Inter}\rangle$ as $(A, p'.S') \xrightarrow{p} (A, S')$

- $S = +\{\mathrm{l_i} : S_i\}_{i \in I}$ with:

$$\frac{\forall i \in I.\ A\ \{S_i\}\ A_i}{A\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}\}\ \bigcap_{i\in I} A_i)} \tag{2}$$

  where $A' = \bigcap_{i\in I} A_i$. Thus, S can reduce by $\langle\texttt{Branch}\rangle$ to (A, S_j) for any $j \in I$.
- $S = \texttt{assert}(n).S'$ with

$$\frac{A \cup \{n\}\ \{S'\}\ A'}{A\ \{\texttt{assert}(n).S'\}\ A'}$$

  Thus S can reduce by $\langle\texttt{Assert}\rangle$ to $(A \cup \{n\}, S')$
- $S = \texttt{consume}(n).S'$ with

$$\frac{A \setminus \{n\}\ \{S'\}\ A' \qquad n \in A}{A\ \{\texttt{consume}(n).S'\}\ A'}$$

  Thus S can reduce by $\langle\texttt{Consume}\rangle$ to $(A \setminus \{n\}, S')$ since well-assertedness gives us that $n \in A$. to give us the side condition of this reduction rule.
- $S = \texttt{require}(n).S'$ with

$$\frac{A \cup \{n\}\ \{S'\}\ A'}{A \cup \{n\}\ \{\texttt{require}(n).S'\}\ A'}$$

  Thus S can reduce by $\langle\texttt{Require}\rangle$ to $(A \cup \{n\}, S')$ since well-assertedness gives us that $n \in (A \cup \{n\})$ to give us the side condition of this reduction rule.
- $S = \mu\texttt{t}.S'$ with:

$$\frac{A\ \{S'\}\ A'}{A\ \{\mu\texttt{t}.S'\}\ A'} \tag{3}$$

  By induction on the first premise we get that $S' = \texttt{end}$ or $(A, S') \to (A''', S'')$.
  - In the case of $S' = \texttt{end}$ then we have $S = \mu\texttt{t}.\texttt{end}$ which by structural congruence (definition 2) then means $S = \texttt{end}$.
  - In the case of $(A, S') \to (A''', S'')$ this provides the premise of the $\langle\texttt{Rec}\rangle$ rule such that S can reduce to $(A''', S''[\mu\texttt{t}.S'/\texttt{t}])$.

Below we write:

$$(A, S) \to^n (A', S') \ \text{ if } \begin{cases} n = 1 & \exists\ell.(A, S) \xrightarrow{\ell} (A', S') \\ n > 1 & \exists\ell.(A, S) \xrightarrow{\ell} (A'', S'') \wedge (A'', S'') \to^{n-1} (A', S') \end{cases}$$

Lemma 3 (Progress of very-well-asserted protocols). *If S is very-well-asserted (i.e., $\emptyset\ \{S\}$) and closed then it exhibits* progress.

Proof. We proceed by induction on the length of the reduction sequence n and prove a stronger lemma:

If S is *closed* ($\mathsf{fv}(S) = \emptyset$) and *very-well-asserted* ($\exists A'.\ \emptyset\ \{S\}\ A'$) then S has *progress*, i.e., $\forall A, n, S'$ if $(\emptyset, S) \to^n (A, S')$ then ($S' = \texttt{end} \vee (\exists A'', S''.(A, S') \to (A'', S'')$ **and** $\exists A'''.\ A''\ \{S''\}\ A'''$).

- $n = 0$.
  Thus, $A = \emptyset$ and $S' = S$.
  By lemma 2, with $\emptyset\ \{S\}\ A'$ then we get that $S = \texttt{end} \vee \exists A'', S''.(A, S) \to (A'', S'')$.
  In the latter case (of a reduction), we then apply lemma 1 to get that $\exists A'''.\ A''\ \{S''\}\ A'''$.
- $n = k + 1$
  Then we have the assumption that $(\emptyset, S) \to^{k+1} (A_{k+1}, S'_{k+1})$ and thus there exists $(\emptyset, S) \to^{k} (A_k, S'_k) \to (A_{k+1}, S'_{k+1})$.
  We can apply the lemma inductively on $(\emptyset, S) \to^{k} (A_k, S'_k)$ (i.e., the $n = k$ case) to get that $S'_k = \texttt{end}$ (not possible because of the $k+1$ reduction here) or $\exists A'', S'.(A_k, S'_k) \to (A'_{k+1}, S''_{k+1})$ and that $\exists A_1.\ A'_{k+1}\ \{S''_{k+1}\}\ A_1$.
  Since reduction is deterministic we have that: $(A_{k+1}, S'_{k+1}) = (A'_{k+1}, S''_{k+1})$.
  The goal here is to show that either $S'_{k+1} = \texttt{end}$ or that $\exists A_{k+2}, S'_{k+2}.(A_{k+1}, S'_{k+1}) \to (A_{k+2}, S'_{k+2})$ where $\exists A_2.\ A_{k+2}\ \{S'_{k+2}\}\ A_2$.

  By lemma 2 (local progress) with $A'_{k+1}\ \{S'_{k+1}\}\ A_1$ then we have that $(S'_{k+1} = \texttt{end}) \vee \exists A_{k+2}, S'_{k+2}.(A_{k+1}, S'_{k+1}) \to (A_{k+2}, S'_{k+2})$. In the case of the left disjunct we are done. In the case of the right disjunct, we then have the remaining piece of evidence via lemma 1 (reduction preserves well-assertedness) with the $A'_{k+1}\ \{S'_{k+1}\}\ A_1$ and $(A_{k+1}, S'_{k+1}) \to (A_{k+2}, S'_{k+2})$ which gives us that $\exists A_2.\ A_{k+2}\ \{S'_{k+2}\}\ A_2$.
  Thus we are done.

D Proof of Proposition 3 on validity of composition

Proposition 3 (Validity) *If* $T_L, T_R, \emptyset, S_1 \circ S_2 \vdash S$ *then* S *is very-well-asserted.*

Proof. Proposition 3 follows immediately from Lemma 5, given in this section, setting $A = \emptyset$. The proof of Lemma 5 relies on an auxiliary lemma, Lemma 4, given below. Lemma 4 makes use of environment weakening (Proposition Proposition 2) given in earlier sections.

Lemma 4. *If* $A_0\ \{S\}\ A$ *and* $A\ \{S'\}$ *then* $A_0\ \{S[S'/\texttt{end}]\}$.

Proof. The proof is by induction on the size of S, proceeding by case analysis on the syntax of S.

Base cases There are two base cases: $S = \texttt{end}$ and $S = \texttt{t}$. If $S = \texttt{end}$, by [end] $A_0 = A$. The thesis follows then immediately from the hypothesis $A\ \{S'\}$ since $\texttt{end}[S'/\texttt{end}] = S'$ If $S = \texttt{t}$ the thesis follows immediately by hypothesis $A_0\ \{S\}\ A$ since $S[S'/\texttt{end}] = S$.

Inductive cases The inductive cases are analyzed below:

- Case $S = p.S''$. By [act] on hypothesis $A_0 \; \{p.S''\} \; A$ follows (as premise)
$$A_0 \; \{S''\} \; A \tag{4}$$
By induction, (Equation (4)) and hypothesis $A \; \{S'\}$ follows
$$A_0 \; \{S''[S'/\mathtt{end}]\} \tag{5}$$
By applying rule [act] to (Equation (5)) obtain $A_0 \; \{p.S''[S'/\mathtt{end}]\}$ as required.
- Case $S = \mathtt{require}(n).S''$. By [assume] on hypothesis $A_0 \; \{\mathtt{require}(n).S''\} \; A$ follows (as premise)
$$A_0 \; \{S''\} \; A \tag{6}$$
with $n \in A_0$. By induction, (Equation (6)) and hypothesis $A \; \{S'\}$ follows
$$A_0 \; \{S''[S'/\mathtt{end}]\} \tag{7}$$
By applying rule [assume] to (Equation (7)) – observe that $n \in A_0$ – obtain
$$A_0 \; \{\mathtt{require}(n).S''[S'/\mathtt{end}]\}$$
as required.
- Case $S = \mathtt{consume}(n).S''$. By [consume] on hypothesis $A_0 \; \{\mathtt{consume}(n).S''\} \; A$ follows (as premise) $n \in A_0$ and
$$A_0 \setminus \{n\} \; \{S''\} \; A \tag{8}$$
By induction, (Equation (8)) and hypothesis $A \; \{S'\}$ follows
$$A_0 \setminus \{n\} \; \{S''[S'/\mathtt{end}]\} \tag{9}$$
By applying rule [consume] to (Equation (9)) obtain $A_0 \; \{\mathtt{consume}(n).S''[S'/\mathtt{end}]\}$ as required.
- Case $S = \mathtt{assert}(n).S''$. By [assert] on hypothesis $A_0 \; \{\mathtt{assert}(n).S''\} \; A$ follows (as premise)
$$A_0 \cup \{n\} \; \{S''\} \; A \tag{10}$$
By induction, (Equation (10)) and hypothesis $A \; \{S'\}$ follows
$$A_0 \cup \{n\} \; \{S''[S'/\mathtt{end}]\} \tag{11}$$
By applying rule [assert] to (Equation (11)) obtain $A_0 \; \{\mathtt{assert}(n).S''[S'/\mathtt{end}]\}$ as required.
- Case $S = \mu\mathtt{t}.S''$. By [rec] on hypothesis $A_0 \; \{\mu\mathtt{t}.S''\} \; A$ follows (as premise)
$$A_0 \; \{S''\} \; A \tag{12}$$
By induction, (Equation (12)) and hypothesis $A \; \{S'\}$ follows
$$A_0 \; \{S''[S'/\mathtt{end}]\} \tag{13}$$
By applying rule [rec] to (Equation (13)) obtain $A_0 \; \{\mu.S''[S'/\mathtt{end}]\}$ as required.

- Case $S = +\{\mathrm{l}_i : S_i\}_{i\in I}$. By [bra] on hypothesis $A_0\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}\}\ A$ follows (as premise)

$$\forall i \in I.\ A_0\ \{S_i\}\ A_i \tag{14}$$

  for some $\{A_i\}_{i\in I}$. Since by [bra] applied in (Equation (14)) $\bigcap_{i\in I} A_i = A$ then

$$\forall i \in I.\ A \subseteq A_i \tag{15}$$

  By Proposition 1 (environment weakening), (Equation (14)), and (Equation (15)), follows

$$A_i\ \{S_i\}\ \text{ for all } i \in I. \tag{16}$$

  By induction, (Equation (14)) and (Equation (16)) give

$$\forall i \in I.\ A_0\ \{S_i[S'/\mathtt{end}]\} \tag{17}$$

  By applying (Equation (17)) as a premise of [bra] obtain $A_0\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}[S'/\mathtt{end}]\}$ as required.

Lemma 5. *If* $T_L, T_R, A, S_1 \circ S_2 \vdash S$ *then* $A\{S\}$.

Proof. The proof is by induction on the derivation, proceeding by case analysis on the last rule (Definition 7) applied.

Base cases. There are two base cases: the last application was rule [end] or [call]. If the last (and only) rule applied was [end] in Definition 7 then $S = \mathtt{end}$ and by rule [end] in Definition 6 $A\ \{\mathtt{end}\}$ as required. If the last (and only) rule applied was [call] in Definition 7 then $S = \mathtt{t}$ and by rule [call] in Definition 6 $A\ \{\mathtt{t}\}$ as required.

Inductive cases. We show below the inductive cases.

- Case last rule is [act]. The conclusion is on the form $T_L, T_R, A, p.S_1' \circ S_2 \vdash p.S'$ with premise

$$T_L, T_R, A, S_1' \circ S_2 \vdash S' \tag{18}$$

  By induction, from (Equation (18)) it follows:

$$A\ \{S'\} \tag{19}$$

  By applying rule [act] in Definition 6 to (Equation (19)) it follows $A\ \{p.S'\}$ as required.
- Case last rule is [sym]. The conclusion is on the form $T_L, T_R, A, S_1 \circ S_2 \vdash S$ with premise

$$T_L, T_R, A, S_2 \circ S_1 \vdash S \tag{20}$$

  By induction, from (Equation (20)) it follows that $A\{S\}$ as required.

- Case last rule is [require]. The conclusion is on the form

$$T_L,\, T_R,\, \{n\} \cup A',\, \texttt{require}(n).S_1' \circ S_2 \vdash \texttt{require}(n).S'$$

with premise

$$T_L,\, T_R,\, \{n\} \cup A',\, S_1' \circ S_2 \vdash S' \tag{21}$$

By induction, from (Equation (21)) follows $\{n\} \cup A' \; \{S'\}$. By using $\{n\} \cup A' \; \{S'\}$ as a premise for rule [require] in Definition 6 we obtain $\{n\} \cup A' \; \{\texttt{require}(n).S'\}$ as required.

- Case last rule is [consume]. The conclusion is on the form

$$T_L,\, T_R,\, \{n\} \cup A',\, \texttt{consume}(n).S_1' \circ S_2 \vdash \texttt{consume}(n).S'$$

with premise

$$T_L,\, T_R,\, A',\, S_1' \circ S_2 \vdash S' \tag{22}$$

By induction, from (Equation (22)) it follows

$$A' \; \{S'\} \tag{23}$$

By applying (Equation (23)) as a premise for rule [consume] in Definition 6 obtain $\{n\} \cup A' \; \{\texttt{consume}(n).S'\}$ as required.

- Case last rule is [assert]. The conclusion is on the form

$$T_L,\, T_R,\, A,\, \texttt{assert}(n).S_1' \circ S_2 \vdash \texttt{assert}(n).S'$$

with premise

$$T_L,\, T_R,\, A \cup \{n\},\, S_1' \circ S_2 \vdash S' \tag{24}$$

By induction, from (Equation (24)) it follows

$$A \cup \{n\} \; \{S'\} \tag{25}$$

By applying rule [act] in Definition 6 to (Equation (19)) it follows $A \; \{\texttt{assert}(n).S'\}$ as required.

- Case last rule is [bra] (without weakening). The conclusion is on the form

$$T_L,\, T_R,\, A,\, +\{\mathrm{l}_i : S_i\}_{i\in I} \circ S_2 \vdash +\{\mathrm{l}_i : S_i'\}_{i\in I}$$

with premise

$$\forall i \in I.\; T_L,\, T_R,\, A,\, S_i \circ S_2 \vdash S_i' \tag{26}$$

By induction, from (Equation (26)) it follows:

$$\forall i \in I.\; A \; \{S_i'\} \tag{27}$$

By applying (Equation (27)) as premise of [bra] in Definition 6 it follows $A \; \{+\{\mathrm{l}_i : S_i'\}_{i\in I}\}$ as required.

- Case last rule is [bra] (with weakening). The conclusion is on the form

  $$T_L,\ T_R,\ A,\ +\{\mathrm{l}_i : S_i\}_{i\in I} \circ S_2 \vdash +\{\mathrm{l}_i : S'_i\}_{i\in I_A} \cup +\{\mathrm{l}_i : S_i\}_{i\in I_B}$$

  with premises $I_A \cup I_B = I$, $I_A \cap I_B = \emptyset$ and $I_A = \emptyset$, and

  $$\forall i \in I_A.\ T_L,\ T_R,\ A,\ S_i \circ S_2 \vdash S'_i \tag{28}$$

  $$\forall i \in I_B.\ A\ \{S_i\} \tag{29}$$

  By induction, from (Equation (28)) it follows:

  $$\forall i \in I_A.\ A\ \{S'_i\} \tag{30}$$

  By applying (Equation (29)) and (Equation (30)) as premise of [bra] in Definition 6 it follows $A\ \{+\{\mathrm{l}_i : S'_i\}_{i\in I_A} \cup +\{\mathrm{l}_i : S_i\}_{i\in I_B}\}$ as required.
- Case last rule is [rec1]. The conclusion is of the form

  $$T_L,\ T_R,\ A,\ \mu\mathtt{t}_1.S'_1 \circ \mu\mathtt{t}_2.S'_2 \vdash \mu\mathtt{t}_1.S$$

  with premise

  $$T_L \cup \{\mathtt{t}_1\},\ T_R,\ A,\ S'_1 \circ \mu\mathtt{t}_2.S'_2 \vdash S \qquad A\ \{\mu\mathtt{t}_1.S\} \tag{31}$$

  The thesis follows by condition $A\ \{\mu\mathtt{t}_1.S\}$ in the premise (Equation (31)).
- Case last rule is [rec2]. The conclusion is of the form

  $$T_L,\ T_R,\ A,\ \mu\mathtt{t}_1.S'_1 \circ S'_2 \vdash S$$

  with premise

  $$T_L,\ T_R \cup \{\mathtt{t}\},\ A,\ S'_1[\mathtt{t}_1/\mathtt{t}] \circ S'_2 \vdash S$$

  . By induction $A\ \{S_1\}$ which is the thesis.
- Case last rule is [rec3]. The conclusion is of the form

  $$T_L,\ T_R,\ A,\ \mu\mathtt{t}_1.S'_1 \circ \mathtt{end} \vdash \mu\mathtt{t}_1.S'_1$$

  with premise $A\ \{\mu\mathtt{t}_1.S'_1\}$ which gives the thesis.

