Featherweight Scribble

Rumyana Neykova[1] and Nobuko Yoshida[2(✉)]

[1] Brunel University London, London, UK
[2] Imperial College London, London, UK
n.yoshida@imperial.ac.uk

Abstract. This paper gives a formal definition of the protocol specification language Scribble. In collaboration with industry, Scribble has been developed as an engineering incarnation of the formal *multiparty session types*. In its ten years of development, Scribble has been applied and extended in manyfold ways as to verify and ensure correctness of concurrent and distributed systems, e.g. type checking, runtime monitoring, code generation, and synthesis. This paper introduces a core version of Scribble, *Featherweight Scribble*. We define the semantics of Scribble by translation to communicating automata and show a behavioural-preserving encoding of Scribble protocols to multiparty session type.

1 Introduction

The computational model, Klaim, introduced by De Nicola and others [8] advocates a hybrid (dynamic and static) approach for access control against *capabilities* (policies) to support static checking integrated within a dynamic access-control procedure. Their capabilities can specify crucial operations for mobile computation such as *read*, *write* and *execute* of processes in relation to the various localities, as *types*. Around the same period, (binary) *session types* [14,27] were proposed to describe a sequence of read (output), write (input) and choice operations for channel passing protocols. Later binary session types were extended to *multiparty session types* [7,15], as a model of *abstract choreographies* of Web Services Choreography Description Language [6]. See [16, §1] for more historical backgrounds.

Scribble [13,26] is a protocol description language, formally based on the multiparty session type theory. A protocol in Scribble represents an agreement on how participating systems interact with each other. It specifies a format and a predefined order for messages to be exchanged. The name of the language embodies the motivation for its creation, as explained by the following quote from the inventor of the Scribble language Kohei Honda:

> *The name (Scribble) comes from our desire to create an effective tool for architects, designers and developers alike to quickly and accurately write down protocols.*

M. Boreale et al. (Eds.): De Nicola-Festschrift, LNCS 11665, pp. 236–259, 2019.
https://doi.org/10.1007/978-3-030-21485-2_14

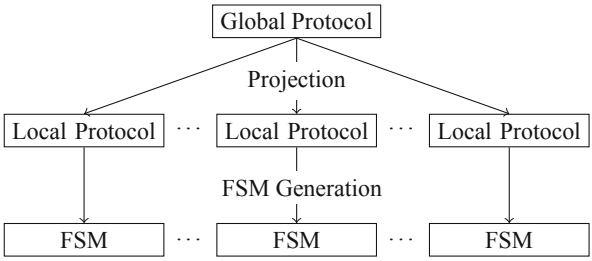

Fig. 1. Scribble development methodology

The development of Scribble is a result of a persistent dialogue between researchers and industry partners. Currently Scribble tools are applied to verification of main stream languages such as Java [18,19], Python [9,17], MPI [24], Go [5], F# [22], Erlang [23].

> *All great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling*

Although Scribble is often referred as "the practical incarnation of multiparty session types (MPST)" [13,26], a formal correspondence between the two is not proven in the literature. In this paper we present the semantics of Scribble protocols, given by translation to communicating automata, and show a behavioural-preserving encoding of Scribble protocols to multiparty session type.

Section 2 gives an overview of Scribble and explains the Scribble framework. Section 3 presents the formal semantics of the Scribble language. Section 4 proves the encoding of Scribble local and global protocols to global and local session types to be behaviour-preserving. Section 5 gives the translation between local Scribble protocols and Communicating Finite State machines (CFSMs) [10]. Section 6 concludes. Appendix contains omitted proofs.

2 Scribble Overview

Scribble protocols describe an abstract structure of message exchanges between *roles*: roles abstract from the actual identity of the endpoints that may participate in a run-time conversation instantiating the protocol. Scribble enables the description of abstract interaction structures including asynchronous message passing, choice, and recursion.

Here we demonstrate the basic Scribble constructs via an example of an online payment service. Figure 2 (left) shows the global Scribble protocol `OnlineWallet`. The first line declares, under the name `OnlineWallet`, the Scribble global protocol and the two participating roles. The protocol has a recursion at thetop-level. In

```
global protocol OnlineWallet(
  role S, role C){
  rec LOOP {
   Balance(int) from S to C;
   Overdraft(int) from S to C;
   choice at C {
    Payment(int) from C to S;
    continue LOOP;
  } or {
    CloseAccount() from C to S;
  } or {
    Quit() from C  to S;
}}}
```

```
local protocol OnlineWallet(
  self S, role C) {
  rec LOOP {
    Balance(int) to C;
    Overdraft(ints) to C;
    choice {
      Payment(int) from C;
      continue LOOP;
    } or {
      CloseAccount() from C;
    } or {
      Quit() from C;
}}}
```

Fig. 2. A global (left) and local (right) Scribble protocol

each iteration, the `Server` (`S`) sends the`Client` (`C`) the current balance and the overdraft limit for client's account. The `Balance` message has an `int` payload; similarly for the `Overdraft`. `Client` then can choose to either make a payment, close the account or quit this session.

Figure 1 gives an abstract overview of the Scribble verification process. From a global Scribble protocol, the toolchain produces (1) a set of local protocols or (2) a set of finite state machines (FSMs). We outline the tasks performed by the Scribble toolchain.

Well-Formedness Check: A global Scribble protocol is verified for correctness to ensure that the protocol is *well-formed*, which *intuitively* represents that a protocol describes a meaningful interaction, beyond the basic syntax defined by the language grammar. This well-formed checking is necessary because some of the protocols are unsafe or inconsistent even if they follow the grammar. For example, two choice branches from the same sender to the same receiver with the same message signature lead to ambiguity at the receiver side. A protocol is well-formed if local protocols can be generated for all of its roles, i.e., the projection function is defined for all roles. The formal definition of projection is given in Definition 4.10. Here we give intuition as to what the main syntactic restrictions are. First, in each branch of a choice the first interaction (possibly after a number of unfoldings) is from the same sender (e.g., `A`) and to the same set of receivers. Second, in each branch of a choice the labels are pair-wise distincs (i.e., protocols are deterministic).

Projection: A global Scribble protocol is projected to a set of local protocols. More precisely, a local Scribble protocol is generated per each role declared in the definition of the global protocol. Local protocols correspond to local (MPST) types, they describe interactions from the viewpoint of a single entity. They can be used directly by a type checker to verify that an endpoint code implementation complies to the interactions prescribed by a specification. Figure 2 (right) lists the Scribble local protocol `OnlineWallet` projected for role `Server`.

FSM Generation: An alternative representation of a local protocol can be given in the form of a communicating finite state machine (FSM). This repre-

sentation is useful for runtime verification. Specifically, at runtime the traces emitted by a program are checked against the language accepted by the FSM.

An implementation of Java and Python based Scribble tools for projection and validation [26], as well as static verification for various languages can be found in [1].

3 Syntax and Semantics of Scribble

3.1 Scribble Global Protocols

We now define the syntax of Scribble global protocols. The grammar is given below.