E Proof of Algebraic and Scoping Properties

Definition 14 (Substituting for $\mathtt{end}$). *Given two protocols S and S' then $S[S'/\mathtt{end}]$ is defined:*

$$\begin{aligned}
(p.S)[S'/\mathtt{end}] &= p.S[S'/\mathtt{end}]\\
(+\{\mathrm{l_i} : S_i\}_{i\in I})[S'/\mathtt{end}] &= +\{\mathrm{l_i} : S_i[S'/\mathtt{end}]\}_{i\in I})\\
(\mathtt{assert}(n).S)[S'/\mathtt{end}] &= \mathtt{assert}(n).S[S'/\mathtt{end}]\\
(\mathtt{consume}(n).S)[S'/\mathtt{end}] &= \mathtt{consume}(n).S[S'/\mathtt{end}]\\
(\mathtt{require}(n).S)[S'/\mathtt{end}] &= \mathtt{require}(n).S[S'/\mathtt{end}]\\
(\mu\mathtt{t}.S)[S'/\mathtt{end}] &= (\mu\mathtt{t}.S[S'/\mathtt{end}])\\
\mathtt{t}[S'/\mathtt{end}] &= \mathtt{t}\\
\mathtt{end}[S'/\mathtt{end}] &= S'
\end{aligned}$$

Lemma E1 *Assume* $T_L, T_R,\ A,\ S_1 \circ S_2 \vdash S$ *and* $\mathtt{t} \in \mathit{fn}(S_1)$*, then* $\mathtt{t} \in \mathit{fn}(S_2)$ *and* $\mathtt{t} \in \mathsf{fv}(S)$*.*

Proof. The proof is by induction on the derivation of S, proceeding by case analysis on the last rule used. Base case [end] holds since $\mathtt{t} \notin \mathit{fn}(S_1)$. Base case [call] yields immediately $\mathtt{t} \in \mathit{fn}(S_2)$ and $\mathtt{t} \in \mathsf{fv}(S)$. Base case [rec3] holds since $\mathit{fn}(S_1) = \emptyset$ (by condition in the premise). All other cases hold directly by induction.

Proposition 4 *If* $T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash S$ *then* $\mathsf{fv}(S_1) \cup \mathsf{fv}(S_2) = \mathsf{fv}(S)$*.*

Proof. By induction on $T_L,\ T_R,\ R,\ S_1 \circ S_2 \vdash S$:

- (act)
$$\frac{T_L,\ T_R,\ A,\ S_1' \circ S_2 \vdash S}{T_L,\ T_R,\ A,\ p.S_1' \circ S_2 \vdash p.S}$$
By induction and then that $\mathsf{fv}(p.S) = \mathsf{fv}(S)$.
- (sym)
$$\frac{T_L,\ T_R,\ A,\ S_2 \circ S_1 \vdash S}{T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash S}$$
By induction and commutativity of $\cup$ on sets.
- (require)
$$\frac{T_L,\ T_R,\ A \cup \{n\},\ S_1' \circ S_2 \vdash S}{T_L,\ T_R,\ A \cup \{n\},\ \mathtt{require}(n).S_1' \circ S_2 \vdash \mathtt{require}(n).S}$$
By induction and then $\mathsf{fv}(\mathtt{require}(n).S) = \mathsf{fv}(S)$ (recall free variables are with respect to recursion variables rather than assertion names).
- (consume)
$$\frac{T_L,\ T_R,\ A \setminus \{n\},\ S_1' \circ S_2 \vdash S \qquad n \notin A}{T_L,\ T_R,\ A,\ \mathtt{consume}(n).S_1' \circ S_2 \vdash \mathtt{consume}(n).S}$$
By induction and then $\mathsf{fv}(\mathtt{consume}(n).S) = \mathsf{fv}(S)$.
- (assert)
$$\frac{T_L,\ T_R,\ A \cup \{n\},\ S_1' \circ S_2 \vdash S}{T_L,\ T_R,\ A,\ \mathtt{assert}(n).S_1' \circ S_2 \vdash \mathtt{assert}(n).S}$$
By induction and then $\mathsf{fv}(\mathtt{assert}(n).S) = \mathsf{fv}(S)$.
- (bra)
$$\frac{\forall i \in I \quad T_L,\ T_R,\ A,\ S_i \circ S_2 \vdash S_i'}{T_L,\ T_R,\ A,\ +\{l_i : S_i\}_{i \in I} \circ S_2 \vdash +\{l_i : S_i'\}_{i \in I}}$$
By induction we have that $\mathsf{fv}(S_i) \cup \mathsf{fv}(S_2) \supseteq \mathsf{fv}(S_i')$ then since $\bigcup_{i \in I} \mathsf{fv}(S_i') = \mathsf{fv}(+\{l_i : S_i'\}_{i \in I})$ and $\bigcup_{i \in I} \mathsf{fv}(S_i) = \mathsf{fv}(+\{l_i : S_i\}_{i \in I})$ we get that $\mathsf{fv}(+\{l_i : S_i'\}_{i \in I}) \cup \mathsf{fv}(S_2) \supseteq \mathsf{fv}(+\{l_i : S_i'\}_{i \in I})$.

- (rec1)

$$\frac{\mathcal{T}_L \cup \{\mathtt{t}_1\}, \mathcal{T}_R,\ A,\ S_1 \circ \mu\mathtt{t}_2.S_2 \vdash S \qquad A\ \{\mu\mathtt{t}_1.S\} \qquad \mathtt{Top}(S_1) = \emptyset}{\mathcal{T}_L, \mathcal{T}_R,\ A,\ \mu\mathtt{t}_1.S_1 \circ \mu\mathtt{t}_2.S_2 \vdash \mu\mathtt{t}_1.S}$$

  By induction $\mathsf{fv}(S_1) \cup \mathsf{fv}(S_2) = \mathsf{fv}(S)$. Since $\mathsf{fv}(\mu\mathtt{t}_1.S_1) = \mathsf{fv}(S_1) \setminus \{\mathtt{t}_1\}$ and $\mathsf{fv}(\mu\mathtt{t}_1.S) = \mathsf{fv}(S) \setminus \{\mathtt{t}_1\}$ then $\mathsf{fv}(\mu\mathtt{t}_1.S_1) \cup \mathsf{fv}(S_2) = \mathsf{fv}(\mu\mathtt{t}_1.S)$ as desired.
- (rec2)

$$\frac{\mathcal{T}_L, \mathcal{T}_R,\ A,\ S_1[\mathtt{t}/\mathtt{t}_1] \circ S_2 \vdash S \qquad \mathtt{t} \in \mathsf{dom}(\mathcal{T}_R) \qquad \mathtt{Top}(S_1) = \emptyset}{\mathcal{T}_L, \mathcal{T}_R,\ A,\ \mu\mathtt{t}_1.S_1 \circ S_2 \vdash S}$$

  By induction we have that $\mathsf{fv}(S_1[\mathtt{t}/\mathtt{t}_1]) \cup \mathsf{fv}(S_2) = \mathsf{fv}(S)$. We have two cases:
  - $\mathtt{t}_1 \notin \mathsf{fv}(S_1)$ then $\mathsf{fv}(\mu\mathtt{t}_1.S_1) = \mathsf{fv}(S_1[\mathtt{t}/\mathtt{t}_1])$ hence $\mathsf{fv}(\mu\mathtt{t}_1.S_1) \cup \mathsf{fv}(S_2) = \mathsf{fv}(S)$, as required.
  - $\mathtt{t}_1 \in \mathsf{fv}(S_1)$ then $\mathtt{t} \in \mathsf{fv}(S_1[\mathtt{t}/\mathtt{t}_1])$. In this case $\mathsf{fv}(\mu\mathtt{t}_1.S_1) = \mathsf{fv}(S_1[\mathtt{t}/\mathtt{t}_1]) \setminus \{\mathtt{t}_1\} \cup \{\mathtt{t}\}$. By lemma E1 $\mathtt{t} \in \mathsf{fv}(S_2)$ hence the thesis also holds for the consequence of [rec2]: $\mathsf{fv}(\mu\mathtt{t}_1.S_1) \cup \mathsf{fv}(S_2) = \mathsf{fv}(S)$, as required.
- (rec3)

$$\frac{A\ \{\mu\mathtt{t}.S\} \qquad \mathsf{fv}(\mu\mathtt{t}.S) = \emptyset}{\mathcal{T}_L, \mathcal{T}_R,\ A,\ \mu\mathtt{t}.S \circ \mathtt{end} \vdash \mu\mathtt{t}.S}$$

  This is a base case and holds since $\mathsf{fv}(\mathtt{end}) = \emptyset$ and trivially $\mathsf{fv}(\mu\mathtt{t}.S) \cup \emptyset = \mathsf{fv}(\mu\mathtt{t}.S)$.
- (call)

$$\frac{\mathtt{t} \in \mathcal{T}_L \vee \mathtt{t} \in \mathcal{T}_R}{\mathcal{T}_L, \mathcal{T}_R,\ A,\ \mathtt{t} \circ \mathtt{t} \vdash \mathtt{t}}$$

  Thus $\mathsf{fv}(\mathtt{t}) \cup \mathsf{fv}(\mathtt{t}) = \mathsf{fv}(\mathtt{t})$ trivially.
- (end)

$$\frac{-}{T_L,\ T_R,\ A,\ \mathtt{end} \circ \mathtt{end} \vdash \mathtt{end}}$$

  Trivial since $\mathsf{fv}(\mathtt{end}) = \emptyset$.

Corollary 2 (Composition preserves closedness) *For all A, S and closed protocols S_1, S_2, if $T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash S$ then S is a closed protocol.*

Proof. Simple corollary of proposition 4 since $\emptyset = \mathsf{fv}(S)$.

Proposition 5 (Interleaving composition has left- and right-units)

$$A\ \{S\} \wedge \mathsf{fv}(S) = \emptyset \implies T_L,\ T_R,\ A,\ S \circ \mathtt{end} \vdash S \ \wedge\ T_L,\ T_R,\ A,\ \mathtt{end} \circ S \vdash S$$

Appendix E details the proofs of the above results.

Proof. We split the proposition into two parts. First proving the right unit, then the left unit.

For the right unit, we proceed by induction on the derivation of $A\ \{S\}$:

- $S = p.S'$

$$\frac{A\ \{S'\}\ A'}{A\ \{p.S'\}\ A'}$$

Then by induction on S' we have that $T_L, T_R, A, S' \circ \texttt{end} \vdash S'$ and thus by (act) we get $T_L, T_R, A, p.S' \circ \texttt{end} \vdash p.S'$

- $S = +\{\mathrm{l}_i : S_i\}_{i \in I}$

$$\frac{\forall i \in I.\ A\ \{S_i\}\ A_i}{A\ \{+\{\mathrm{l}_i : S_i\}_{i\in I}\}\ \bigcap_{i\in I} A_i}$$

By induction on the premise for each S_i we get that $T_L, T_R, A, S_i \circ \texttt{end} \vdash S_i$. Applying all these as the premises of (bra), we get that:

$$T_L, T_R, A, +\{l_i : S_i\}_{i\in I} \circ \texttt{end} \vdash +\{l_i : S_i\}_{i\in I}$$

Satisfying the goal.

- $S = \texttt{require}(n).S'$

$$\frac{A' \cup \{n\}\ \{S'\}\ A''}{A' \cup \{n\}\ \{\texttt{require}(n).S'\}\ A''}$$

thus $A = A' \cup \{n\}$
By induction on the premise with S' then we have $T_L, T_R, A' \cup \{n\}, S' \circ \texttt{end} \vdash S'$.
Applying this to (require) for interleaving composition then gives us:

$$T_L, T_R, A' \cup \{n\}, \texttt{require}(n).S' \circ \texttt{end} \vdash \texttt{require}(n).S'$$

Satisfying the goal.

- $S = \texttt{consume}(n).S'$

$$\frac{A \setminus \{n\}\ \{S'\}\ A' \qquad n \in A}{A\ \{\texttt{consume}(n).S'\}\ A'}$$

By induction on the premise with S' then we have $T_L, T_R, A \setminus \{n\}, S' \circ \texttt{end} \vdash S'$.
Applying this to (consume) for interleaving composition (with the side condition of $n \in A$ from the well-assertedness rule) then gives us:

$$T_L, T_R, A, \texttt{consume}(n).S' \circ \texttt{end} \vdash \texttt{consume}(n).S'$$

Satisfying the goal.

- $S = \texttt{assert}(n).S'$

$$\frac{A \cup \{n\}\ \{S'\}\ A'}{A\ \{\texttt{assert}(n).S'\}\ A'}$$

By induction on the premise with S' then we have $T_L, T_R, A \cup \{n\}, S' \circ \texttt{end} \vdash S'$.
Applying this to (assert) for interleaving composition then gives us:

$$T_L, T_R, A, \texttt{assert}(n).S' \circ \texttt{end} \vdash \texttt{assert}(n).S'$$

Satisfying the goal.

- $S = \mu\mathtt{t}.S'$

$$\frac{A\ \{S'\}\ A \cup A'}{A\ \{\mu\mathtt{t}.S'\}\ A \cup A'}$$

  If $\mathsf{fv}(\mu\mathtt{t}.S) = \emptyset$ then we apply (rec3) and obtain the thesis. If $\mathsf{fv}(\mu\mathtt{t}.S) \neq \emptyset$ the hypothesis does not hold hence done.

- $S = \mathtt{end}$

$$\frac{-}{A\ \{\mathtt{end}\}\ A}$$

  We can then conclude with our goal via the (end) rule of interleaving composition: $T_L,\ T_R,\ A,\ \mathtt{end} \circ \mathtt{end} \vdash \mathtt{end}$.

- $S = \mathtt{t}$

$$\frac{-}{A\ \{\mathtt{t}\}\ A}$$

  In this case $\mathsf{fv}(S) = \{\mathtt{t}\} \neq \emptyset$ which contradicts the hypothesis hence done.

Thus we have proved the right unit property of interleaving composition: that for all S we have $T_L,\ T_R,\ A,\ S \circ \mathtt{end} \vdash S$.

To prove the left unit property we can then compose the above result with the (sym) rule of interleaving composition:

$$\frac{T_L,\ T_R,\ A,\ S \circ \mathtt{end} \vdash S}{T_L,\ T_R,\ A,\ \mathtt{end} \circ S \vdash S}\ [\text{sym}]$$

giving the left-unit property. □

F Proof of behaviour preservation (Theorem 1)

We recall the theorem for convenience.

Theorem 1 (Behaviour preservation of compositions - closed).

$$\emptyset,\ \emptyset,\ A,\ S_1 \circ S_2 \vdash S \quad \Rightarrow \quad (A, S) \lesssim (A, S_1 \,\|\, S_2)$$

Proof. We use stratification of similarity, along the lines of [30](Definition 2.2.10). Consider the relation $R = R1 \cup R2 \cup R3 \cup R4$ where

$$\begin{aligned}
R1 &= \{(S, S_1 \,\|\, S_2) \mid \emptyset,\ \emptyset,\ A,\ S_1 \circ S_2 \vdash S\}\\
R2 &= \{(S'[\mu\mathtt{t}.S/\mathtt{t}], S_1'[\mu\mathtt{t}.S_1/\mathtt{t}] \,\|\, S_2) | S \lesssim S_1 \wedge S' \lesssim S_1\ \ \wedge S' \neq \mathtt{t}\}\\
R3 &= \{(S', S_1'[\mu\mathtt{t}.S_1/\mathtt{t}] \,\|\, S_2)\}\\
R4 &= \{(S, S \,\|\, S_2)\}
\end{aligned}$$

First note that:

- $R1$ is used to capture the initial scenario of two *closed* types producing an interleaving composition

- $R2$ is used to handle the scenario where two recursive processes in $R1$ (one component and the composition) have a transition involving unfolding: $\emptyset, \emptyset, A, \mu\mathtt{t}.S_1 \circ S_2 \vdash \mu\mathtt{t}.S$ with $(A, \mu\mathtt{t}.S_1) \xrightarrow{\ell} (A', S_1'[\mu\mathtt{t}.S_1/\mathtt{t}])$ and $(A, \mu\mathtt{t}.S) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}.S/\mathtt{t}])$. $R2$ is a simulation by Lemma F7.
- $R3$ is used to handle the scenario where one recursive processes in $R1$ (one component) has a transition involving unfolding: $\emptyset, \emptyset, A, \mu\mathtt{t}.S_1 \circ S_2 \vdash S$ with $(A, \mu\mathtt{t}.S_1) \xrightarrow{\ell} (A', S_1'[\mu\mathtt{t}.S_1/\mathtt{t}])$ and $(A, S) \xrightarrow{\ell} (A', S')$. $R3$ is a simulation by Lemma F8 .
- $R4$ is a simulation as it is a special case of $\underline{R4}$ in Lemma F9 where $S = S_1$, and models the case of weak branching.

We show that is two processes are in relation $R1$, upon transition they will 'stay' in $R1$ or go to states related by $R2$, $R3$, or $R4$, which are simulations. First assume that the two processes considered are in $R1$.

Assume $(A, S) \xrightarrow{\ell} (A', S')$, we proceed by case analysis on the structure of S.

Case $\ell \notin \{\oplus l, \& l\}$ By hypothesis $(S, S_1 \parallel S_2) \in R_1$. By lemma 6 either S_1 or S_2 can move. Assume first that S_1 moves: $(A, S_1) \xrightarrow{\ell} (A', S_1')$ and $\emptyset, \emptyset, A', S_1' \circ S_2 \vdash S'$ with S_1' up to unfolding. If there is no unfolding then immediately $(S', S_1' \parallel S_2) \in R_1$. If there is unfolding in both S_1' and S then $(S', S_1' \parallel S_2) \in R_2$, if there is unfolding only in S_1 then $(S', S_1' \parallel S_2) \in R_3$. If S_2 moves the case is symmetric (as the previous case using symmetry in composition rules and in transition rules of protocol ensembles).

Case $\ell \in \{\oplus l, \& l\}$ By Lemma 6 (point 3) we have two cases. If the transition is matched by a transition of either S_1 or S_2 that preserves the composition derivation this case is identical to the case for $\ell \notin \{\oplus l, \& l\}$). If, instead, $(A, S) \xrightarrow{\ell} (A', \overline{S}[\mu\mathtt{t}.S/\mathtt{t}])$ and $(A, S_i) \xrightarrow{\ell} (A', \overline{S}[\mu\mathtt{t}.S_i/\mathtt{t}])$ with $i \in \{1, 2\}$ in this case the protocols are in relation $R4$.

F.1 Behaviour preservation - auxiliary definitions and lemmas

Lemma F1 *If* $(A, S_1) \xrightarrow{\ell} (A', S_1')$ *and* $\mathtt{t} \notin fn(S_1)$ *then* $\mathtt{t} \notin fn(S_1')$.

Proof sketch. The proof is by induction observing that no reduction rule adds free names.

Lemma F2 $(A, S_1[\mathtt{t}/\mathtt{t}_1]) \xrightarrow{\ell} (A', S_1') \wedge \mathtt{t} \notin fn(S_1) \implies (A, S_1) \xrightarrow{\ell} (A', S_1'[\mathtt{t}_1/\mathtt{t}])$.

Proof sketch. The proof is by induction on the proof of $\xrightarrow{\ell}$. All cases are base cases (trivial) except $\langle\mathtt{rec}\rangle$. For $\langle\mathtt{rec}\rangle$ we consider two cases:

1. $S_1 = \mu\mathtt{t}_1.S_1$. In this case $\mu\mathtt{t}_1.S_1[\mathtt{t}/\mathtt{t}_1] = \mu\mathtt{t}_1.S_1$ and by $\langle\mathtt{rec}\rangle$

$$\frac{(A, S_1) \xrightarrow{\ell} (A', S_1')}{\mu\mathtt{t}_1.S_1 \xrightarrow{\ell} (A', S_1'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1])}$$

The thesis is by observing that $S'_1[\mu\mathtt{t}_1.S_1/\mathtt{t}_1] = S'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1][\mathtt{t}_1/\mathtt{t}]$ since

- $\mathtt{t} \notin \mathit{fn}(\mu\mathtt{t}_1.S_1)$ by hypothesis;
- $\mathtt{t} \notin \mathit{fn}(S'_1[\mu\mathtt{t}_1.S_1/\mathtt{t}_1])$ by Lemma F1.

2. $S_1 = \mu\mathtt{t}_2.S_1$. By hypothesis (and rule $\langle\mathtt{rec}\rangle$)

$$\frac{(A, S_1[\mathtt{t}/\mathtt{t}_1]) \xrightarrow{\ell} (A', S'_1)}{\mu\mathtt{t}_2.S_1[\mathtt{t}/\mathtt{t}_1] \xrightarrow{\ell} (A', S'_1[\mu\mathtt{t}_2.S_1[\mathtt{t}/\mathtt{t}_1]/\mathtt{t}_2])} \tag{32}$$

By induction using the premise of eq. (32)

$$(A, S_1) \xrightarrow{\ell} (A', S'_1[\mathtt{t}_1/\mathtt{t}])$$

The above used as premise of $\langle\mathtt{rec}\rangle$ gives

$$(A, \mu\mathtt{t}_2.S_1) \xrightarrow{\ell} (A', S'_1[\mathtt{t}_1/\mathtt{t}][\mu\mathtt{t}_2.S_1/\mathtt{t}_2]) \tag{33}$$

Looking at eq. (32), we need to prove that $S'_1[\mu\mathtt{t}_2.S_1[\mathtt{t}/\mathtt{t}_1]/\mathtt{t}_2][\mathtt{t}_1/\mathtt{t}]$ is equal to $S'_1[\mathtt{t}_1/\mathtt{t}][\mu\mathtt{t}_2.S_1/\mathtt{t}_2]$ from eq. (33). We show this below.

$$\begin{aligned} & S'_1[\mu\mathtt{t}_2.S_1[\mathtt{t}/\mathtt{t}_1]/\mathtt{t}_2][\mathtt{t}_1/\mathtt{t}] \\ &\quad = S'_1[\mathtt{t}_1/\mathtt{t}][\mu\mathtt{t}_2.S_1[\mathtt{t}/\mathtt{t}_1][\mathtt{t}_1/\mathtt{t}]/\mathtt{t}_2] && \text{(distribution of substitution)} \\ &\quad = S'_1[\mathtt{t}_1\mathtt{t}][\mu\mathtt{t}_2.S_1/\mathtt{t}_2] && \text{(since } \mathtt{t} \notin \mathit{fn}(S_1) \text{ then } S'_1[\mathtt{t}/\mathtt{t}_1][\mathtt{t}_1/\mathtt{t}] = S'_1\text{)} \end{aligned}$$

As desired.

Lemma F3

$$A\{S\} \wedge A'\{S'\}A \Rightarrow A'\{S'[S/\mathtt{t}_1]\}$$

Proof. By induction on the syntax of S'

Base cases

- If $S' = \mathtt{t}$ then $A'\{\mathtt{t}\}A$. If $\mathtt{t} \neq \mathtt{t}_1$ then thesis is by hypothesis. If $\mathtt{t} = \mathtt{t}_1$ then $A'\{\mathtt{t}_1\}A$ and $A' = A$ by [call] so $A\{\mathtt{t}_1\}A$ hence hypothesis $A\{S_1\}$ yields the thesis.
- If $S' = \mathtt{end}$ then $A'\{\mathtt{end}\}A$ and the thesis is the hypothesis as $\mathtt{t}_1 \notin \mathit{fn}(\mathtt{end})$.

Inductive cases

- If $S' = p.S''$ then by well-formedness rule [act]

$$\frac{A'\{S''\}A}{A'\{p.S''\}A}$$

By induction $A'\{S''[S/\mathtt{t}_1]\}A$ which, by [act] gives $A'\{p.S''[S/\mathtt{t}_1]\}A$ as desired.

- If $S' = \mathtt{consume}(n).S''$ then by well-formedness rule [consume]

  $$\frac{A' \setminus \{n\}\{S''\}A}{A'\{\mathtt{consume}(n).S''\}A}$$

  By induction $A' \setminus \{n\}\{S''[S/\mathtt{t}_1]\}A$ which by [consume] gives

  $$A'\{\mathtt{consume}(n).S''[S/\mathtt{t}_1]\}A$$

  as desired.
- The cases for assert, assume, and branching are similar to consume.
- If $S' = \mu\mathtt{t}.S''$ then by well-formedness rule [rec]

  $$\frac{A'S''A \cup A''}{A'\mu\mathtt{t}.S''A \cup A''}$$

  We have two cases: if $\mathtt{t} = \mathtt{t}_1$ then $A'\{\mu\mathtt{t}_1.S''[S/\mathtt{t}_1] = \mu\mathtt{t}_1.S''\}A \cup A''$ hence done; if $\mathtt{t} \neq \mathtt{t}_1$ then by induction $A'S''[S/\mathtt{t}_1]A \cup A''$ which used as premise of [rec] gives $A'\{\mu\mathtt{t}.S''[S/\mathtt{t}_1]\}A \cup A''$.

Lemma F4 (Environment Unfolding)

$$\underline{T}_L, \underline{T}_R, \underline{A}, \mu\mathtt{t}_1.\underline{S}_1 \circ \underline{S}_2 \vdash \mu\mathtt{t}_1.\underline{S} \tag{34}$$

and

$$\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup T_R, A, S_1 \circ S_2 \vdash S \tag{35}$$

and

$$A\{S\}\underline{A} \tag{36}$$

imply

$$\underline{T}_L \cup \mathit{fn}(S_1) \cap T_L, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R, A, S_1[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash S[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

Proof. This lemma holds for all variants of composition: $\vdash_s$, $\vdash_w$, $\vdash_c$, and $\vdash_{wc}$ (recall that notation $\vdash$ is used to refer to any of the aforementioned composition judgments). The proof focusses on proving $\vdash_{wc}$ which is the most general case; the other cases can be obtained by simply omitting the inductive cases for rules not used by that kind of composition (e.g., for $\vdash_s$ omit the [wbra] and [cbra] case).

The proof by induction on the derivation of S by case analysis on the last rule used.

[rec1] - $S_1 = \mu\mathtt{t}.S'_1$ *and* $S = \mu\mathtt{t}.S'$. By hypothesis (showing the last rule application by [rec1])

$$\frac{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1, \mathtt{t}\}, \emptyset, A, S'_1 \circ S_2 \vdash S' \quad A\{\mu\mathtt{t}.S'\}}{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \emptyset, A, \mu\mathtt{t}.S'_1 \circ S_2 \vdash \mu\mathtt{t}.S'} \tag{37}$$

By induction, considering the premise of eq. (37)

$$\underline{T}_L \cup \mathit{fn}(S_1') \cap T_L \cup \{\mathtt{t}\}, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A,\ S_1'[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

Since $T_R \subseteq T_R'$ and $T_R' = \emptyset$ the above is equivalent to

$$\underline{T}_L \cup \mathit{fn}(S_1') \cap T_L \cup \{\mathtt{t}\}, \emptyset,\ A,\ S_1'[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1]\ \circ\ S_2[\underline{S}_2/\mathtt{t}_1] \vdash S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1] \quad (38)$$

By hypothesis eq. (36) $A\{\mu\mathtt{t}.S'\}\underline{A}$ and by hypothesis eq. (34) it follows $\underline{A}\{S\}$ hence by lemma F3

$$A\{\mu\mathtt{t}.S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]\} \quad (39)$$

Then I can use eq. (38) and eq. (39) as premise of [rec1] obtaining the thesis

$$\underline{T}_L \cup \mathit{fn}(S_1) \cap T_L, \emptyset,\ A,\ \mu\mathtt{t}.S_1'[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1]\ \circ\ S_2[\underline{S}_2/\mathtt{t}_1] \vdash \mu\mathtt{t}.S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

as required.

[rec2] - $S_1 = \mu\mathtt{t}.S_1'$ By hypothesis (showing the last rule application by [rec2])

$$\frac{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup T_R \cup \{\mathtt{t}_2\},\ A,\ S_1'[\mathtt{t}_2/\mathtt{t}]\ \circ\ S_2 \vdash S}{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup T_R \cup \{\mathtt{t}_2\},\ A,\ \mu\mathtt{t}.S_1'\ \circ\ S_2 \vdash S} \quad (40)$$

By induction

$$\underline{T}_L \cup \mathit{fn}(S_1'[\mathtt{t}_2/\mathtt{t}]) \cap T_L, \underline{T}_R \cup \mathit{fn}(S_2') \cap T_R \cup \mathtt{t}_2,\ A,\ S_1'[\mathtt{t}_2/\mathtt{t}][\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1 \vdash S[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

By applying the above as a premise of [rec2] we obtain

$$\underline{T}_L \cup \mathit{fn}(S_1'[\mathtt{t}_2/\mathtt{t}]) \cap T_L, \underline{T}_R \cup \mathit{fn}(S_2') \cap T_R \cup \mathtt{t}_2,\ A,\ \mu\mathtt{t}.S_1'[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1 \vdash S[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1] \quad (41)$$

Observe the following:

$$\mathit{fn}(S_1'[\mathtt{t}_2/\mathtt{t}]) = \mathit{fn}(S_1') \qquad \text{since } \mathtt{t} \notin T_L \text{by bound names convention.} \quad (42)$$

$$T_L \cup \mathit{fn}(S_1) = T_L \qquad \text{since } \mathit{fn}(S_1) \in T_L \text{ by hypothesis (1) and Lemma B} \quad (43)$$

$$\mathit{fn}(S_1') \cap T_L = \mathit{fn}(S_1') \setminus \{\mathtt{t}_1\} \cap T_L \qquad \text{since } \mathtt{t}_1 \notin T_L \quad (44)$$

$$\mathit{fn}(S_1'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1]) = \mathit{fn}(S_1') \setminus \{\mathtt{t}_1\} \cup \mathit{fn}(S_1) \quad (45)$$

So

$$\begin{array}{rcll}
T_L \cup \mathit{fn}(S_1'[\mathtt{t}_2/\mathtt{t}]) \cap T_L & = & T_L \cup \mathit{fn}(S_1') \cap T_L & \text{by } \mathit{eq.}\ (42) \\
 & = & T_L \cup \mathit{fn}(S_1) \cup \mathit{fn}(S_1') \cap T_L & \text{by } \mathit{eq.}\ (43) \\
 & = & T_L \cup \mathit{fn}(S_1) \cup \mathit{fn}(S_1') \setminus \{\mathtt{t}_1\} \cap T_L & \text{by } \mathit{eq.}\ (44) \\
 & = & T_L \cup \mathit{fn}(S_1) \cup \mathit{fn}(S_1?) \setminus \{ & \\
rv_1\} \cap T_L & \text{by } \mathit{eq.}\ (45) & & \\
 & = & T_L \cup \mathit{fn}(S_1'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1]) \cap T_L &
\end{array}$$

By substituting $T_L \cup \mathit{fn}(S_1'[\mathtt{t}_2/\mathtt{t}]) \cap T_L$ with $T_L \cup \mathit{fn}(S_1'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1]) \cap T_L$ in eq. (41) we have the thesis.