Definition 3.1 (Scribble Global Protocols)

$$
\begin{array}{lll}
\texttt{P} ::= \texttt{global protocol pro (role A}_1\texttt{, ..., role A}_n\texttt{)\{G\}} & & \textit{specification} \\
\texttt{G} ::= \texttt{a(S) from A to B; G} & & \textit{interaction} \\
\quad | \quad \texttt{choice at A \{G\} or ... or \{G\}} & & \textit{choice} \\
\quad | \quad \texttt{rec t \{G\}} & & \textit{recursion} \\
\quad | \quad \texttt{continue t} & & \textit{call}
\end{array}
$$

Protocol names are ranged over by `pro`. A (global) specification `P` declares a protocol with name `pro`, involving a list $(\texttt{A}_1, ..\texttt{A}_n)$ of roles, and prescribing the behaviour in `G`. The other constructs are explained below:

- An interaction `a(S) from A to B; G` specifies that a message `a(S)` should be sent from role `A` to role `B` and that the protocol should then continue as prescribed by the continuation `G`. Messages are of the form `a(S)` with `a` being a label and `S` being the constant type of exchanged messages (such as `real, bool` and `int`).
- A choice `choice at A {G} or ... or {G}` specifies a branching where role `A` chooses to engage in the interactions prescribed by one of the options `G`. The decision itself is an internal process to role `A`, i.e. how `A` decides which scenario to follow is not specified by the protocol.
- A recursion `rec t {G}` defines a scope with a name `t` and a body `G`. Any call `continue t` occurring inside `G` executes another recursion instance (if `continue t` is not in an appropriate scope than it remains idle).

Formal Semantics of Global Protocols. The formal semantics of global protocols characterises the desired/correct behaviour of the roles in a multiparty protocol. We give the semantics for Scribble protocols as a Labelled Transition System (LTS). The LTS is defined over the following set of transition labels:

$$\ell ::= \texttt{AB!a(S)} \mid \texttt{AB?a(S)}$$

Label `AB!a(S)` is for a send action where role `A` sends to role `B` a message `a(S)`. Label `AB?a(S)` is for a receive action where `B` receives (i.e., collects from the

queue associated to the appropriate channel) message a(S) that was previously sent by A. We define the subject of an action, modelling the role that has the responsibility of performing that action, as follows:

$$subj(\texttt{AB!a(S)}) = \texttt{A} \qquad subj(\texttt{AB?a(S)}) = \texttt{B}$$

As, due to asynchrony, send and receive are two distinct actions, the LTS shall also model the intermediate state where a message has been sent but it has not been yet received. To model these intermediate states we introduce the following additional global Scribble interaction:

$$\texttt{transit : a(S) from A to B; G}$$

to describe the state in which a message a(S) has been sent by A but not yet received by B. We call *runtime* global protocol a protocol obtained by extending the syntax of Scribble with these intermediate states.

The transition rules are given in Fig. 3. Rule ⌊SEND⌋ models a sending action; it produces a label AB!a(S). The sending action yields a state in which the global protocol is in an intermediate state.

Rule ⌊RECV⌋ models the dual receive action, from an intermediate state to a continuation G. Rule ⌊CHOICE⌋ continues the execution of the protocol as a continuation of one of the branches. Rule ⌊REC⌋ is standard and unfolds recursive protocols.

We explain the remaining rules with more detailed illustration.

Due to asynchrony and distribution, in a particular state of a Scribble global protocol it may be possible to trigger more than one action. For instance, the protocol in (1) allows two possible actions: AB!a(S) or CD!a(S).

$$\begin{array}{l}\texttt{a(S) from A to B;}\\ \texttt{a(S) from C to D;}\end{array} \tag{1}$$

This is due to the fact that the two send actions are not *causally related* as they have different subjects (which are independent roles). We want the semantics of Scribble to allow, in the state with protocol (1), not only the first action that occurs syntactically (e.g., AB!a(S)) but also any action that occurs later, syntactically, but it is not causally related with previous actions in the protocol (e.g., CD!a(S)). Rule ⌊ASYNC1⌋ captures this asynchronous feature. CD!a(S), which occurs syntactically later than AB!a(S) to possibly occur before. In fact, the LTS allows (1) to take one of these two actions: either AB!a(S) by rule ⌊SEND⌋ or CD!a(S) is allowed by ⌊ASYNC1⌋. Rule ⌊ASYNC2⌋ is similar to ⌊ASYNC1⌋ but caters for intermediate states, and is illustrated by the protocol in (2).

$$\begin{array}{l}\texttt{transit :a(S) from A to B;}\\ \texttt{a(S) from C to D;}\end{array} \tag{2}$$

The protocol in (2) is obtained from (1) via transition AB!a(S) by rule ⌊SEND⌋. The above protocol can execute either AB?a(S) by rule ⌊RECV⌋, or CD!a(S) by rule ⌊ASYNC2⌋.

$$\mathtt{a(S)\ from\ A\ to\ B; G} \xrightarrow{\mathtt{AB!a(S)}} \mathtt{transit : a(S)\ from\ A\ to\ B; G} \quad \lfloor \text{SEND} \rfloor$$

$$\frac{}{\mathtt{transit : a(S)\ from\ A\ to\ B; G} \xrightarrow{\mathtt{AB?a(S)}} \mathtt{G}} \qquad \frac{i \in \{1,..,n\} \quad \mathtt{G_i} \xrightarrow{\ell} \mathtt{G'_i}}{\mathtt{choice\ at\ A\ \{G_1\}\ or\ \ldots\ or\ \{G_n\}} \xrightarrow{\ell} \mathtt{G'_i}} \quad \lfloor \text{RECV} \rfloor / \lfloor \text{CHOICE} \rfloor$$

$$\frac{\mathtt{G[rec\ t\ \{G\}/continue t]} \xrightarrow{\ell} \mathtt{G'}}{\mathtt{rec\ t\ \{G\}} \xrightarrow{\ell} \mathtt{G'}} \qquad \frac{\mathtt{G} \xrightarrow{\ell} \mathtt{G'} \qquad \mathtt{A,B} \notin subj(\ell)}{\mathtt{a(S)\ from\ A\ to\ B; G} \xrightarrow{\ell} \mathtt{a(S)\ from\ A\ to\ B; G'}} \quad \lfloor \text{REC} \rfloor / \lfloor \text{ASYNC1} \rfloor$$

$$\frac{\mathtt{G} \xrightarrow{\ell} \mathtt{G'} \qquad \mathtt{B} \notin subj(\ell)}{\mathtt{transit : a(S)\ from\ A\ to\ B; G} \xrightarrow{\ell} \mathtt{transit : a(S)\ from\ A\ to\ B; G'}} \quad \lfloor \text{ASYNC2} \rfloor$$

Fig. 3. Labelled transitions for global protocols.