[call] If $S'_1 = \mathtt{t}$ then by hypothesis $\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A,\ \mathtt{t} \circ \mathtt{t} \vdash \mathtt{t}$ The thesis is immediate as $\mathtt{t}[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] = \mathtt{t}[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1] = \mathtt{t}[S_1/\mathtt{t}_1] = \mathtt{t}$.

If $S'_1 = \mathtt{t}_1$ then by hypothesis $\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A,\ \mathtt{t}_1 \circ \mathtt{t}_1 \vdash \mathtt{t}_1$ The thesis is equivalent to hypothesis eq. (34) observing that

$$\underline{T}_L = \underline{T}_L \cup \mathit{fn}(S'_1) \cap T_L = \underline{T}_L \cup (\emptyset \cap T_L)$$

$$\underline{T}_R = \underline{T}_R \cup (\mathit{fn}(S_2) \cap T_R) = T_R \cup (\emptyset \cap T_R)$$

as desired.

[consume] - $S_1 = \mathtt{consume}(n).S'_1$ By hypothesis

$$\frac{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A,\ S'_1 \circ S_2 \vdash S}{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A \cup \{n\},\ \mathtt{consume}(n).S'_1 \circ S_2 \vdash S}$$

By induction, considering the premise of the derivation above:

$$\underline{T}_L \cup \mathit{fn}(S'_1) \cup T_L, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A,\ S'_1[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash S[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

by using the above as a premise of [consume] we obtain

$$\underline{T}_L \cup \mathit{fn}(\mathtt{consume}(n).S'_1) \cup T_L, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A \cup n,\ \mathtt{consume}(n).S'_1[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash S[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

which is the thesis, observing that $\mathit{fn}(\mathtt{consume}(n).S'_1) = \mathit{fn}(S'_1)$ as desired.

[wbra] - $S_1 = +\{\mathrm{l_i} : S_i\}_{i \in I}$ By hypothesis

$$\frac{\begin{array}{ll} \forall i \in I_A & \underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A,\ S_i \circ S_2 \vdash S'_i \\ \forall i \in I_B & A\{S_i\} \wedge \underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A,\ S_i \circ S_2 \not\vdash S'_i \end{array}}{\underline{T}_L \cup T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \ T_R,\ A,\ +\{\mathrm{l_i} : S_i\}_{i \in I} \circ S_2 \vdash +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup \{\mathrm{l}_i : S_i\}_{i \in I_B}} \tag{46}$$

By induction:

$$\forall i \in I_A \quad \underline{T}_L \cup \mathit{fn}(S_i) \cap T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A,\ S_i[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash S'_i[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1] \tag{47}$$

and by second premise of eq. (46)

$$\forall i \in I_B \quad \underline{T}_L \cup \mathit{fn}(S_i) \cap T_L \cup \{\mathtt{t}_1\}, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A,\ S_i[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \not\vdash \tag{48}$$

By applying eq. (47) and eq. (48) as premise of [wbra] we obtain

$$\begin{array}{l} \underline{T}_L \cup \mathit{fn}(+\{\mathrm{l_i} : S_i\}_{i \in I}) \cup T_L, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A,\ +\{\mathrm{l_i} : S_i[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1]\}_{i \in I} \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash \\ \qquad\qquad +\{\mathrm{l_i} : S'_i[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]\}_{i \in I_A} \cup \{\mathrm{l}_i : S_i[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]\}_{i \in I_B} \end{array}$$

which by definition of substitution is equivalent to

$$\begin{array}{l} \underline{T}_L \cup \mathit{fn}(+\{\mathrm{l_i} : S_i\}_{i \in I}) \cup T_L, \underline{T}_R \cup \mathit{fn}(S_2) \cap T_R,\ A,\ +\{\mathrm{l_i} : S_i\}_{i \in I}[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S_2[\underline{S}_2/\mathtt{t}_1] \vdash \\ \qquad\qquad (+\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup \{\mathrm{l}_i : S_i\}_{i \in I_B})[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1] \end{array}$$

which is the thesis, observing that $\mathit{fn}(+\{\mathrm{l_i} : S_i\}_{i \in I}) = \bigcup_{i \in I} \mathit{fn}(S_i)$ as desired.

[bra] - $S_1 = +\{\mathrm{l_i} : S_i\}_{i \in I}$ This case is a special case of [wbra] above with $I_B = \emptyset$.

F.2 On the relation $R1$

Definition 15 (Folding).

$$\begin{array}{l}
\mathtt{Fold}(p.S, \mathtt{t}) = p.\mathtt{Fold}(S, \mathtt{t}) \\
\mathtt{Fold}(\mathtt{assert}(n).S_1, \mathtt{t}) = \mathtt{assert}(n).\mathtt{Fold}(S, \mathtt{t}) \\
\mathtt{Fold}(\mathtt{consume}(n).S_1, \mathtt{t}) = \mathtt{consume}(n).\mathtt{Fold}(S, \mathtt{t}) \\
\mathtt{Fold}(\mathtt{require}(n).S_1, \mathtt{t}) = \mathtt{require}(n).\mathtt{Fold}(S, \mathtt{t}) \\
\mathtt{Fold}(+\{\mathrm{li} : S_i\}_{i \in I}, \mathtt{t}) = +\{\mathrm{li} : \mathtt{Fold}(S_i, \mathtt{t})\}_{i \in I} \\
\mathtt{Fold}(\mu\mathtt{t}.S_1, \mathtt{t}) = \mathtt{t} \\
\mathtt{Fold}(\mu\mathtt{t}'.S_1, \mathtt{t}) = \mu\mathtt{t}'.\mathtt{Fold}(S_1, \mathtt{t}) \\
\mathtt{Fold}(\mathtt{end}, \mathtt{t}) = \mathtt{end} \\
\mathtt{Fold}(\mathtt{t}', \mathtt{t}) = \mathtt{t}'
\end{array}$$

Definition 16 (Top).

$$\mathtt{Top}(S) = \begin{cases} \mathtt{t} & \textit{if } S = \mu\mathtt{t}.S' \\ \emptyset & \textit{otherwise} \end{cases}$$

Lemma F5 *If* $T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash \mu\mathtt{t}.S$ *then* $\mathtt{Top}(S) = \emptyset$

Proof sketch. Can be proved by induction on the proof of $\mu\mathtt{t}.S$. Observe that the composition rules never concatenate recursions from the same protocol (by premise $\mathtt{Top}(S_1) = \emptyset$ in [rec1]) or from different protocols (by premise $\mathtt{Top}(S_1) = \emptyset$ in [rec2]).

Lemma F6 (Preservation - open protocols) *If* $T_L,\ T_R,\ A,\ S_1 \circ S_2 \vdash S$ *and* $(A, S) \xrightarrow{\ell} (A', S')$ *for some* ℓ, A', S' *then one of the following holds:*

1. $(A, S_1) \xrightarrow{\ell} (A', S_1')$ *and* $T_L,\ T_R,\ A',\ \mathtt{Fold}(S_1', \mathtt{Top}(S_1))@ \circ S_2 \vdash \mathtt{Fold}(S', \mathtt{Top}(S))$, *for some* @ *substitution of* $\mathtt{t}' \in \mathtt{Top}(S_1) \setminus \mathit{fn}(S')$ *with* $\mathtt{t} \in T_R$
2. $(A, S_2) \xrightarrow{\ell} (A', S_2')$ *and* $T_L,\ T_R,\ A',\ S_1 \circ \mathtt{Fold}(S_2', \mathtt{Top}(S_2))@ \vdash \mathtt{Fold}(S', \mathtt{Top}(S))$, *for some* @ *substitution of* $\mathtt{t}' \in \mathtt{Top}(S_2) \setminus \mathit{fn}(S')$ *with* $\mathtt{t} \in T_L$
3. $(A, S_i) \xrightarrow{+1} (A', S_i')$ *with* $i \in \{1, 2\}$ *and* $S' = \overline{S}[S/\mathtt{Top}(S)]$ *and* $S_i' = \overline{S}[S_i/\mathtt{Top}(S_i)]$ *for some* $\overline{S}$

Proof. This lemma holds for all variants of composition: $\vdash_s$, $\vdash_w$, $\vdash_c$, and $\vdash_{wc}$ (recall that notation $\vdash$ is used to refer to any of the aforementioned composition judgments). The proof focusses on proving $\vdash_{wc}$ which is the most general case; the other cases can be obtained by simply omitting the inductive cases for rules not used by that kind of composition (e.g., for $\vdash_s$ omit the [wbra] and [cbra] case).

By induction on the derivation of S by case analysis on last rule used.

[end] The hypothesis does not hold since $(A, \texttt{end}) \not\rightarrow$ hence done.

[call] The hypothesis does not hold since $(A, \texttt{t}) \not\rightarrow$ hence done.

[consume] - $S = \texttt{consume}(n).\underline{S}$ The top of the derivation is of the following form:

$$\frac{\mathcal{T}_L,\, T_R,\, A,\, \underline{S}_1 \circ S_2 \vdash \underline{S}}{\mathcal{T}_L,\, T_R,\, A \cup \{n\},\, \texttt{consume}(n).\underline{S}_1 \circ S_2 \vdash \texttt{consume}(n).\underline{S}} \tag{49}$$

By hypothesis

$$(A \cup \{n\}, \texttt{consume}(n).\underline{S}) \xrightarrow{\ell} (A, S')$$

Since the only transition rule applicable to $(A \cup \{n\}, \texttt{consume}(n).\underline{S})$ is $\langle\texttt{consume}\rangle$ then $\ell = \texttt{consume}(n)$ and $S' = \underline{S}$

$$(A \cup \{n\}, \texttt{consume}(n).\underline{S}) \xrightarrow{\texttt{consume}(n)} (A, \underline{S})$$

Similarly, by $\langle\texttt{consume}\rangle$

$$(A \cup \{n\}, \texttt{consume}(n).\underline{S}_1) \xrightarrow{\texttt{consume}(n)} (A, \underline{S}_1)$$

The thesis (item 1) follows immediately by the premise of (eq. (49)) observing that $\texttt{Top}(S_1) = \emptyset$ and $\texttt{Top}(S) = \emptyset$ and @ is the empty substitution.

Cases [pref], [assume], [assert] are similar to the case for [consume].

Cases [wbra] By hypothesis

$$\frac{\begin{array}{l} I_A \cap I_B = \emptyset \quad I_A \cup I_B = I \quad I_A \neq \emptyset \\ \forall i \in I_A \quad \mathcal{T}_L,\, T_R,\, A,\, S_i \circ S_2 \vdash S'_i \\ \forall i \in I_B \quad A\{S_i\} \quad \mathcal{T}_L,\, T_R,\, A,\, S_i \circ S_2 \not\vdash \end{array}}{T_L,\, T_R,\, A,\, +\{\mathrm{l_i} : S_i\}_{i \in I} \circ S_2 \vdash +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup +\{\mathrm{l_i} : S_i\}_{i \in I_B}} \tag{50}$$

$S = +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup +\{\mathrm{l_i} : S_i\}_{i \in I_B}$ can only move by $\langle\texttt{Branch}\rangle$ with $\ell = \mathrm{l}_j$ and either $j \in I_A$ or $j \in I_B$.

Case $j \in I_A$. $(A, +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup +\{\mathrm{l_i} : S_i\}_{i \in I_B}) \xrightarrow{+l_i} (A, S'_j)$. Similarly, by $\langle\texttt{Branch}\rangle$ on S_1

$$(A, +\{\mathrm{l_i} : S_i\}_{i \in I}) \xrightarrow{+l_j} (A, S_j)$$

The thesis hold (item 1) as it is the premise in (eq. (50)) for $i = j \in I_A$ observing that $\texttt{Top}(+\{\mathrm{l_i} : S'_i\}_{i \in I}) = \emptyset$ and $\texttt{Top}(+\{\mathrm{l_i} : S_i\}_{i \in I}) = \emptyset$ and @ is the empty substitution.

Case $j \in I_B$. $(A, +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup +\{\mathrm{l_i} : S_i\}_{i \in I_B}) \xrightarrow{+l_i} (A, S_j)$. Similarly, by $\langle\texttt{branch}\rangle$ on S_1

$$(A, +\{\mathrm{l_i} : S'_i\}_{i \in I}) \xrightarrow{+l_j} (A, S_j)$$

Thesis holds (item 3) with $\overline{S} = S_j$ since $\mathit{fn}(S_j) \setminus \mathit{fn}(S) = \mathit{fn}(S_j) \setminus \mathit{fn}(S_1) = \emptyset$.

Cases [bra] As the case [wbra] assuming $I_B = \emptyset$.

Cases [cbra] By hypothesis

$$\frac{\begin{array}{c}\forall i \in I\ J_i \neq \emptyset \quad \bigcup_{i\in I} J_i = J \\ \forall j \in J_i \quad T_L,\, T_R,\, A,\, S_i \circ S'_j \vdash S_{ij} \\ \forall j \in J \setminus J_i \quad T_L,\, T_R,\, A,\, S_i \circ S'_j \nvdash \end{array}}{T_L,\, T_R,\, A,\, +\{\mathrm{l_i} : S_i\}_{i\in I} \circ +'\{\mathrm{l'_j} : S'_j\}_{j\in J} \vdash +\{\mathrm{l_i} : +'\{\mathrm{l'_j} : S_{ij}\}_{i\in J_i}\}_{i\in I}} \tag{51}$$

$S = +\{\mathrm{l_i} : +'\{\mathrm{l'_j} : S_{ij}\}_{i\in J_i}\}_{i\in I}$ can only move by $\langle \texttt{Branch} \rangle$ with $\ell = \mathrm{l}_i$ as follows:

$$(A, +\{\mathrm{l_i} : +'\{\mathrm{l'_j} : S_{ij}\}_{i\in J_i}\}_{i\in I}) \xrightarrow{+l_i} (A, +'\{\mathrm{l'_j} : S_{ij}\}_{i\in J_i})$$

Similarly, by $\langle \texttt{Branch} \rangle$ on S_1

$$(A, +\{\mathrm{l_i} : S_i\}_{i\in I}) \xrightarrow{+l_i} (A, S_i)$$

The first premise in (eq. (50)) can be applied as axiom in the derivation below to obtain the thesis (item 1) and observing that $\texttt{Top}(+\{\mathrm{l_i} : S'_i\}_{i\in I}) = \emptyset$ and @ is the empty substitution:

$$\dfrac{\dfrac{\dfrac{T_L,\, T_R,\, A,\, S_i \circ S'_j \vdash S_{ij}}{T_R,\, T_L,\, A,\, S'_j \circ S_i \vdash S_{ij}}\,[\text{sym}]}{T_R,\, T_L,\, A,\, +'\{\mathrm{l'_j} : S'_j\}_{j\in J} \circ S_i \vdash +'\{\mathrm{l'_j} : S_{ij}\}_{i\in J_i}}\,[\text{bra}]}{T_L,\, T_R,\, A,\, S_i \circ +'\{\mathrm{l'_j} : S'_j\}_{j\in J} \vdash +'\{\mathrm{l'_j} : S_{ij}\}_{i\in J_i}}\,[\text{sym}]$$

Case [sym] The last rule applies is of the following form:

$$\frac{T_L,\, T_R,\, A,\, S_2 \circ S_1 \vdash S}{T_L,\, T_R,\, A,\, S_1 \circ S_2 \vdash S}$$

By hypothesis $(A, S) \xrightarrow{\ell} (A', S')$.

By induction one of the following holds:

1. if $(A, S_2) \xrightarrow{\ell} (A', S'_2)$ then $T_L,\, T_R,\, A',\, \texttt{Fold}(S'_2, \texttt{Top}(S_2))@ \circ S_1 \vdash \texttt{Fold}(S', \texttt{Top}(S))$ which yields the thesis when applied as a premise of [sym]. The case for
2. if $\ell \in \{\oplus l, \& l\}$ and $(A, S_1) \xrightarrow{\ell} (A', S'_1)$ and $S' = \overline{S}[S/\mathit{fn}(\overline{S}) \setminus \mathit{fn}(S)]$ and $S'_1 = \overline{S}[S_1/\mathit{fn}(\overline{S}) \setminus \mathit{fn}(S_1)]$ for some $\overline{S}$ then the thesis (item 3) holds after applying [sym].
3. if $\ell \in \{\oplus l, \& l\}$ and $(A, S_2) \xrightarrow{\ell} (A', S'_2)$ the case is similar to case (2).
4. if $(A, S_1) \xrightarrow{\ell} (A', S'_1)$ the case is similar to case (1).

Case [rec1] - $S = \mu\mathtt{t}_1.\underline{S}$ and $T_R = \emptyset$ By hypothesis

$$\frac{\mathcal{T}_L \cup \{\mathtt{t}_1\}, \emptyset,\ A,\ \underline{S}_1 \circ \mu\mathtt{t}_2.\underline{S}_2 \vdash \underline{S} \quad \mathtt{Top}(\underline{S}_1) = \emptyset}{\mathcal{T}_L, \emptyset,\ A,\ \mu\mathtt{t}_1.\underline{S}_1 \circ \mu\mathtt{t}_2.\underline{S}_2 \vdash \mu\mathtt{t}_1.\underline{S}} \tag{52}$$

and

$$\frac{(A, \underline{S}) \xrightarrow{\ell} (A', S')}{(A, \mu\mathtt{t}_1.\underline{S}) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1])} \tag{53}$$

By induction, considering the premise of (eq. (52)) we have one of the following three cases:

Case S_1 moves and composition is preserved. If $(A, \underline{S}_1) \xrightarrow{\ell} (A', S'_1)$ and

$$\mathcal{T}_L \cup \{\mathtt{t}_1\}, \emptyset,\ A',\ \mathtt{Fold}(S'_1, \mathtt{Top}(\underline{S}_1))@ \circ S_2 \vdash \mathtt{Fold}(S', \mathtt{Top}(\underline{S}))$$

then by premise of (eq. (52)) $\mathtt{Top}(\underline{S}_1) = \emptyset$ hence @ is empty substitution and we get

$$\mathcal{T}_L \cup \{\mathtt{t}_1\}, \emptyset,\ A',\ S'_1 \circ S_2 \vdash S' \tag{54}$$

By $\langle\mathtt{rec}\rangle$, $(A, \mu\mathtt{t}_1.S_1) \xrightarrow{\ell} (A', S'_1[\mu\mathtt{t}_1.S_1])$. The thesis to prove is therefore

$$\mathcal{T}_L, \emptyset,\ A',\ \mathtt{Fold}(S'_1[\mu\mathtt{t}_1.S_1/\mathtt{t}_1], \mathtt{Top}(\underline{S}_1))@ \circ S_2 \vdash \mathtt{Fold}(S'[\mu\mathtt{t}_1.S/\mathtt{t}_1], \mathtt{Top}(\underline{S}))$$

Observing that $\mathtt{Top}(S_1) = \mathtt{t}_1$, the derivation above is equivalent to

$$\mathcal{T}_L, \emptyset,\ A',\ \mathtt{Fold}(S'_1[\mu\mathtt{t}_1.S_1/\mathtt{t}_1], \mathtt{t}_1)@ \circ S_2 \vdash \mathtt{Fold}(S'[\mu\mathtt{t}_1.S/\mathtt{t}_1], \mathtt{t}_1)$$

that is

$$\mathcal{T}_L, \emptyset,\ A',\ S'_1@ \circ S_2 \vdash S'$$

Observing that @ is the empty substitution since in $\mathtt{Top}(S_1) = \mathtt{t}_1$ and $\mathtt{t}_1 \in \mathit{fn}(S')$ the above follows immediately by eq. (54).

Case S_2 moves and composition is preserved. If $(A, S_2) \xrightarrow{\ell} (A', S'_2)$ and

$$\mathcal{T}_L \cup \{\mathtt{t}_1\}, \emptyset,\ A',\ \underline{S}_1 \circ \mathtt{Fold}(S'_2, \mathtt{Top}(S_2))@ \vdash \mathtt{Fold}(S', \mathtt{Top}(\underline{S}))$$

then by Lemma F5, $\mathtt{Top}(S) = \emptyset$ and @ is empty. Therefore, the above is equivalent to

$$\mathcal{T}_L \cup \{\mathtt{t}_1\}, \emptyset,\ A',\ \underline{S}_1 \circ \mathtt{Fold}(S'_2, \mathtt{Top}(S_2)) \vdash S' \tag{55}$$

The thesis

$$\mathcal{T}_L, \emptyset,\ A',\ \mu\mathtt{t}_1.\underline{S}_1 \circ \mathtt{Fold}(S'_2, \mathtt{Top}(S_2))@ \vdash \mathtt{Fold}(S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1], \mathtt{Top}(S))$$

is equivalent to eq. (55) since: $\mathtt{Top}(S) = \mathtt{t}_1$, $\mathtt{Fold}(S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1], \mathtt{t}_1) = S'$ and @ is the empty substitution (as $\mathit{fn}(S') = \mathtt{t}_1$ which is not a name in S_2 as we assume bound names of S_1 and S_2 to be disjoint.

Case $\ell \in \{\oplus l, \& l\}$ and composition is not preserved. By induction either S_1 or S_2 makes a transition with label ℓ. We show the case in which S_1 moves, as the case in which $(A, \underline{S}_2) \xrightarrow{\ell} (A', S')$ is symmetric.

Assume by induction $(A, \underline{S}_1) \xrightarrow{\ell} (A', S')$. Then: (1) since $\mathtt{Top}(\underline{S}_1) = \emptyset$ then $\mathit{fn}(S') \setminus \mathit{fn}(\underline{S}_1) = \emptyset$, and (2) by Lemma F5 $\mathtt{Top}(\underline{S}_1) = \emptyset$ then $\overline{S} = S'$ and $\mathit{fn}(S') \setminus \mathit{fn}(\underline{S}) = \emptyset$. So, by $\langle\mathtt{rec}\rangle$

$$(A, \mu\mathtt{t}_1.S_1) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1] \qquad (A, \mu\mathtt{t}_1.S) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}_1.S/\mathtt{t}_1])$$

with $\mathit{fn}(S') \setminus \mathit{fn}(\mu\mathtt{t}_1.\underline{S}) = \mathtt{t}_1$ and $\mathit{fn}(S'_1) \setminus \mathit{fn}(\mu\mathtt{t}_1.\underline{S}_1) = \mathtt{t}_1$ hence the thesis.

Case [rec2] - $S = \mu\mathtt{t}_1.\underline{S}$ and $T_R = \underline{T}_R \cup \{\mathtt{t}\}$ By hypothesis

$$\frac{T_L, \underline{T}_R \cup \{\mathtt{t}\},\ A,\ \underline{S}_1[\mathtt{t}/\mathtt{t}_1] \circ S_2 \vdash S \quad \mathtt{Top}(\underline{S}_1) = \emptyset}{T_L, \underline{T}_R \cup \{\mathtt{t}\},\ A,\ \mu\mathtt{t}_1.\underline{S}_1 \circ S_2 \vdash S} \tag{56}$$

and

$$\frac{(A, \underline{S}) \xrightarrow{\ell} (A', S')}{(A, \mu\mathtt{t}_1.\underline{S}) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1])} \tag{57}$$

We apply induction to the premise of eq. (56).

By induction considering the premise of eq. (56) we have one of the following cases:

1. First, assume

$$(A, \underline{S}_1) \xrightarrow{\ell} (A', S'_1) \tag{58}$$

and

$$T_L, \underline{T}_R \cup \{\mathtt{t}\},\ A',\ \mathtt{Fold}(S'_1, \mathtt{Top}(\underline{S}_1[\mathtt{t}/\mathtt{t}_1]))@ \circ S_2 \vdash \mathtt{Fold}(S', \mathtt{Top}(S)) \tag{59}$$

Observe that by premise of eq. (56) $\mathtt{Top}(S_1) = \emptyset$ hence $\mathtt{Top}(S_1[\mathtt{t}/\mathtt{t}_1]) = \emptyset$. It follows that @ is empty and eq. (59) is equivalent to

$$T_L, \underline{T}_R \cup \{\mathtt{t}\},\ A',\ S'_1 \circ S_2 \vdash \mathtt{Fold}(S', \mathtt{Top}(S)) \tag{60}$$

By $\langle\mathtt{rec}\rangle$ with premise eq. (58)

$$(A, \mu\mathtt{t}_1.\underline{S}_1[\mathtt{t}/\mathtt{t}_1]) \xrightarrow{\ell} (A', S'_1[\mu\mathtt{t}_1.\underline{S}_1[\mathtt{t}/\mathtt{t}_1]/rv_1])$$

By the transition above and Lemma F2

$$(A, \mu\mathtt{t}_1.\underline{S}_1) \xrightarrow{\ell} (A', S'_1[\mathtt{t}_1/\mathtt{t}][\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1]) \tag{61}$$

We need to prove

$$T_L, \underline{T}_R \cup \{\mathtt{t}\},\ A',\ \mathtt{Fold}(S'_1[\mathtt{t}_1/\mathtt{t}][\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1], \mathtt{Top}(\mu\mathtt{t}_1.\underline{S}_1))@ \circ S_2 \vdash \mathtt{Fold}(S', \mathtt{Top}(S)) \tag{62}$$

which, since $\texttt{Top}(\mu\texttt{t}_1.\underline{S}_1) = \texttt{t}_1$, is equivalent to

$$T_L, \underline{T}_R \cup \{\texttt{t}\},\ A',\ \texttt{Fold}(S_1'[\texttt{t}_1/\texttt{t}][\mu\texttt{t}_1.\underline{S}_1/\texttt{t}_1], \texttt{t}_1)@ \circ S_2 \vdash \texttt{Fold}(S', \texttt{Top}(S))$$

which, by applying the folding on $S_1'[\texttt{t}_1/\texttt{t}]$, is equivalent to

$$T_L, \underline{T}_R \cup \{\texttt{t}\},\ A',\ S_1'[\texttt{t}_1/\texttt{t}]@ \circ S_2 \vdash \texttt{Fold}(S', \texttt{Top}(S)) \tag{63}$$

In eq. (63), $@ = [\texttt{t}/\texttt{t}_1]$ since $\texttt{t}_1 \in \texttt{Top}(S_1)$ and $\texttt{t}_1 \notin \mathit{fn}(S)$ and $\texttt{t} \in \underline{T}_R \cup \{\texttt{t}\}$, hence eq. (63) is equivalent to eq. (60). The thesis eq. (62) holds therefore by eq. (60).

2. Second, assume

$$(A, S_2) \xrightarrow{\ell} (A', S_2') \tag{64}$$

By induction on the premise of eq. (56)

$$T_L, \underline{T}_R \cup \{\texttt{t}\},\ A',\ \underline{S}_1[\texttt{t}/\texttt{t}_1] \circ \texttt{Fold}(S_2', \texttt{Top}(S_2))@ \vdash \texttt{Fold}(S', \texttt{Top}(S)) \tag{65}$$

for some @. Applying eq. (65) as premise of [rec2] we obtain the thesis

$$T_L, \underline{T}_R \cup \{\texttt{t}\},\ A',\ \mu\texttt{t}_1.\underline{S}_1 \circ \texttt{Fold}(S_2', \texttt{Top}(S_2))@ \vdash \texttt{Fold}(S', \texttt{Top}(S))$$

as desired.

3. Finally, assume $\ell \in \{\oplus l, \& l\}$. By induction we have $(A, \underline{S}_1) \xrightarrow{\ell} (A', S')$ with $\mathit{fn}(S') \setminus \mathit{fn}(\underline{S}) = \emptyset$ (the case in which $(A, \underline{S}_2) \xrightarrow{\ell} (A', S')$ is symmetric). So, by $\langle\texttt{rec}\rangle$

$$(A, \mu\texttt{t}_1.S_1) \xrightarrow{\ell} (A', S'[\mu\texttt{t}_1.S_1/\texttt{t}_1]$$

Recall also that

$$(A, S) \xrightarrow{\ell} (A', S')$$

The thesis hold for $\overline{S} = S'$ since $\mathit{fn}(S') \setminus \mathit{fn}(S) = \emptyset$, $\mathit{fn}(S_1') \setminus \mathit{fn}(\mu\texttt{t}_1.\underline{S}_1) = \texttt{t}_1$.

Lemma 6 (Preservation - closed protocols). *Assume $\emptyset, \emptyset,\ A,\ S_1 \circ S_2 \vdash S$. For all ℓ, A', S_1 such that $(A, S) \xrightarrow{\ell} (A', S')$ either*

1. $(A, S_1) \xrightarrow{\ell} (A', S_1')$ and $\emptyset, \emptyset,\ A,\ S_1' \circ S_2 \vdash S'$ (S_1' up to unfolding), or
2. $(A, S_2) \xrightarrow{\ell} (A', S_2')$ and $\emptyset, \emptyset,\ A,\ S_1 \circ S_2' \vdash S'$ (S_2' up to unfolding)
3. $\ell \in \{\oplus l, \& l\}$ and $(A, S_i) \xrightarrow{\ell} (A', \overline{S}[S_1/\mathit{fn}(\overline{S}) \setminus \mathit{fn}(S_1)])$ with $i \in \{1, 2\}$ and $S' = \overline{S}[S/\mathit{fn}(\overline{S}) \setminus \mathit{fn}(S)]$ for some $\overline{S}$

Proof. This lemma holds for all variants of composition: $\vdash_s$, $\vdash_w$, $\vdash_c$, and $\vdash_{wc}$ (recall that notation $\vdash$ is used to refer to any of the aforementioned composition judgments). The proof focusses on proving $\vdash_{wc}$ which is the most general case; the other cases can be obtained by simply omitting the inductive cases for rules not used by that kind of composition (e.g., for $\vdash_s$ omit the [wbra] and [cbra] case).

We proceed by induction on derivation of S proceeding by case analysis on the last rule used.

Case [sym] The last rule applies is of the following form:

$$\frac{\emptyset, \emptyset,\ A,\ S_2\ \circ\ S_1 \vdash S}{\emptyset, \emptyset,\ A,\ S_1\ \circ\ S_2 \vdash S}$$

By induction either $(A, S_2) \xrightarrow{\ell} (A', S_2')$ and $\emptyset, \emptyset,\ A',\ S_2'\ \circ\ S_1 \vdash S'$ which applied as a premise of [sym] yields the thesis (item 2) $\emptyset, \emptyset,\ A',\ S_1 \circ S_2' \vdash S'$, or $(A, S_1) \xrightarrow{\ell} (A', S_1')$ and $\emptyset, \emptyset,\ A',\ S_2\ \circ\ S_1' \vdash S'$ which applied as a premise of [sym] yields the thesis (item 1) $\emptyset, \emptyset,\ A',\ S_1' \circ S_2 \vdash S'$. Alternatively, case (3) applies by induction and yields the thesis as item 3 is symmetric ($i \in \{1, 2\}$).

Case [consume] - $S = \texttt{consume}(n).\underline{S}$ proceeds as the corresponding case in Lemma F6.
The top of the derivation is of the following form:

$$\frac{\emptyset, \emptyset,\ A,\ \underline{S}_1\ \circ\ S_2 \vdash \underline{S}}{\emptyset, \emptyset,\ A \cup \{n\},\ \texttt{consume}(n).\underline{S}_1\ \circ\ S_2 \vdash \texttt{consume}(n).\underline{S}}$$

By $\langle\texttt{consume}\rangle$

$$(A \cup \{n\}, \texttt{consume}(n).S) \xrightarrow{\texttt{consume}(n)} (A, S)$$

and

$$(A \cup \{n\}, \texttt{consume}(n).S_1) \xrightarrow{\texttt{consume}(n)} (A, S_1)$$

The thesis holds as it is identical to the the premise of Equation (49).