$$\mathtt{a(S)\ to\ B; T} \xrightarrow{\mathtt{AB!a(S)}} \mathtt{T} \quad \lfloor \text{SEND} \rfloor \qquad \mathtt{a(S)\ from\ B; T} \xrightarrow{\mathtt{BA?a(S)}} \mathtt{T} \quad \lfloor \text{RECV} \rfloor$$

$$\frac{i \in \{1,..,n\} \quad \mathtt{T_i} \xrightarrow{\ell} \mathtt{T'_i}}{\mathtt{choice\ at\ A\ \{T_1\}\ or \ldots \{T_n\}} \xrightarrow{\ell} \mathtt{T'_i}} \quad \lfloor \text{CHOICE} \rfloor \qquad \frac{\mathtt{T[rec\ t\ \{T\}/continue t]} \xrightarrow{l} \mathtt{T'}}{\mathtt{rec\ t\ \{T\}} \xrightarrow{l} \mathtt{T'}} \quad \lfloor \text{REC} \rfloor$$

Fig. 4. Labelled transitions for local protocols (from A's point of view)

$$\begin{array}{lll} G & ::= \mathtt{A} \to \mathtt{B} : \{\mathtt{a}_i\langle \mathtt{S}_i\rangle.G_i\}_{i\in I} \mid \mu \mathtt{t}.G \mid \mathtt{t} \mid \mathtt{end} & \text{global types} \\ T & ::= \mathtt{B}!\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.T'_i\}_{i\in I} \mid \mathtt{B}?\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.T'_i\}_{i\in I} \mid \mu \mathtt{t}.T \mid \mathtt{t} \mid \mathtt{end} & \text{local types} \end{array}$$

Fig. 5. Syntax for global and local types

$$\frac{j \in I}{(\mathtt{A} \to \mathtt{B} : \{\mathtt{a}_i\langle \mathtt{S}_i\rangle.G_i\}_{i\in I}) \xrightarrow{\mathtt{AB!a}_j\langle \mathtt{S}_j\rangle} (\mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}_j\langle \mathtt{S}_j\rangle.G_j)} \quad \lfloor \text{SELECT} \rfloor$$

$$\frac{(G[\mu \mathtt{t}.G/\mathtt{t}]) \xrightarrow{\ell} (G')}{(\mu \mathtt{t}.G) \xrightarrow{\ell} (G')} \qquad \frac{\forall k \in I \quad G_k \xrightarrow{\ell} G'_k \quad \mathtt{A,B} \notin subj(\ell) \quad \ell \neq t}{(\mathtt{A} \to \mathtt{B} : \{\mathtt{a}_i\langle \mathtt{S}_i\rangle.G_i\}_{i\in I}) \xrightarrow{\ell} (\mathtt{A} \to \mathtt{B} : \{\mathtt{a}_i\langle \mathtt{S}_i\rangle.G'_i\}_{i\in I})} \quad \lfloor \text{REC} \rfloor / \lfloor \text{ASYNC1} \rfloor$$

$$\frac{}{(\mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S}\rangle.G) \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} (G)} \qquad \frac{G \xrightarrow{\ell} G' \quad \mathtt{B} \notin subj(\ell)}{(\mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S}\rangle.G) \xrightarrow{\ell} (\mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S}\rangle.G')} \quad \lfloor \text{BRANCH} \rfloor / \lfloor \text{ASYNC2} \rfloor$$

Fig. 6. Labelled transitions for global types (adapted from [11])

$$\mathtt{B}!\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.T_i\}_{i\in I}) \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} T_j \ \ (j \in I) \ \lfloor \text{LSEL} \rfloor \qquad \mathtt{B}?\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.T_i\}_{i\in I}) \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} T_j \ \ (j \in I) \ \lfloor \text{LBRA} \rfloor$$

$$T[\mu \mathtt{t}.T/\mathtt{t}] \xrightarrow{\ell} T' \ \ \textit{imply} \ \ \mu \mathtt{t}.T \xrightarrow{\ell} T' \ \lfloor \text{LREC} \rfloor$$

Fig. 7. LTS for local session types (adapted from [11])

3.2 Scribble Local Protocols

Scribble local protocols describe a session from the perspective of a single participant. The syntax of Scribble local protocols is given below.

Definition 3.2 (Scribble Local Protocols)

$$\begin{array}{l} \mathtt{L} ::= \mathtt{local\ protocol\ pro\ at\ A_i(role\ A_1, ..., roleA_1)\{T\}} \\ \mathtt{T} ::= \mathtt{a(S)\ to\ B;T} \mid \mathtt{a(S)\ from\ B;T} \mid \mathtt{choice\ at\ A\ \{T_1\}\ or \ldots or\ \{T_n\}} \\ \quad \mid \quad \mathtt{rec\ pro\ \{T\}} \mid \mathtt{continue\ pro} \mid \mathtt{end} \end{array}$$

The construct $\mathtt{a(S)\ to\ B;T}$ models a send action from $\mathtt{A}$ to $\mathtt{B}$; the dual local protocol is $\mathtt{a(S)\ from\ B;T}$ that models a receive action of $\mathtt{A}$ from $\mathtt{B}$. The other protocol constructs are similar to the corresponding global protocol constructs. Recursive variables are guarded in the standard way, i.e. they only occur under a prefix. For convenience we will, sometimes, use the notation $\mathtt{choice\ at\ \{a_i(S_i)\ from\ A;T_i\}}_{i\in\{1,..,n\}}$ to denote protocols of the form $\mathtt{choice\ at\ \{a_1(S_1)\ \ from\ A;T_1\}\ or \ldots or\ \{a_n(S_n)\ from\ A;T_n\}}$ with $n > 1$, or of the form $\mathtt{a(S)\ from\ A;T}$ when $n = 1$.

Decomposing global protocols into a set of local protocols is called *projection*. Projection is a key mechanism to enable distributed enforcement of global properties. Projection preserves the interaction structures and message exchanges required for the target role to fulfil his/her part in the conversation. The formal definition of projection, for a normal (canonical) form of global protocols, is given by Definition 4.10.

Formal Semantics of Local Protocols. The LTS for local protocols is defined by the rules in Fig. 4, and uses the same labels as the global semantics in Fig. 3. The rules ⌊SEND⌋, ⌊RECV⌋, ⌊CHOICE⌋, ⌊REC⌋ are similar to the respective rules for global protocols. No rules for asynchrony are required as each participant is assumed to be single threaded.

Formal Semantics of Configurations. The LTS in Fig. 4 describes the behaviour of each single role in isolation. In the rest of this section we give the semantics of systems resulting from the composition of Scribble local protocols and communication channels. Given a set of roles $\{1, \ldots, n\}$ we define configurations $(\mathtt{T_1}, \ldots, \mathtt{T_n}, \vec{w})$ where $\vec{w} ::= \{w_{ij}\}_{i\neq j\in\{1,\ldots,n\}}$ are unidirectional, possibly empty (denoted by ϵ), unbounded FIFO queues with elements of the form $\mathtt{a(S)}$.