Cases [pref][assume][assert] are similar to [consume].

Case [wbra] Proceeds as the corresponding case in Lemma F6. By hypothesis

$$\frac{\begin{array}{l} I = I_A \cup I_B \quad I_A \cup I_B \neq \emptyset \\ \forall i \in I_A.\emptyset, \emptyset,\ A,\ S_i\ \circ\ S_2 \vdash S_i' \\ \forall i \in I_B.\emptyset, \emptyset,\ A,\ S_i\ \circ\ S_2 \not\vdash\ \wedge\ A\{S_i\} \end{array}}{\emptyset, \emptyset,\ A,\ +\{\mathrm{l_i} : S_i\}_{i \in I}\ \circ\ S_2 \vdash +\{\mathrm{l_i} : S_i'\}_{i \in I_A} \cup \{l_i : S_i'\}_{i \in I_B}}$$

By $\langle\texttt{bra}\rangle$, picking $i \in I_A$ which is not empty by premise of the derivation above

$$(A, +\{\mathrm{l_i} : S_i\}_{i \in I}) \xrightarrow{+\mathrm{l}_j} (A, S_j)$$

and

$$(A, +\{\mathrm{l_i} : S_i'\}_{i \in I}) \xrightarrow{+\mathrm{l}_j} (A, S_j')$$

The thesis holds as it is identical to the the premise of the derivation above for $j \in I_A$.

Case [bra] This follows by [wbra] setting $I_B = \emptyset$.

Case [cbra] By hypothesis

$$\frac{\begin{array}{c}\forall i \in I \ J_i \neq \emptyset \quad \bigcup_{i\in I} J_i = J \\ \forall j \in J_i \quad \emptyset, \emptyset, \ A, \ S_i \ \circ \ S'_j \vdash S_{ij} \\ \forall j \in J \setminus J_i \quad \emptyset, \emptyset, \ A, \ S_i \ \circ \ S'_j \not\vdash \end{array}}{\emptyset, \emptyset, \ A, \ +\{l_i : S_i\}_{i\in I} \ \circ \ +'\{l'_j : S'_j\}_{j\in J} \vdash +\{l_i : +'\{l'_j : S_{ij}\}_{j\in J_i}\}_{i\in I}} \tag{66}$$

By $\langle \texttt{bra} \rangle$, picking $i \in I$

$$(A, +\{l_i : +'\{l'_j : S'_j\}_{j\in J_i}\}_{i\in I}) \xrightarrow{+l_i} (A, +'\{l'_j : S_{ij}\}_{j\in J_i})$$

and similarly

$$(A, +\{l_i : S_i\}_{i\in I}) \xrightarrow{+l_i} (A, S_i)$$

By applying [sym] to the second premise of eq. (66):

$$\forall j \in j_i +' \{l'_j : S'_j\}_{j\in J_i} \ \circ \ S_i \vdash +'\{l'_j : S_{ij}\}_{j\in J_i} \tag{67}$$

By applying eq. (67) as premise of [bra] and then [sym] we obtain the thesis (1):

$$\emptyset, \emptyset, \ A, \ S_i \ \circ \ +'\{l'_j : S'_j\}_{j\in J_i} \vdash +'\{l'_j : S_{ij}\}_{j\in J_i}$$

Case [rec1] - $S = \mu\texttt{t}_1.\underline{S}$ and $T_R = \emptyset$ By hypothesis

$$\frac{\{\texttt{t}_1\}, \emptyset, \ A, \ \underline{S}_1 \ \circ \ S_2 \vdash S \quad \texttt{Top}(\underline{S}_1) = \emptyset \quad A\{\mu\texttt{t}_1.S\}}{\emptyset, \emptyset, \ A, \ \mu\texttt{t}_1.\underline{S}_1 \ \circ \ S_2 \vdash \mu\texttt{t}_1.\underline{S}} \tag{68}$$

and

$$\frac{(A, \underline{S}) \xrightarrow{\ell} (A', S')}{(A, \mu\texttt{t}_1.\underline{S}) \xrightarrow{\ell} (A', S'[\mu\texttt{t}_1.\underline{S}/\texttt{t}_1])} \tag{69}$$

By Lemma F6 we have one of the following three cases:

1. $(A, \underline{S}_1) \xrightarrow{\ell} (A', S'_1)$ hence by $\langle \texttt{rec} \rangle$

   $$(A, \mu\texttt{t}_1.\underline{S}_1) \xrightarrow{\ell} (A', S'_1[\mu\texttt{t}_1.\underline{S}/\texttt{t}_1])$$

   and

   $$\{\texttt{t}_1\}, \emptyset, \ A', \ \texttt{Fold}(S'_1, \texttt{Top}(S_1)) \ \circ \ S_2 \vdash \texttt{Fold}(S', \texttt{Top}(S)) \tag{70}$$

   By premise of eq. (68) $\texttt{Top}(\underline{S}_1) = \emptyset$ and by Lemma F5 $\texttt{Top}(S) = \emptyset$. Hence by eq. (70) we obtain, with @ being the empty substitution:

   $$\{\texttt{t}_1\}, \emptyset, \ A', \ S'_1 \ \circ \ S_2 \vdash S' \tag{71}$$

   By premise of eq. (68) [rec1] $A\{\mu\texttt{t}_1.S\}$ which looking at the well formedness rule [rec] can be written as

   $$A\{\mu\texttt{t}_1.S\}A \cup A'' \tag{72}$$

for some A''. By Lemma 1 eq. (72) and eq. (69) imply

$$A'\{S'_1[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1]\}A''' \text{ such that } A''' \supseteq A \cup A'' \tag{73}$$

By Lemma F4 since eq. (68) and eq. (71) and eq. (73) we obtain

$$\emptyset, \emptyset,\ A',\ S'_1[\mu\mathtt{t}_1.S_1/\mathtt{t}_1] \circ S_2 \vdash S'[\mu\mathtt{t}_1.S/\mathtt{t}_1]$$

as desired.

2. $(A, \underline{S}_2) \xrightarrow{\ell} (A', S'_2)$ and

$$\{\mathtt{t}_1\}, \emptyset,\ A',\ S_1 \circ \mathtt{Fold}(S'_2, \mathtt{Top}(S_2)) \vdash \mathtt{Fold}(S', \mathtt{Top}(\underline{S})) \tag{74}$$

By Lemma F5, $\mathtt{Top}(\underline{S}) = \emptyset$ hence eq. (74) is equivalent to

$$\{\mathtt{t}_1\}, \emptyset,\ A',\ S_1 \circ \mathtt{Fold}(S'_2, \mathtt{Top}(S_2)) \vdash S' \tag{75}$$

We proceed by inner induction on the syntax of S_2.

- If $S_2 = p.\underline{S}_2$ then $S'_2 = \underline{S}_2$, and $\mathtt{Top}(S_2) = 0$ hence and @ is the empty substitution. Therefore, eq. (75) is equivalent to the thesis $\{\mathtt{t}_1\}, \emptyset,\ A',\ S_1 \circ S'_2 \vdash S'$ as desired;
- If $S_2 = a.\underline{S}_2$ with $a \in \{\mathtt{assert}(n), \mathtt{consume}(n), \mathtt{require}(n)\}$ the case is similar to the prefix case above;
- If $S_2 = \mathtt{end}$ or $S_2 = \mathtt{t}$ then $(A, S_2) \not\rightarrow$ hence done.
- If $S_2 = \mu\mathtt{t}_2.\underline{S}_2$ (interesting case) then $S'_2 = \underline{S}'_2[\mu\mathtt{t}_2.S_2/\mathtt{t}_2]$ with $(A, \underline{S}_2) \rightarrow (A', \underline{S}'_2)$ as premise of $\langle\mathtt{rec}\rangle$. Since $\mathtt{Top}(S_2) = \mathtt{t}_2$ and $\mathit{fn}(S') \not\ni \mathtt{t}_2$ then $@ = [\mathtt{t}_1/\mathtt{t}_2]$. Therefore, $\mathtt{Fold}(S'_2, \mathtt{Top}(S_2))@ = S'_2[\mathtt{t}_1/\mathtt{t}_2]$ and substituting this in eq. (75) we obtain

  $$\{\mathtt{t}_1\}, \emptyset,\ A',\ S_1 \circ S'_2[\mathtt{t}_1/\mathtt{t}_2] \vdash S'$$

  By applying Lemma F4 to the above we get

  $$\emptyset, \emptyset,\ A',\ S_1[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S'_2[\mathtt{t}_1/\mathtt{t}_2][\mu\mathtt{t}_2.\underline{S}_2/\mathtt{t}_1] \vdash S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

  which is equivalent to

  $$\emptyset, \emptyset,\ A',\ S_1[\mu\mathtt{t}_1.\underline{S}_1/\mathtt{t}_1] \circ S'_2[\mu\mathtt{t}_2.\underline{S}_2/\mathtt{t}_2] \vdash S'[\mu\mathtt{t}_1.\underline{S}/\mathtt{t}_1]$$

  as desired.
- By Lemma F6 either S_1 or S_2 makes a transition with label ℓ and

  $$(A, \mu\mathtt{t}_1.S_1) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}_1.S_1/\mathtt{t}_1] \qquad (A, \mu\mathtt{t}_1.S) \xrightarrow{\ell} (A', S'[\mu\mathtt{t}_1.S/\mathtt{t}_1])$$

  with $\mathit{fn}(S') \setminus \mathit{fn}(\mu\mathtt{t}_1.\underline{S}) = \mathtt{t}_1$ and $\mathit{fn}(S'_1) \setminus \mathit{fn}(\mu\mathtt{t}_1.\underline{S}_1) = \mathtt{t}_1$ hence the thesis.

Case [rec2] $S = \mu\mathtt{t}_1.\underline{S}$ and $T_R \neq \emptyset$ Contradicts the hypothesis ($T_R \neq \emptyset$) hence done.

F.3 $\boldsymbol{R2}$, $\boldsymbol{R3}$ and $\boldsymbol{R4}$ are simulations

Lemma F7

$$R2 = \{(S'[\mu\mathtt{t}.S/\mathtt{t}], S_1'[\mu\mathtt{t}.S_1/\mathtt{t}]) \mid S \lesssim S_1 \wedge S' \lesssim S_1' \wedge S' \neq \mathtt{t}\}$$

$\underline{R2}$ *is a simulation.*

Proof. (sketch) $(\underline{S}, \underline{S_1}) \in \underline{R2}$. So $\underline{S} = S'[\mu\mathtt{t}.S/\mathtt{t}]$ and $\underline{S_1} = S_1'[\mu\mathtt{t}.S_1/\mathtt{t}]$. We proceed by induction on S'. All cases are immediate as any step of S' is matched by a step of S_1' since $S' \lesssim S_1'$. Notice that the only case that would require care, $S' = \mathtt{t}$ is ruled out.

Lemma F8 *Let*

$$\underline{R3} = \{(S', S_1'[\mu\mathtt{t}.S_1/\mathtt{t}]) \mid S' \lesssim S_1'\}$$

$\underline{R3}$ *is a simulation.*

Proof. (sketch) We proceed by induction on S'. All cases (except the case $S' = \mathtt{t}$) are immediate as any step of S' is matched by a step of S_1' since $S' \lesssim S_1'$. If $S' = \mathtt{t}$ then $S' \not\rightarrow$ hence done.

Lemma F9 *Let*

$$\underline{R4} = \{(S, S_1 \mid S_2) \mid S \lesssim S_1\}$$

$\underline{R4}$ *is a simulation.*

Proof. (sketch) Straightforward by induction on the structure of S.

G Proofs of fairness

Definition 17. *Define the following context:*

$$\begin{array}{rl} C[\,\cdot\,] = & g.C[\,\cdot\,] \quad g \in \{p, \mathtt{assert}(n), \mathtt{consume}(n), \mathtt{require}(n)\} \\ \mid & +\{\mathrm{l} : C[\,\cdot\,]\} \cup \{\mathrm{l}_i : S_i\}_{i \in I} \\ \mid & \mu\mathtt{t}.C[\,\cdot\,] \\ \mid & [\,\cdot\,] \end{array}$$

Write $S = C[\,\cdot\,]$ if $S = C[\,S'\,]$ for some S'. Write $C' \in C$ (resp. $C' \notin C$) is there exists (resp. there exists no) C_1, C_2 such that $C = C_1[\,C'[\,C_3[\,\cdot\,]\,]\,]$. Define the following functions:

$$\mathtt{clab}(g.S) = \{g\} \qquad \mathtt{clab}(+\{\mathrm{l}_i : S_i\}_{i \in I}) = \{+\mathrm{l}_i\}_{i \in I} \qquad \mathtt{clab}(\mu\mathtt{t}.S) = \mathtt{clab}(S)$$

and

$$\begin{array}{l} \mathtt{V}(g.C[\,\cdot\,]) = g, \mathtt{V}(C[\,\cdot\,]) \qquad \mathtt{V}([\,\cdot\,]) = \epsilon \qquad \mathtt{V}(\mu\mathtt{t}.C[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,]) \\ \mathtt{V}(+\{\mathrm{l}_j : C[\,\cdot\,]\} \cup \{\mathrm{l}_i : S_i\}_{i \in I \setminus j}) = +\mathrm{l}_j, \mathtt{V}(C[\,\cdot\,]) \end{array}$$

Lemma G1 *If* $(A, S) \xrightarrow{\ell} (A', S')$ *then* $(A, S[\underline{S}/\mathtt{t}]) \xrightarrow{\ell} (A', S'[\underline{S}/\mathtt{t}])$

Proof. (sketch) Mechanical by induction on the transition, by case analysis on the last rule used to make step ℓ.

Lemma G2 *If* $T_L,\ T_R,\ A,\ S_0 \circ S_1 \vdash S$ *and* $S_1 = \mathtt{end}$ *then* $S_0 = S$

Proof. (sketch) By induction on the proof of S proceeding by case analysis on the last rule used, observing that the last rule used cannot be $[rec1]$ or $[rec2]$ as the only axiom that can be used if $[end]$ (i.e., not $[call]$) due to the form of S_1.

Lemma G3 *If* $(A, S) \xrightarrow{\ell}$ *then* $\ell \in \mathtt{clab}(S)$*.*

Proof. (sketch) Mechanical by induction on the derivation of transition $\xrightarrow{\ell}$ proceeding by case analysis on the last transition rule used.

Lemma G4 *If* $(A, S) \xrightarrow{\boldsymbol{r}}$ *then* $S = C[S']$ *for some* S' *and* $\mathtt{V}(C) = \boldsymbol{r}$*.*

Proof. (sketch) First prove that $(A, S) \xrightarrow{r}$ implies $S = C[S']$ for some S' and $\mathtt{V}(C) = r$ by induction on the transition proceeding by case analysis on the last transition rule used. Then by induction on the size of $\boldsymbol{r}$ based on the fact that contexts compositionality.

Lemma G5 *If* $S = C[\,S'\,]$*,* $A\{S\}$*, and* $\ell \in \mathtt{clab}(S')$*, then* $(A, S) \xrightarrow{\boldsymbol{r}}\xrightarrow{\ell}$ *for some (possibly empty) vector* $\boldsymbol{r}$ *of transition labels such that* $\mathtt{V}(C) = \boldsymbol{r}$*.*

Proof. By induction on the syntax of C.