Definition 3.3 (Semantics of configurations). *The LTS of* $(\mathtt{T_1}, \ldots, \mathtt{T_n}, \vec{w})$ *is defined as follows:* $(\mathtt{T_1}, \ldots, \mathtt{T_n}, \vec{w}) \xrightarrow{\ell} (\mathtt{T'_1}, \ldots, \mathtt{T'_n}, \vec{w}')$ *iff: :*

(1) $\mathtt{T_B} \xrightarrow{\mathtt{AB!a(S)}} \mathtt{T'_B} \wedge w'_{\mathtt{AB}} = w_{\mathtt{AB}} \cdot \mathtt{a(S)} \wedge (ij \neq \mathtt{AB} \Rightarrow w_{ij} = w'_{ij} \wedge \mathtt{T_i} = \mathtt{T'_i})$

(2) $\mathtt{T_B} \xrightarrow{\mathtt{AB?}l\mathtt{a(S)}} \mathtt{T'_B} \wedge \mathtt{a(S)} \cdot w'_{\mathtt{AB}} = w_{\mathtt{AB}} \wedge (ij \neq \mathtt{AB} \Rightarrow w_{ij} = w'_{ij} \wedge \mathtt{T_j} = \mathtt{T'_j})$

with $\mathtt{A}, \mathtt{B}, i, j \in \{1, \ldots, n\}$.

In (1) the configuration makes a send action given that one of the participants can perform that send action. Case (1) has the effect of adding a message, that

Fig. 8. Workflow of proving soundness of the projection

is sent, to the corresponding queue. In (2) the configuration makes a receive action given that one of its participant can perform such an action and that the message being received is currently stored in the corresponding queue. Thus, (2) has the effect of removing the message received from the queue.

4 Correspondence Between Scribble and MPST

In this section we show that a trace of a global protocol corresponds exactly to a trace of its projected local protocols. Correspondence is important as it ensures that the composition of processes, each implementing some local protocol, will behave as prescribed by the original global specification. In the context of MPST, this property is known as *soundness of the projection* (Theorem 3.1, [11]) and has already been proven for global types as defined in [11]. As explained in Sect. 4.1 a translation of this result to Scribble, however, is not obvious.

Figure 8 gives a high level overview of the results presented in this section. First, we discuss the (syntactic) differences between global types and global protocols. We present a normal form for global protocols such that a Scribble global protocol in a normal form can be encoded into (MPST) global types and it preserves semantics. We then prove a similar correspondence between Scribble local protocols and (MPST) local types. The soundness of the projection of global protocols then follows from soundness of the projection of MPST global types (Theorem 3.1 from [11]).

4.1 Scribble Normal Form

We recall the syntax of global types from [11] in Fig. 5. It is very similar to the syntax of Scribble global protocols in Sect. 3 except: (1) Scribble does not cater for delegation and higher order protocols whereas global types do; and (2) the choice and interaction protocols are two separated constructs in Scribble while they are modelled as a unique construct in global types and (3) differently than MPST, Scribble allows unguarded choice. The case of (2) is a consequence of the specific focus of Scribble as a protocol design language directed at practitioners that are familiar with e.g., Java notation, who proved to find this notation friendlier [12,13,26,28]. Regarding (3) the choice construct in Scribble directly supports recursion and choice while in MPST the choice is always directly followed by an interaction. In the following section we explain that these differences are indeed syntactic and do not affect the soundness of the language.

```
choice at A {
 choice at A {
  m1 from A to B;
  } or {
  m2 from A to B;
 }
} or {
  m3 from A to B;
}
```

```
choice at A {

 m1 from A to B;
} or {
 m2 from A to B;

} or {
 m3 from A to B;
}
```

Fig. 9. Scribble protocol (left), and its `flatten` form (right)

```
choice at A {

 rec Loop{
  choice at A {
   m1 from A to B;
   continue Loop;
   } or {
   m2 from A to B;
 }


} or {
 m3 from A to B;}
```

```
choice at A {
 m1 from A to B;
 rec Loop {
  choice at A {
  m1 from A to B;
  continue Loop;
  } or {
  m2 from A to B;
 }
} or {
m2 from A to B;
} or {
m3 from A to B;}
```

Fig. 10. Scribble protocol (left), and its normal form (right)

Definition 4.1 (Scribble Normal Form (SNF))

$$\texttt{G} ::= \texttt{choice at A } \{\texttt{N}_\texttt{i}\}_{i\in\{1,..,n\}} \mid \texttt{N} \mid \texttt{rec t } \{\texttt{N}\}$$
$$\texttt{N} ::= \texttt{a(S) from A to B;G} \mid \texttt{continue t} \mid \textit{end}$$

First, we observe that a Scribble syntax with a guarded and a singleton choice directly corresponds to MPST. We refer to a Scribble protocol, where all choices are guarded, as a Scribble Normal Form (SNF). Later we show that there is a behaviour preserving translation between a well-formed Scribble protocol and its normal form. The Scribble Normal Form (SNF) for global protocols is given below.

The encoding of Scribble global protocols to SNF requires two auxiliary functions: `flatten`(G) and `unfold`(G). The latter collects top level global types from a choice type, and is utilised in the encoding as to remove nested choice. The former performs one unfolding of a recursion. We demonstrate `flatten`(G) in the example in Fig. 9.

Definition 4.2 (Flatten). *Given a Scribble protocol* G *then* `flatten`(G) *is defined as* $\texttt{flatten}(\texttt{G}_0)\cup...\cup\texttt{flatten}(\texttt{G}_n)$ *if* $\texttt{G}=\texttt{choice at A } \{\texttt{G}_\texttt{i}\}_{i\in\{1,..,n\}}$. *In all other cases,* `flatten`(G) *is homomorphic,* $\texttt{flatten}(\texttt{G}) = \texttt{G}$

Definition 4.3 (Unfold). *Given a global Scribble protocols* G *then* `unfold`(G) *is defined as* $\texttt{unfold}(\texttt{G}'[\texttt{rec t } \{\texttt{G}'\}/\texttt{continue t}]))$ *if* $\texttt{G} = \texttt{rec t } \{\texttt{G}'\}$ *and homomrhic otherwise*

Thus for any recursive type, unfold is the result of repeatedly unfolding the top level recursion until a non-recursive type constructor is reached. Unfold terminates given the assumption that recursive types are contractive, as in our case. Intuitively, a protocol is translated to a normal form after first unfolding all recursions once and then flattening nested choice. Figure 10 shows a Scribble protocol and its translation to its normal form, and the encoding is given in Definition 4.4.

Definition 4.4 (Encoding $\langle\rangle$ of Global Protocols to SNF)

$$\begin{array}{rl} \langle \mathtt{a(S)\ from\ A\ to\ B;G} \rangle & = \mathtt{a(S)\ from\ A\ to\ B;}\langle \mathtt{G} \rangle \\ \langle \mathtt{choice\ at\ A}\ \{\mathtt{G_i}\}_{i\in\{1,..,n\}} \rangle & = \mathtt{choice\ at\ A}\ \{\mathtt{flatten}(\langle \mathtt{G_i} \rangle)\}_{i\in\{1,..,n\}} \end{array}$$

$$\langle \mathtt{end} \rangle = \mathtt{end} \quad \langle \mathtt{rec\ t}\ \{\mathtt{G}\} \rangle = \mathtt{unfold(rec\ t}\ \{\langle \mathtt{G} \rangle\}) \quad \langle \mathtt{continue\ t} \rangle = \mathtt{continue\ t}$$

Trace Equivalence. The definition of trace equivalence, denoted by $\approx$ is standard. We write $\mathtt{G} \approx \mathtt{G'}$ if $TR(\mathtt{G}) = TR(\mathtt{G'})$ where $TR(\mathtt{G})$ is the set of traces obtained by reducing $\mathtt{G}$

$$TR(\mathtt{G}) = \{\, \vec{\ell} \mid \exists \mathtt{G'}, \mathtt{G} \xrightarrow{\vec{\ell}} \mathtt{G'} \}$$

We assume $\mathtt{G}$ is closed, i.e does not contain free type variables, where a type variable $\mathtt{t}$ is bound in $\mathtt{rec\ t}\ \{\mathtt{G'}\}$, and free otherwise. We extend the definition of traces for local protocols, global and local types, and we also extend $\approx$ and $\lesssim$ to local protocols, as well as global and local types, and configuration of local protocols.

Lemma 4.5. *Given a global protocol* $\mathtt{G}$ *then: (1)* $\mathtt{G} \approx \mathtt{flatten(G)}$ *(2)* $\mathtt{G} \approx \mathtt{unfold(G)}$*; and (3)* $\langle \mathtt{G'} \rangle [\mathtt{rec\ t}\ \{\langle \mathtt{G'} \rangle\}/\mathtt{continue\ t}] \approx \langle \mathtt{G'}[\mathtt{rec\ t}\ \{\mathtt{G'}\}/\mathtt{continue\ t}] \rangle$

Proposition 4.6 (SNF Translation). *Let* $\mathtt{G}$ *be a Scribble local protocol, then* $\mathtt{G} \approx \langle \mathtt{G} \rangle$.

4.2 From Global Protocols to Global Types

Definition 4.7 (Encoding of Global Protocols to Global Types). *The encoding* $[\![]\!]$ *from SNF to global types is given below:*

$$[\![\ \mathtt{a(S)\ from\ A\ to\ B;G}]\!] = \mathtt{A} \to \mathtt{B} : \{\mathtt{a}\langle \mathtt{S} \rangle.[\![\mathtt{G}]\!]\}$$

$$[\![\mathtt{choice\ at\ A}\ \{\mathtt{a}_j(\mathtt{S}_j)\ \mathtt{from\ A\ to\ B;G_j}\}_{j\in\{1\cdots n\}}]\!] = \mathtt{A} \to \mathtt{B} : \{\mathtt{a}_j\langle \mathtt{S}_j \rangle.[\![\mathtt{G_j}]\!]\}_{j\in\{1\cdots n\}}$$

$$[\![\mathtt{rec\ t}\ \{\mathtt{G}\}]\!] = \mu \mathtt{t}.[\![\mathtt{G}]\!] \quad [\![\mathtt{continue\ t}]\!] = \mathtt{t} \quad [\![\mathtt{end}]\!] = \mathtt{end}$$

For convenience, we recall the semantics of global types in Fig. 6. The semantics of global protocols and global types are similar except that the one for MPSTs from [11] have no rule $\lfloor$Choice$\rfloor$ as choice is handled directly in the rule for send/selection and branch/receive. To match Scribble global protocols and MPST step by step we extend the definition of encoding to account for intermediate steps:

$$[\![\mathtt{transit}:\mathtt{a(S)\ from\ A\ to\ B;G''}]\!] = \mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S} \rangle.[\![\mathtt{G''}]\!]$$

Proposition 4.8 (Correspondence of Global Protocols and Global Types). *Let* $\mathtt{G}$ *be a Scribble global protocol, then* $\mathtt{G} \approx [\![\mathtt{G}]\!]$.

4.3 From Local Protocols to Local Types

The syntactic differences between Scribble local protocols and local types (given in Fig. 5) reflect the difference between Scribble global protocols and MPST global types. We define an encoding of local protocols (Definition 3.2) to local types on the normal form of a Scribble local protocol (Definition 4.9).

Definition 4.9 (Local Scribble Normal Form (LSNF))

$$\mathtt{T} ::= \mathtt{choice\ at\ A}\{\mathtt{N_i}\}_{i\in I} \mid \mathtt{N} \mid \mathtt{rec\ t}\ \{\mathtt{N}\}$$
$$\mathtt{N} ::= \mathtt{a(S)\ from\ B;T} \mid \mathtt{a(S)\ to\ B;T} \mid \mathtt{continue\ t} \mid \mathtt{end}$$

Local types are generated from global types following a syntactic procedure, called *projection*. In a similar way we define projection on global protocols. The definition of projection is given in Definition 4.10. We denote by $\mathscr{P}(\mathtt{G})$ the set of roles in a protocol $\mathtt{G}$.

Definition 4.10 (Projection). *The projection of* $\mathtt{G}$ *onto* $\mathtt{A} \in \mathscr{P}(\mathtt{G})$*, written* $\mathtt{G}\downarrow_{\mathtt{A}}$*, is defined by induction on* $\mathtt{G}$ *as follows:*

$$(\mathtt{a(S)\ from\ B\ to\ C;G'})\downarrow_{\mathtt{A}} = \begin{cases} \mathtt{a(S)\ from\ B;}(\mathtt{G'}\downarrow_{\mathtt{A}}) & \text{if } \mathtt{A}=\mathtt{C} \\ \mathtt{a(S)\ to\ C;}(\mathtt{G'}\downarrow_{\mathtt{A}}) & \text{if } \mathtt{A}=\mathtt{B} \\ \mathtt{G'}\downarrow_{\mathtt{A}} & \text{if } \mathtt{A}\neq\mathtt{B},\mathtt{C} \end{cases}$$

$$(\mathtt{rec\ t}\ \{\mathtt{G'}\})\downarrow_{\mathtt{A}} = \begin{cases} \mathtt{rec\ t}\ \{(\mathtt{G'}\downarrow_{\mathtt{A}})\} & \mathtt{G'} \neq \mathtt{continue\ t} \\ \mathtt{end} & \textit{otherwise} \end{cases}$$

$$(\mathtt{choice\ at\ B}\ \{\mathtt{a_i(S_i)\ from\ B\ to\ C;G_i}\}_{i\in I})\downarrow_{\mathtt{A}} = \begin{cases} \mathtt{choice\ at\ B}\ \{\mathtt{a_i(S_i)\ from\ B;}(\mathtt{G_i}\downarrow_{\mathtt{A}})\}_{i\in I} & \text{if } \mathtt{A}=\mathtt{C} \\ \mathtt{choice\ at\ B}\ \{\mathtt{a_i(S_i)\ to\ C;}(\mathtt{G_i}\downarrow_{\mathtt{A}})\}_{i\in I} & \text{if } \mathtt{A}=\mathtt{B} \\ \mathtt{choice\ at\ D}\ (\sqcup\{(\mathtt{G_i}\downarrow_{\mathtt{A}})\}_{i\in I}) & \text{if } \mathtt{A}\neq\mathtt{B},\mathtt{C};\ \mathtt{G_i}\downarrow_{\mathtt{A}} = \mathtt{a_i(S_i)\ from\ D;G'_i}\downarrow_{\mathtt{A}}, \forall i\in I \end{cases}$$

$$(\mathtt{continue\ t})\downarrow_{\mathtt{A}} = \mathtt{continue\ t} \qquad (\mathtt{end})\downarrow_{\mathtt{A}} = \mathtt{end}$$