- Case $C = [\,\cdot\,]$ (and hence $\mathtt{V}(C)$ is the empty vector of labels). We show the case for $S' = \mathtt{consume}(n).S''$ and hence $\mathtt{clab}(S) = \{\mathtt{consume}(n)\}$. By well-assertedness of S, which in this case last applies rule [consume], $n \in A$, then by semantic rule $\langle\mathtt{consume}\rangle$ we have

$$(A, \mathtt{consume}(n).S') \xrightarrow{\mathtt{consume}(n)} (A \setminus \{n\}, S')$$

  as desired. The cases for $S' \in \{\mathtt{require}(n).S'', \mathtt{assert}(n).S'', p.S''\}$ are similar.
- If $C = \mathtt{consume}(n).C'[\,\cdot\,]$ then we proceed with a generic S'. By well-assertedness of S which last applies rule [consume] we have $n \in A$ hence by semantic rule $\langle\mathtt{consume}\rangle$ we have

$$(A, \mathtt{consume}(n).C'[S']) \xrightarrow{\mathtt{consume}(n)} (A \setminus \{n\}, C'[S'])$$

  By Lemma 1 (well-assertedness is preserved by transition) we have

$$A \setminus \{n\}\{C'[\,\underline{C}[\,S'\,]\,]\}$$

By induction $(A \setminus \{n\}, C'[\,S'\,]) \xrightarrow{\boldsymbol{r}}\xrightarrow{\ell}$ with $\ell \in \texttt{lab}(S')$ and $\texttt{V}(C') = \boldsymbol{r}$ hence

$$(A, \texttt{consume}(n).C'[S']) \xrightarrow{\texttt{consume}(n)}\xrightarrow{\boldsymbol{r}}\xrightarrow{\ell}$$

with $\ell \in \texttt{lab}(S')$ and $\texttt{V}(C) = \texttt{consume}(n), \texttt{V}(C') = \texttt{consume}(n), \boldsymbol{r}$ as desired.

- The other cases for $C = g.C'[\cdot]$ are similar to the case above.
- If $C = +\{\mathrm{l} : C'[\cdot]\} \cup \{\mathrm{l}_i : S_i\}_{i \in I}$. By semantic rule $\langle\texttt{branch}\rangle$ we have

  $$(A, +\{\mathrm{l} : C'[S']\} \cup \{\mathrm{l}_i : S_i\}_{i \in I}) \xrightarrow{+\mathrm{l}} (A, C'[S'])$$

  By Lemma 1 (well-assertedness is preserved by transition) we have

  $$A\{C'[\,S'\,]\}$$

  By induction $(A, C'[\,S'\,]) \xrightarrow{\boldsymbol{r}}\xrightarrow{\ell}$ with $\ell \in \texttt{clab}(S')$ and $\texttt{V}(S') = \boldsymbol{r}$ hence

  $$(A, +\{\mathrm{l} : C'[S']\} \cup \{\mathrm{l}_i : S_i\}_{i \in I}) \xrightarrow{+\mathrm{l}}\xrightarrow{\boldsymbol{r}}\xrightarrow{\ell}$$

  with $\ell \in \texttt{clab}(S')$ and $\texttt{V}(C) = +\mathrm{l}, \texttt{V}(C) = +\mathrm{l}, \boldsymbol{r}$ as desired.
- If $C = \mu\texttt{t}.C'[\cdot]$ then by well-assertedness of $A\{S\}$. In this case the last rule applied is $[rec]$. We have by premise of $[rec]$, $A\{C'[\,S'\,]\}$. Hence, by Lemma 2 (well-asserted protocols are not stuck)

  $$(A, C'[\,S'\,]) \xrightarrow{\ell'} (A', C''[\,S'\,]) \tag{76}$$

  for some C'' and by Lemma 1 $A'\{C''[\,S'\,]\}$. By induction

  $$(A', C'[\,S'\,]) \xrightarrow{\ell'}\xrightarrow{\boldsymbol{r}}\xrightarrow{\ell} \qquad \ell \in \texttt{clab}(S') \quad \ell', \boldsymbol{r} = \texttt{V}(C'[\,\cdot\,]) \tag{77}$$

  By $\langle\texttt{Rec}\rangle$ with as premise the first transition of eq. (77):

  $$(A, \mu\texttt{t}.C'[\,S'\,]) \xrightarrow{\ell} (A', C''[\,S'\,][\mu\texttt{t}.C'[\,S'\,]/\texttt{t}])$$

  By Lemma G1 and eq. (77)

  $$(A', C''[\,S'\,][\mu\texttt{t}.C'[\,S'\,]/\texttt{t}]) \xrightarrow{\boldsymbol{r}}\xrightarrow{\ell}$$

  hence

  $$(A, \mu\texttt{t}.C'[\,S'\,]) \xrightarrow{\ell'}\xrightarrow{\boldsymbol{r}}\xrightarrow{\ell} \qquad \ell \in \texttt{clab}(S')$$

  $\texttt{V}(\mu\texttt{t}.C'[\,\cdot\,]) = \texttt{V}(C'[\,\cdot\,])$ and by induction $\texttt{V}(\mu\texttt{t}.C'[\,\cdot\,]) = \ell', \boldsymbol{r}$ as desired.

Lemma G6 *If* $T_L,\ T_R,\ A,\ S_0 \circ S_1 \vdash S$ *then* $\forall \ell \in \texttt{clab}(S_1)\, \exists C[\,\cdot\,], C_0[\,\cdot\,], S', S'_0$ *such that*

1. $S = C[\,S'\,]$ *and* $S_0 = C_0[\,S'_0\,]$
2. $\texttt{V}(C[\,\cdot\,]) = \texttt{V}(C_0[\,\cdot\,])$
3. $\ell \in \texttt{clab}(S')$

Proof. We proceed by induction on the proof of S, proceeding by case analysis of the last rule applied.

Case [end] In this case $\texttt{clab}(S_1) = \emptyset$ hence done.

Case [call] In this case $\texttt{clab}(S_1) = \emptyset$ hence done.

Case [act] Fix $\ell \in \texttt{clab}(S_1)$. In this case $S_0 = p.S'_0$ and $S = p.S'$ then

$$\frac{T_L,\ T_R,\ A,\ S'_0 \circ S_1 \vdash S'}{T_L,\ T_R,\ A,\ p.S'_0 \circ S_1 \vdash p.S'}$$

By induction there exists $C[\,S''\,] = S'$ and $C_0[\,S''_0\,] = S'_0$ such that $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_0[\,\cdot\,])$ and $\ell \in \texttt{clab}(S'')$. Hence there exists $p.C$ and $p.C_0$ such that $C[\,S''_0\,] = S_0$ and $p.C[\,S''\,] = p.S' = S$ and $p.C_0[\,S''_0\,] = p.S'_0 = S_0$. Moreover, since $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_0[\,\cdot\,])$ then $p, \mathtt{V}(C[\,\cdot\,]) = p, \mathtt{V}(C_0[\,\cdot\,])$ and hence $\mathtt{V}(p.C[\,\cdot\,]) = \mathtt{V}(p.C_0[\,\cdot\,])$. Finally, still $\ell \in \texttt{clab}(S'')$ as desired.

Case [consume] [assert], [require] Are similar to [act].

Case [wbra] Fix $\ell \in \texttt{clab}(S_1)$. In this case $S = +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup \{\mathrm{l}_i : S_i\}_{i \in I_B}$, $S_0 = +\{\mathrm{l_i} : S_i\}_{i \in I}$ and

$$\frac{\begin{array}{l} I_A \cup I_B = I \quad I_A \cap I_B = \emptyset \\ \forall i \in I_A.\ T_L,\ T_R,\ A,\ S_i \circ S_1 \vdash S'_i \\ \forall i \in I_B.\ A\{S_i\} \quad T_L,\ T_R,\ A,\ S_i \circ S_1 \nvdash \end{array}}{T_L,\ T_R,\ A,\ +\{\mathrm{l_i} : S_i\}_{i \in I} \circ S_1 \vdash +\{\mathrm{l_i} : S'_i\}_{i \in I_A} \cup \{\mathrm{l}_i : S_i\}_{i \in I_B}}$$

By induction $\forall i \in I_A$ with $I_A \neq \emptyset$ there exist $C_i[\,\underline{S}_i\,] = S_i$ and $C'_i[\,\underline{S}'_i\,] = S'_i$ such that $\mathtt{V}(C_i[\,\cdot\,]) = \mathtt{V}(C'_i[\,\cdot\,])$ and $\ell \in \texttt{clab}(\underline{S}'_i)$.

Hence, there exists $C_0[\,\underline{S}_i\,] = +\{\mathrm{l}_i : C_i[\,\underline{S}_i\,]\} \cup \{\mathrm{l}_j : S_j\}_{j \in I \setminus \{i\}}$ and $C[\,\underline{S}'_i\,] = +\{\mathrm{l}_i : C'_i[\,\underline{S}'_i\,]\} \cup \{\mathrm{l}_j : S'_j\}_{j \in I_A \setminus \{i\}} \cup \{\mathrm{l}_i : S_i\}_{i \in I_B}$. Moreover, $\mathtt{V}(C_i[\,\cdot\,]) = \mathtt{V}(C'_i[\,\cdot\,])$ implies $\mathrm{l}_i, \mathtt{V}(C_i[\,\cdot\,]) = \mathrm{l}_i, \mathtt{V}(C'_i[\,\cdot\,])$ and hence $\mathtt{V}(C_0[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,])$. Finally, still $\ell \in \texttt{clab}(\underline{S}'_i)$ as desired.

Case [bra] As the case [wbra] assuming $I_B = \emptyset$.

Case [cbra] Fix $\ell \in \texttt{clab}(S_1)$. In this case $S = +\{\mathrm{l_i} : +'\{\mathrm{l_j} : S_{ij}\}_{j \in J_i}\}_{i \in I}$, $S_0 = +\{\mathrm{l_i} : S_i\}_{i \in I}$ and

$$\frac{\begin{array}{l} \forall i \in I\ J_i \neq \emptyset \quad \bigcup_{i \in I} J_i = J \\ \qquad j \in J_i.\ T_L,\ T_R,\ A,\ S_i \circ S_j \vdash S_{ij} \\ \qquad \forall j \in J \setminus J_i\ T_L,\ T_R,\ A,\ S_i \circ S_j \nvdash \end{array}}{T_L,\ T_R,\ A,\ +\{\mathrm{l_i} : S_i\}_{i \in I} \circ +'\{\mathrm{l_j} : S_j\}_{j \in J} \vdash +\{\mathrm{l_i} : +'\{\mathrm{l_j} : S_{ij}\}_{j \in J_i}\}_{i \in I}}$$

By induction $\forall i \in I$ there exist $C_i[\,\underline{S}_i\,] = S_i$ and $C'_i[\,\underline{S}_{ij}\,] = S_{ij}$ such that $\mathtt{V}(C_i[\,\cdot\,]) = \mathtt{V}(C'_i[\,\cdot\,])$ and $\ell \in \texttt{clab}(\underline{S}_{ij})$.

Hence, forall $\mathrm{l}_j \in \texttt{clab}(S_0)$ (which is non empty since $I \neq \emptyset$) there exist $j \in J_i, C_1[\,\underline{S}_i\,] = +\{\mathrm{l}_i : C_i[\,\underline{S}'_i\,]\} \cup \{\mathrm{l}_j : S_j\}_{j \in I \setminus \{j\}}$ and $C[\,\underline{S}_{ij}\,] = +\{\mathrm{l}_i : C'_i[\,\underline{S}_{ij}\,]\} \cup \{\mathrm{l}_j : S_{ij}\}_{j \in J_i \setminus \{i\}}$, $\mathtt{V}(C_0[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,]) = \mathrm{l}_i, \mathtt{V}(C_i[\,\cdot\,]) = \mathrm{l}_i, \mathtt{V}(C'_i[\,\cdot\,])$ and still $\ell \in \texttt{clab}(\underline{S}_{ij})$.

Case [rec1] Fix $\ell \in \mathtt{clab}(S_1)$. In this case $S_0 = \mu\mathtt{t}.S_0'$, $S = \mu\mathtt{t}.S'$ (and for simplicity we leave the recursive form of S_1 implicit as it is immaterial here). By composition rule [rec1]:

$$\dfrac{T_L \cup \{\mathtt{t}\},\ T_R,\ A,\ S_0' \circ S_1 \vdash S'}{T_L,\ T_R,\ A,\ \mu\mathtt{t}.S_0' \circ S_1 \vdash \mu\mathtt{t}.S'}$$

By induction there exist $C[\,\underline{S'}\,] = S'$ and $C_0[\,\underline{S_0'}\,] = S_0'$ such that $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_0[\,\cdot\,])$ and $\ell \in \mathtt{clab}(\underline{S'})$. Hence there exists $\mu\mathtt{t}.C[\,\underline{S'}\,] = S$ and $\mu\mathtt{t}.C_0[\,\underline{S_0'}\,] = S_0$. Moreover, since by induction $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_0[\,\cdot\,])$ and hence $\mathtt{V}(\mu\mathtt{t}.C[\,\cdot\,]) = \mathtt{V}(\mu\mathtt{t}.C_0[\,\cdot\,])$ (since $\mathtt{V}(\mu\mathtt{t}.C[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,])$ and $\mathtt{V}(\mu\mathtt{t}.C_0[\,\cdot\,]) = \mathtt{V}(C_0[\,\cdot\,])$ by definition of $\mathtt{V}()$). Finally, still $\ell \in \mathtt{clab}(\mu\mathtt{t}.\underline{S'})$ as desired.

Case [rec2] Fix $\ell \in \mathtt{clab}(S_1)$. In this case

$$\dfrac{T_L,\ T_R \cup \mathtt{t}',\ A,\ S_0'[\mathtt{t}'/\mathtt{t}] \circ S_1 \vdash S}{T_L,\ T_R \cup \mathtt{t}',\ A,\ \mu\mathtt{t}.S_0' \circ S_1 \vdash S}$$

By induction there exists $C[\,\underline{S'}\,] = S$ and $C_0[\,\underline{S_0'}\,] = S_0'[\mathtt{t}'/\mathtt{t}]$ such that $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_0[\,\cdot\,])$ and $\ell \in \mathtt{clab}(\underline{S'})$.

Hence there exists $C[\,\underline{S'}\,] = S$ and $\mu\mathtt{t}.C_0[\mathtt{t}/\mathtt{t}'][\,\underline{S_0'}[\mathtt{t}/\mathtt{t}']\,] = S_0$. By induction $\mathtt{V}(C_0) = \mathtt{V}(C)$ and by definition of $\mathtt{V}()$ (observing that $\mathtt{t}'$ does not affect the returned value), $\mathtt{V}(\mu\mathtt{t}.C_0[\mathtt{t}/\mathtt{t}']) = \mathtt{V}(C_0[\mathtt{t}/\mathtt{t}'])$ and $\mathtt{V}(C_0[\mathtt{t}/\mathtt{t}']) = \mathtt{V}(C_0)$. Hence $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(\mu\mathtt{t}.C_0[\mathtt{t}/\mathtt{t}'])$ and still $\ell \in \mathtt{clab}(\mu\mathtt{t}.\underline{S'}) = \mathtt{clab}(\underline{S'})$.

Case [sym] In this case

$$\dfrac{T_L,\ T_R,\ A,\ S_1 \circ S_0 \vdash S}{T_L,\ T_R,\ A,\ S_0 \circ S_1 \vdash S} \tag{78}$$

Assume that [sym] is applied only once. If [sym] it is applied multiple (but finite) times subsequently, say n times, then if n is even the thesis is immediate by hypothesis, and if n is odd then the case is equivalent to the one where the rule is applied once. Fix $\ell \in \mathtt{clab}(S_1)$. If $\mathtt{clab}(S_0) \neq \emptyset$ then by induction there exists $C[\,S'\,] = S$ and $C_1[\,S_1'\,]$ such that $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_1[\,\cdot\,])$ and $\ell \in \mathtt{clab}(S')$. From $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(C_1[\,\cdot\,])$, $C[\,S'\,] = S$ and $C_1[\,S_1'\,]$ it follows that S and S_1 have the same first prefix hence

$$\ell \in \mathtt{clab}(S)$$

hence the thesis with contexts $[\,\cdot\,]$ for S and S_0 and trivially $\mathtt{V}([\,\cdot\,]) = \mathtt{V}([\,\cdot\,])$. If $\mathtt{clab}(S_0) = \emptyset$ then either $S_0 = \mathtt{end}$ or $S_0 = \mathtt{t}$. In either case $S_1 = S$: by lemma G2 if $S_0 = \mathtt{end}$ and by [call] if $S_0 = \mathtt{t}$. Hence with contexts $[\,\cdot\,]$ for S and S_0 trivially $\mathtt{clab}(S_1) = \mathtt{clab}(S)$ and hence $\ell \in \mathtt{clab}(S)$ and $\mathtt{V}([\,\cdot\,]) = \mathtt{V}([\,\cdot\,])$

Lemma G7 is a stronger version of Lemma G6 where quantification over contexts is universal rather than existential, and holds only for strong composition (not for weak one).

Lemma G7 *If* $T_L,\ T_R,\ A,\ C_0[\,S_0\,] \circ S_1 \vdash_s S$ *and* $\mathtt{clab}(S_1) \neq \emptyset$*, then either:*

1. *there exist* C_0', C_0'', $C[S'] = S$ *such that*
   - $C_0[\,\cdot\,] = C_0'[\,C_0''[\,\cdot\,]\,]$, *and*
   - $\mathtt{V}(C_0'[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,])$, *and*
   - $\mathtt{clab}(S') = \mathtt{clab}(S_1)$, *or*
2. *there exist* $C_0'[S_0']$, $C[S'] = S$ *such that*
   - $C_0[\,C_0'[\,S'\,]\,] = S_0$, *and*
   - $\mathtt{V}(C_0[\,C_0'[\,\cdot\,]\,]) = \mathtt{V}(C[\,\cdot\,])$, *and*
   - $\mathtt{clab}(S') = \mathtt{clab}(S_1)$

Proof. We proceed by induction on the syntax of C_0.

Case $C_0[\,\cdot\,] = p.\underline{C}_0[\,\cdot\,]$ By hypothesis

$$\frac{T_L,\ T_R,\ A,\ \underline{C}_0[\,S_0\,] \circ S_1 \vdash_s S}{T_L,\ T_R,\ A,\ p.\underline{C}_0[\,S_0\,] \circ S_1 \vdash_s p.S}$$

By induction either of the following holds:

1. there exist $C_0'[\,C_0''[\,\cdot\,]\,] = \underline{C}_0[\,\cdot\,]$ and $C[S'] = S$ such that $\mathtt{V}(\underline{C}_0'[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,])$ and $\mathtt{clab}(S') = \mathtt{clab}(S_1)$. Therefore there exist $p.C_0'[\,C_0''[\,\cdot\,]\,] = C_0[\,\cdot\,]$ and $p.C[S'] = p.S$ such that $\mathtt{V}(C_0'[\,\cdot\,]) = p, \mathtt{V}(\underline{C}_0'[\,\cdot\,]) = p, \mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(p.C[\,\cdot\,])$ and $\mathtt{clab}(S') = \mathtt{clab}(S_1)$.
2. there exist $C_0'[S_0']$, $C[S'] = S$ such that $\underline{C}_0[\,C_0'[\,S'\,]\,] = S_0$, $\mathtt{V}(\underline{C}_0[\,C_0'[\,\cdot\,]\,]) = \mathtt{V}(C[\,\cdot\,])$, and $\mathtt{clab}(S') = \mathtt{clab}(S_1)$. Therefore there exist $C_0'[S_0']$, $p.C[S'] = S$ such that $p.\underline{C}_0[\,C_0'[\,S'\,]\,] = p.S_0$, $\mathtt{V}(p.\underline{C}_0[\,C_0'[\,\cdot\,]\,]) = p, \mathtt{V}(\underline{C}_0[\,C_0'[\,\cdot\,]\,]) = p, \mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(p.C[\,\cdot\,])$ and $\mathtt{clab}(S') = \mathtt{clab}(S_1)$.

The cases for consume, assert, and require are similar to the prefix case above.

Case $C_0[\,\cdot\,] = +\{\mathrm{l}_j : \underline{C}_0[\,\cdot\,]\} \cup \{\mathrm{l}_i : S_i\}_{i\in I\setminus\{j\}}$ By hypothesis

$$\frac{\begin{array}{ll}\forall i \in I \setminus \{j\} & T_L,\ T_R,\ A,\ S_i \circ S_1 \vdash_s S_i' \\ \underline{C}_0[\,\underline{S}_j] = S_j & T_L,\ T_R,\ A,\ \underline{C}_1[\,\underline{S}_j] \circ S_1 \vdash_s S_j'\end{array}}{T_L,\ T_R,\ A,\ +\{\mathrm{l}_j : \underline{C}_0[\,\underline{S}_j\,]\} \cup \{\mathrm{l}_i : S_i\}_{i\in I\setminus\{j\}} \circ S_1 \vdash_s +\{\mathrm{l}_i : S_i\}_{i\in I}}$$

By induction either of the following holds:

1. there exist $C_0'[\,C_0''[\,\cdot\,]\,] = \underline{C}_0[\,\cdot\,]$ and $C[S_j''] = S_j'$ such that $\mathtt{V}(C_0'[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,])$ and $\mathtt{clab}(S_j'') = \mathtt{clab}(S_1)$. Therefore there exist $+\{\mathrm{l}_j : C_0'[\,C_0''[\,\cdot\,]\,]\} \cup \{\mathrm{l}_i : S_i\}_{i\in I\setminus\{j\}} = C_0[\,\cdot\,]$ and $+\{\mathrm{l}_j : C[\,S_j''\,]\} \cup \{\mathrm{l}_i : S_i\}_{i\in I\setminus\{j\}} = +\{\mathrm{l}_i : S_i\}_{i\in I}$ such that $\mathtt{V}(+\{\mathrm{l}_j : C_0'[\,\cdot\,]\} \cup \{\mathrm{l}_i : S_i\}_{i\in I\setminus\{j\}}) = \mathrm{l}_j, \mathtt{V}(C_0'[\,\cdot\,]) = \mathtt{V}(+\{\mathrm{l}_j : C[\,\cdot\,]\} \cup \{\mathrm{l}_i : S_i'\}_{i\in I\setminus\{j\}})$ and $\mathtt{clab}(S_j') = \mathtt{clab}(S_1)$.

2. there exist $C'_0[S'''_j]$, $C[S''_j] = S'_j$ such that $C_0[\,C'_0[\,S'''_j\,]\,] = S_j$, $\mathtt{V}(\underline{C}_0[\,C'_0[\,\cdot\,]\,]) = \mathtt{V}(C[\,\cdot\,])$, and $\mathtt{clab}(S') = \mathtt{clab}(S_1)$. Therefore there exist $\underline{C}_0[\,C'_0[S'''_j]\,]$, $+\{\mathrm{l}_j : C[\,S''_j\,]\} \cup \{\mathrm{l}_i : S_i\}_{i \in I \setminus \{j\}} = +\{\mathrm{l}_i : S_i\}_{i \in I}$ such that $\mathtt{V}(C_0) = \mathrm{l}_j$, $\mathtt{V}(\underline{C}_0[\,C'_0[\,\cdot\,]\,]) = \mathrm{l}_j$, $\mathtt{V}(C[\,\cdot\,]) = \mathtt{V}(+\{\mathrm{l}_j : C[\,\cdot\,]\} \cup \{\mathrm{l}_i : S_i\}_{i \in I \setminus \{j\}})$ and $\mathtt{clab}(S') = \mathtt{clab}(S_1)$.

Case $C_0[\,\cdot\,] = \mu\mathtt{t}.\underline{C}_0[\,\cdot\,]$ By induction observing that $\mathtt{V}(\mu\mathtt{t}.\underline{C}_0[\,\cdot\,]) = \mathtt{V}(\underline{C}_0[\,\cdot\,])$.

Case $C_0[\,\cdot\,] = [\,\cdot\,]$ Immediate by induction.

Theorem 2 (Fairness of compositions with $\vdash$). *If* $\emptyset, \emptyset, A, S_0 \circ S_1 \vdash S$ *then* S *is* fair *w.r.t.* S_0 *and* S_1 *on* A.

Proof. Immediately from Lemma G8.

Lemma G8 (Fairness) *Let* $\emptyset, \emptyset, A, S_0 \circ S_1 \vdash S$*. Then* $\forall i \in \{0, 1\}$ *and any transition* $(A_i, S_i) \xrightarrow{\ell} (A'_i, S'_i)$ *there exists* $\boldsymbol{r}$ *such that: (1)*

- $(A, S_{|1-i|}) \xrightarrow{\boldsymbol{r}} (A'_{|1-i|}, S'_{|1-i|})$
- $(A, S) \xrightarrow{\boldsymbol{r}\ell} (A'', S')$
- $\emptyset, \emptyset, A'', S'_0 \circ S'_1 \vdash S'$.

Proof. Assume $(A, S_1) \xrightarrow{\ell} (A'_1, S'_1)$. By Lemma G3 $\ell \in \mathtt{clab}(S_1)$ so, by Lemma G6, there are two contexts C_0 and C such that the hypothesis can be rewritten as

$$\emptyset, \emptyset, A, C_0[\,S'_0\,] \circ S_1 \vdash C[\,S'\,]$$

with

$$\mathtt{V}(C_0[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,]) \tag{79}$$

and

$$\ell \in \mathtt{clab}(S') \tag{80}$$

By eq. (79), eq. (80) and Lemma G5

$$\begin{array}{ll} (A, S_0) \xrightarrow{\boldsymbol{r}} (A'_0, S'_0) & (\textit{for some } A'_0, S'_0) \\ (A, S) \xrightarrow{\boldsymbol{r}\ell} (A', S') & (\textit{for some } A'', S') \end{array} \tag{81}$$

It remains to prove that

$$\emptyset, \emptyset, A'', S'_0 \circ S'_1 \vdash S' \tag{82}$$

For every transition $r \in \boldsymbol{r}$, by case (1) of Lemma 6 the composition relation is preserved. More precisely, let $\boldsymbol{r} = r_0, \ldots, r_n$:

$\emptyset, \emptyset, A, S_0 \circ S_1 \vdash S \;\wedge\; (A, S_0) \xrightarrow{r_0} (A^1, S^1_0) \;\wedge\; (A, S) \xrightarrow{r_0} (A^1, S^1) \Rightarrow \emptyset, \emptyset, A^1, S^1_0 \circ S_1 \vdash S^1$
$\ldots$
$\emptyset, \emptyset, A^n, S^n_0 \circ S_1 \vdash S^n \;\wedge\; (A^n, S^n_0) \xrightarrow{r_n} (A', S'_0) \;\wedge\; (A^n, S^n) \xrightarrow{r_n} (A', S^{n+1}) \Rightarrow \emptyset, \emptyset, A^{n+1}, S'_0 \circ S_1 \vdash S^{n+1}$

Note that, when using Lemma 6, case (1) of Lemma 6 can always apply (case 2 applies for the symmetric case in which S_0 moves first). Assume by contradiction that only case (3) applies (the case where continuations are not preserved), then S_0 and S would move to a state in which they are both $\underline{S}$. By taking any S_0 (and hence S) that does not include any ℓ action we have a counter-example for eq. (81) (second row) already proved above. Hence case (1) must always be applicable. Hence done.

Assume now that $(A, S_0) \xrightarrow{\ell} (A', S_0')$. Then by applying $[sys]$ after the last composition rule in the hypothesis we obtain

$$\emptyset,\ \emptyset,\ A,\ S_1 \circ S_0 \vdash S'$$

and the case is then identical to the one where S_1 moves, proved above.

Theorem 3 (Strong fairness of compositions with $\vdash_s$). *Assume*

$$\emptyset,\ \emptyset,\ A,\ S_0 \circ S_1 \vdash_s S$$

then S is strongly fair *with respect to S_0 and S_1 on A.*

Proof. Immediately from Lemma G9.

Lemma G9 *Let $\emptyset, \emptyset,\ A,\ S_0 \circ S_1 \vdash_s S$. Then $\forall i \in \{0,1\}$ and all transitions $(\_, S_i) \xrightarrow{\ell} (\_, S_i')$ and $(A, S_{|1-i|}) \xrightarrow{\boldsymbol{r}}$, there exist $\boldsymbol{r'}$, $\boldsymbol{r''}$ with $(A, S_{|1-i|}) \xrightarrow{\boldsymbol{r'}} (\_, S'_{|1-i|})$ with either*

1. *$\boldsymbol{r'r''} = \boldsymbol{r}$ ($\boldsymbol{r'}$ is a prefix of $\boldsymbol{r}$), or*
2. *$\boldsymbol{r'} = \boldsymbol{rr''}$ ($\boldsymbol{r}$ is an ex prefix of $\boldsymbol{r'}$)*

such that $(A, S) \xrightarrow{\boldsymbol{r'}\ell} (A', S')$ and $\emptyset, \emptyset,\ A',\ S_0' \circ S_1' \vdash_s S'$.

Proof. We fix $i = 1$. By Lemma G3 if $(A, S_1) \xrightarrow{\ell} (A', S_1')$ then $\ell \in \mathtt{clab}(S_1)$ and hence $\mathtt{clab}(S_1) \neq \emptyset$. Fix any $\boldsymbol{r}$ such that $(A, S_0) \xrightarrow{\boldsymbol{r}}$. By Lemma G4 we can rewrite S_0 as $C_0[S_0'']$ with $\mathtt{V}(C_0[\,\cdot\,]) = \boldsymbol{r}$. By Lemma G7, since $\mathtt{clab}(S_1) \neq \emptyset$, for C_0 either

1. there exist C_0', C_0'', $C[S''] = S$ such that
   - $C_0[\,\cdot\,] = C_0'[C_0''[\,\cdot\,]]$, and
   - $\mathtt{V}(C_0'[\,\cdot\,]) = \mathtt{V}(C[\,\cdot\,])$, and
   - $\mathtt{clab}(S'') = \mathtt{clab}(S_1)$, or
2. there exist $C_0'[S_0']$, $C[S''] = S$ such that
   - $C_0[\,C_0'[\,S''\,]\,] = S_0$, and

- $\mathtt{V}(C_0[\,C_0'[\,\cdot\,]\,]) = \mathtt{V}(C[\,\cdot\,])$, and
- $\mathtt{clab}(S'') = \mathtt{clab}(S_1)$

In case (1) above, we can write S_0 as $C_0'[S_0''']$ for some S_0''', $\boldsymbol{r'} = \mathtt{V}(C_0')$, and $\boldsymbol{r''} = \mathtt{V}(C_0'')$. By Lemma G4

$$(A, C_0'[S_0''']) \xrightarrow{\boldsymbol{r'}} (\_, S_0')$$

for some S_0'. Since $\mathtt{clab}(S'') = \mathtt{clab}(S_1)$ then $\ell \in \mathtt{clab}(S'')$. Since $A\{S\}$ by hypothesis (it is a composition) and $\ell \in \mathtt{clab}(S'')$ then by Lemma G5

$$(A, C[\,S''\,]) \xrightarrow{\boldsymbol{r'}\ell} (A', S')$$

for some A' and S'.

In case (2) above, we set $\boldsymbol{r'} = \mathtt{V}(C_0[\,C_0'[\,\cdot\,]\,])$ and we can write S_0 as $C_0[\,C_0'[\,S_0'''\,]\,]$ for some S_0'''. By Lemma G4

$$(A, C_0[\,C_0'[\,S_0'''\,]\,] \xrightarrow{\boldsymbol{r'}} (\_, S_0')$$

for some S_0'. Since $\mathtt{clab}(S'') = \mathtt{clab}(S_1)$ then $\ell \in \mathtt{clab}(S'')$. Since $A\{S\}$ by hypothesis (it is a composition) and $\ell \in \mathtt{clab}(S'')$ then by Lemma G5

$$(A, C[\,S''\,]) \xrightarrow{\boldsymbol{r'}\ell} (A', S')$$

for some A' and S'.

In both case (1) and case (2) above, it remains to prove that

$$\emptyset, \emptyset,\ A',\ S_0' \circ S_1' \vdash_s S'$$

For every transition $r \in \boldsymbol{r}$, by Lemma 6 (1) the composition relation is preserved; this can be shown proceeding as in Lemma G8.

The case for $i = 1$ is symmetric (proceeds similarly, thanks to symmetric rules of composition and transition of protocols ensembles).