If no side condition applies then $\mathtt{G}$ is *not projectable* on $\mathtt{A}$ and the global protocol $\mathtt{G}$ is not well-formed. The case for `choice` uses the merge operator $\sqcup$ to ensure that (1) the locally projected behaviour is independent of the chosen branch (i.e $\mathtt{G_i} = \mathtt{G_j}$, for all $i, j \in I$), or (2) the chosen branch is identifiable by $\mathtt{A}$ via a unique label. The merge operator $\sqcup$ [11] is defined as a partial commutative operator over two types s.t.

$$\{\mathtt{a_i(S_i)\ from\ B;T_i}\}_{i\in I} \sqcup \{\mathtt{a'_j(S'_j)\ from\ B;T'_j}\}_{j\in J} = \{\mathtt{a_k(S_k)\ from\ B;T_k}\}_{k\in I\setminus J}$$
$$\cup\ \{\mathtt{a'_j(S'_j)\ from\ B;T'_j}\}_{j\in J\setminus I} \cup\ \{\mathtt{a_k(S_k)\ from\ B;T_k} \sqcup \mathtt{T'_k}\}_{k\in I\cap J}$$

where for each $k \in I \cap J$, $\mathtt{a_k} = \mathtt{a'_k}, \mathtt{S}_k = \mathtt{S}'_k$ Merge is homomorphic for all other types (i.e $\mathscr{E}[\mathtt{T_k}] \sqcup \mathscr{E}[\mathtt{T'_k}] = \mathscr{E}[\mathtt{T_k} \sqcup \mathtt{T'_k}]$, where $\mathscr{E}$ is a context for local protocols.). We say that $\mathtt{G}$ is *well-formed* if for all $\mathtt{A} \in \mathscr{P}(\mathtt{G})$, $\mathtt{G}\downarrow_{\mathtt{A}}$ is defined. Note that a normal form is preserved during projection, i.e a Sribble global protocol in a normal form is projected to a Scribble local protocol in a normal form. Next we give the encoding between Scribble local protocols and MPST local types. Hereafter we write Scribble local protocol when referring to LSNF protocols.

Definition 4.11 (Encoding of Local Protocols to Local Types). *The encoding* $(\!|\ |\!)$ *from (Scribble) local protocols to MPST local types is given below:*

$$(\!|\mathtt{a(S)\ to\ B;T}|\!) = \mathtt{B!\{a : \langle S\rangle.(\!|T|\!)\}} \quad (\!|\mathtt{a(S)\ from\ B;T}|\!) = \mathtt{B?\{a : \langle S\rangle.(\!|T|\!)\}}$$

$$(\!|\mathtt{choice\ at\ A\ \{T'_i\}_{i\in I}}|\!) = \begin{cases} \mathtt{B!\{a_i : \langle S_i\rangle.(\!|T'_i|\!)\}_{i\in I}} & \textit{if } \mathtt{T'_i = a_i(S_i)\ to\ B;T_i} \\ \mathtt{A?\{a_i : \langle S_i\rangle.(\!|T'_i|\!)\}_{i\in I}} & \textit{if } \mathtt{T'_i = a_i(S_i)\ from\ A;T_i} \end{cases}$$

$$(\!|\mathtt{rec\ t\ \{T\}}|\!) = \mu\mathtt{t}.(\!|\mathtt{T}|\!) \qquad (\!|\mathtt{continue\ t}|\!) = \mathtt{t} \qquad (\!|\mathtt{end}|\!) = \mathtt{end}$$

Proposition 4.12 (Correspondence of Local Protocols and Local Types). *Let* $\mathtt{T}$ *be a Scribble local protocol, then* $\mathtt{T} \approx (\!|\mathtt{T}|\!)$.

Proposition 4.13 (Correspondence of Configurations). *Let* $(\mathtt{T_1}, \ldots, \mathtt{T_n}, \vec{w})$ *be a configuration of Scribble local protocols, then* $(\mathtt{T'_1}, \ldots, \mathtt{T'_n}, \vec{w}) \approx ((\!|\mathtt{T_1}'|\!), \ldots, (\!|\mathtt{T'_n}|\!), \vec{w'})$.

4.4 Correspondence of Global and Local Protocols

Theorem 4.14 gives the correspondence between the traces produced by a global protocol $\mathtt{G}$ and those produced by the configuration that consists of the composition of the projections of $\mathtt{G}$ onto $\mathscr{P}(\mathtt{G})$.

Theorem 4.14 (Soundness of projection). *Let* $\mathtt{G}$ *be a Scribble global protocol and* $\{\mathtt{T_1}, \ldots, \mathtt{T_n}\} = \{\mathtt{G} \downarrow_{\mathtt{A}}\}_{\mathtt{A}\in\mathscr{P}(\mathtt{G})}$ *be the set of its projections, then*

$$\mathtt{G} \approx (\mathtt{T_1}, \ldots, \mathtt{T_n}, \vec{\varepsilon})$$

Theorem 4.14 directly follows by: (i) the correspondence between (Scribble) global protocols and MPSTs global types given in Sect. 4; (ii) trace equivalence between global types and configuration of projected global types (Theorem 3.1 in [11]); (iii) the correspondence between configurations of MPSTs local types and configurations of Scribble local protocols given in Sect. 4.

5 From Scribble to CFSMs

This section gives the translation of local protocols to CFSMs [4]. First, we start from some preliminary notations. ϵ is the empty word. A is a finite alphabet and A^* is the set of all finite words over A. $|x|$ is the length of a word x and $x.y$ or xy the concatenation of two words x and y. Let $\mathscr{P}$ be a set of participants fixed throughout the section: $\mathscr{P} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots \mathtt{p}, \mathtt{q}, \ldots\}$.

Definition 5.1 (CFSM). *A communicating finite state machine is a finite transition system given by a 5-tuple* $M = (Q, C, q_0, A, \delta)$ *where (1)* Q *is a finite set of states; (2)* $C = \{\mathtt{AB} \in \mathscr{P}^2 | \mathtt{A} \neq \mathtt{B}\}$ *is a set of channels; (3)* $q_0 \in Q$ *is an initial state; (4)* A *is a finite alphabet of message labels, and (5)* $\delta = Q \times (C \times \{!, ?\} \times A) \times Q$ *is a finite set of transitions.*

Final State is a state $q \in Q$, which does not have any outgoing transitions. If all states in Q are final, δ is the empty relation. *A (communicating) system* S is a tuple $S = (M_p)_{p \in \mathscr{P}}$ of CFSMs such that $M_p = (Q_p, C, q_{0_p}, A, \delta_p)$. We define *a configuration* for M_p to be a tuple $s = (\vec{q}, \vec{w})$ where $\vec{q} = (q_p)_{p \in \mathscr{P}}$ and where $w = (w_{\mathtt{pq}})_{p \neq q \in \mathscr{P}}$ with $w_{pq} \in A^*$. *A path* in M is a finite sequence of $q_0, \ldots, q_n (n \geq 0)$ such that $(q_i, \ell, q_{i+1}) \in \delta (0 \leq i \leq n-1)$ and we write $q \xrightarrow{\ell} q'$ if $(q, \ell, q') \in \delta$.

Definition 5.2 gives the translation of local Scribble protocols to CFSMs. For convenience, we do not separate a label from a payload and we write `msg` instead of `a(S)`. Without loss of generality we assume all nested recursive types are given as $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}\}$, where $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}\} = \mathtt{T}$ if $|\vec{\mathtt{t}}| = 0$. If $\vec{\mathtt{t}} = (\mathtt{t}_0, \ldots, \mathtt{t}_n)$, $\mathtt{T}' \neq \mathtt{rec}\ \mathtt{t}\ \{\mathtt{T}''\}$, then $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}\} = \mathtt{rec}\ \mathtt{t}_0\{\ldots \mathtt{rec}\ \mathtt{t}_n\ \{\mathtt{T}'\}\ldots\}$.

We use the auxiliary function `body(T)` to denote the body of a recursive term. Hence $\mathtt{body}(\mathtt{T}) = \mathtt{T}'$ `if` $\mathtt{T} = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'\}$; in all other cases $\mathtt{body}(\mathtt{T}) = \mathtt{T}$. We remind that recursive variables are guarded in the standard way, i.e. they only occur under a prefix and therefore `body(T)` cannot be `continue t`.

Definition 5.2 (Translation from local types to CFSMs). *We write* $\mathtt{T}' \in \mathtt{T}$ *if* $\mathtt{T}'$ *occurs in* $\mathtt{T}$. *Let* $\mathtt{T}_0$ *be the normal form of the local type of participant* `A` *projected from* `G`. *The automaton corresponding to* $\mathtt{T}_0$ *is* $\mathcal{A}(\mathtt{T}_0) = (Q, C, q_0, A, \delta)$ *where: (1)* $Q = \{\mathtt{T}' | \mathtt{T}' \in \mathtt{T}_0, \mathtt{T}' \neq \mathtt{continue}\ \mathtt{t}\} \setminus (\{\mathtt{T}' | \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'\} \in \mathtt{T}_0\} \cup \{\mathtt{T}_i | \mathtt{choice\ at\ A}\ \{\mathtt{T}_i\}_{i \in I} \in \mathtt{T}_0\})$; *(2)* $q_0 = \mathtt{T}_0$ *(3)* $C = \{\mathtt{AB} \mid \mathtt{A}, \mathtt{B} \in \mathtt{G}\}$; *(4)* $A = \{\mathtt{msg} \mid \mathtt{msg}$ *occurs in* $\mathtt{G}\}$ *is the set of labels* `msg` *in* `G`; *and (5)* δ *is defined below:*

1. *if* $\mathtt{body}(\mathtt{T}) = \mathtt{msg\ to\ B}; \mathtt{T}' \in Q$, *then*

$$\begin{cases} 1)(\mathtt{T}, (\mathtt{AB}!\mathtt{msg}), \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\}) \in \delta & \text{if } \mathtt{T}' = \mathtt{continue\ t}, \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\} \in \mathtt{T}_0 \\ 2)(\mathtt{T}, (\mathtt{AB}!\mathtt{msg}), \mathtt{T}') \in \delta & \text{otherwise} \end{cases}$$

2. *if* $\mathtt{body}(\mathtt{T}) = \mathtt{msg\ from\ B}; \mathtt{T}' \in Q$, *then*

$$\begin{cases} 1)(\mathtt{T}, (\mathtt{BA}?\mathtt{msg}), \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\}) \in \delta & \text{if } \mathtt{T}' = \mathtt{continue\ t}, \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\} \in \mathtt{T}_0 \\ 2)(\mathtt{T}, (\mathtt{BA}?\mathtt{msg}), \mathtt{T}') \in \delta & \text{otherwise} \end{cases}$$

3. *if* $\mathtt{T} = \mathtt{choice\ at\ A}\ \{\mathtt{T}_i\}_{i \in I}$, *then:*
   (a) *if* $\mathtt{T}_i = \mathtt{msg}_i\ \mathtt{to\ B}; \mathtt{T}'$

$$\begin{cases} 1)\mathtt{T}, (\mathtt{AB}!\mathtt{msg}_i), \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\}) \in \delta & \text{if } \mathtt{T}' = \mathtt{continue\ t}, \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\} \in Q, \\ 2)\mathtt{T}, (\mathtt{AB}!\mathtt{msg}_i), \mathtt{T}') \in \delta & \text{otherwise} \end{cases}$$

   (b) *if* $\mathtt{T}_i = \mathtt{msg}_i\ \mathtt{from\ A}; \mathtt{T}'$

$$\begin{cases} 1)(\mathtt{T}, (\mathtt{BA}?\mathtt{msg}_i), \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\}) \in \delta & \text{if } \mathtt{T}' = \mathtt{continue\ t}, \underset{\mathtt{t} \in \vec{\mathtt{t}}}{\mathtt{rec}\ \vec{\mathtt{t}}}\ \{\mathtt{T}''\} \in Q \\ 2)(\mathtt{T}, (\mathtt{BA}?\mathtt{msg}_i), \mathtt{T}') \in \delta & \text{otherwise} \end{cases}$$

```
rec t1 {
  m1 to B;
  m2 to B;
  continue t1;
}
```

Fig. 11. Scribble protocol (left) and corresponding CFSM (right)

```
rec t1 { m1 to B;
rec t2 { m2 to B;
  choice at A {
   m3 to B;
   continue t2;
  } or {
   m4 to B;
   continue t1}}}
```

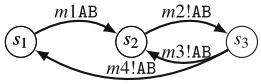

Fig. 12. Scribble protocol (left) and corresponding CFSM (right)

Examples. We illustrate the translation with two examples, in Figs. 11 and 12. The CFSM $\mathscr{A}(\mathtt{T})$ for the local protocol T from Fig. 11 (left) is $\mathscr{A}(\mathtt{T}) = (Q, C, q_0, A, \delta)$. We first generate the states Q of $\mathscr{A}(\mathtt{T})$ from the suboccurrences of the initial local protocol T. The states are denotes as s_1 and s_2 where $s_1 = \mathtt{T}$ and $s_2 = (\mathtt{m2\ to\ B;\ continue\ t1;})$. $\mathscr{A}(\mathtt{T})$ is defined as the 5-tuple: 1) $Q = \{s_1, s_2\}$; 2) $C = \{\mathtt{AB}, \mathtt{BA}\}$; 3) $q_0 = s_1$; 4) $A = \{\mathtt{m1,\ m2}\}$; 5) $\delta = \{(s_1, \mathtt{m1!AB}, s_2), (s_2, \mathtt{m2!AB}, s_1)\}$.

Next we consider the local type T, given on Fig. 12 (left). From the suboccurrences of the local protocol T we generate three states s_1, s_2, and s_3, where $s_1 = \mathtt{T}$; $s_2 = \mathtt{rec\ t2\ \ m2\ to\ B;\ choice\ at\ A\ m3\ to\ B;\ continue\ t2\ or\ m4\ to\ B;\ continue\ t1}$; and $s_3 = \mathtt{choice\ at\ A\ m3\ to\ B;\ continue\ t2\ or\ m4\ to\ B;\ continue\ t1}$. Then the corresponding automaton $\mathscr{A}(\mathtt{T})$ is the 5-tuple (Q, C, A, q_0, δ) where 1) $Q = \{s_1, s_2, s_3\}$; 2) $C = \{\mathtt{AB}, \mathtt{BA}\}$; 3) $q_0 = s_1$ 4) $A = \{\mathtt{m1,\ m2,\ m3,\ m4}\}$; 5) $\delta = \{(s_1, \mathtt{m1!AB}, s_2), (s_2, \mathtt{m2!AB}, s_3), (s_3, \mathtt{m3!AB}, s_2), (s_3, \mathtt{m4!AB}, s_1)\}$.

We proceed by proving operational correspondence between a local type T and its corresponding $\mathscr{A}(\mathtt{T})$. We use an auxiliary function to map recursive variables to types.

Definition 5.3 (Unfold mapping). *We define a function* $\mathtt{unfMap} : \mathtt{T} \times \sigma \to \sigma$*, where* T *is a type and* σ *is a mapping from recursive variables to types* $\mathtt{unfMap}(\mathtt{T}, \sigma) =$

$$\begin{cases} \bigcup_{i \in I} \mathtt{unfMap}(\mathtt{T_i}, \sigma) \ \mathtt{if\ T} = \mathtt{choice\ at\ A}\ \{\mathtt{T_i}\}_{i \in I} \\ \mathtt{unfMap}(\mathtt{T'}, \sigma) \ \mathtt{if\ either\ T} = \mathit{msg}\ \mathtt{to\ B;T'},\ \mathtt{or\ T} = \mathit{msg}\ \mathtt{from\ B;T'} \\ \mathtt{unfMap}(\mathtt{T'}[\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T'}\}/\mathtt{continue\ t}]_{\forall \mathtt{t} \in \vec{\mathtt{t}}}, \sigma \bigcup_{\mathtt{t}_i \in \vec{\mathtt{t}}} \{\mathtt{t}_i \mapsto \mathtt{T}\}) \\ \qquad \mathtt{if\ T} = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T'}\};\ \mathtt{t}_i \notin \sigma \\ \sigma \ \mathtt{if\ either\ T} = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T'}\}\ \mathtt{and}\ \exists \mathtt{t'} \in \vec{\mathtt{t}} : \mathtt{t'} \in \sigma,\ \mathtt{or\ T} = \mathtt{end} \end{cases}$$

We assume all recursive variables are distinct and also $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T'}\}\sigma = \mathtt{T'}\sigma$. Hence, σ can contain $\mathtt{t} \in \vec{\mathtt{t}}$ and we apply the substitution σ without α-renaming.

Lemma 5.4 (Suboccurrences). *Given a local protocol* T*, with a suboccurrence* $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T'}\}_{(t \in \vec{t})} \in T$ *and a substitution* σ *s.t* $\sigma = \mathtt{unfMap}((\mathtt{T}, \varnothing))$*,then*

$$\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T'}\}_{(t \in \vec{t})}\sigma = \mathtt{T''}\ \mathtt{with}\ \{t \mapsto \mathtt{T''}\} \in \sigma$$

Theorem 5.5 (Soundness of translation). *Given a local protocol* T*, then* $\mathtt{T} \approx \mathscr{A}(\mathtt{T})$.

6 Conclusion and Related Work

De Nicola is the first person who proposed a location-based distributed model with rich capability types, and *implemented* that model in Java as to demonstrate a practical use of formal foundations for mobile computing. Following his spirits, this paper gave a formal definition of a practical protocol description language, Scribble. We proved a correspondence between Scribble and MPST and showed that a global protocol corresponds to a system of CFSMs.

The work [10] is the first to explore the connection of MPST and CFSMs. [10] gives a sound and complete characterisation of a class of communicating automata that can be expressed by the language of multiparty session types. The presented work is closely based on the translation of session types to CFSMs presented in [10], and hence we adhere to the same conditions as theirs, namely the CFSM is deterministic and directed without mixed states (each state is either sending or receiving to the same participant with distinct labels). Lange et al. [20] presents an algotithm for synthesising *global graphs* from local multiparty specifications, given as CFSMs, that allows more general constructs, such as fork and join. Scribble currently does not support such constructs. The correspondence of MPST and CFSMs with time constraints is further explored in [2,3]. The work [21] uses the result in [3] to implement a runtime monitor based on an extension of Scribble with time annotations, but the work does not prove formal correspondence between timed Scribble and timed automata.

The encoding of Scribble protocols to CFSMs presented in this article is an important basis when building and verifying distributed systems. It guarantees that global safety properties can be ensured through local, i.e, decentralised verification. The setting defined by CFSMs does not require synchronisation at runtime. Therefore our approach is more efficient to implement than a centralised approach. In [9,17], we rely on this result to design and build a sound Scribble-based framework for runtime and hybrid verification.

Several implementation works use the Scribble toolchian and the local CFSM representation to generate APIs for programming distributed protocols [18,22]. In recent years, Scribble-based code generation has been extended with various contructs, e.g. parameterised role [5] for distributed Go programming, delegation in Scala [25], time constarints in Python [3], explicit connections [19] for dynamic joining of roles in Java, and payload constraints in F# [22]. The above mentioned works are either practical (hence no formal semantics nor operational correspondence results are given) and/or informally rely on the correspondence between MPST and Scribble as to justify the soundness of their respective implementations and extensions.

Future work includes formalisations of extended Scribble in the literature explained above. In particular, there exists no operational semantics of multiple multiparty session types with *delegations* and *higher-order code mobility* since a single system of CFSMs corresponds to a single multiparty session type with fixed participants. We plan to tackle this problem first extending CFSM models from a fixed set to a family of participants.

Acknowledgments. We thank the reviewers for their comments. This work is partially supported by EPSRC projects EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1. The first author was supported by an EPSRC Doctoral Prize Fellowship.

A Scribble Normal Form

Proposition 4.6 (SNF Translation): Let $\mathtt{G}$ be a Scribble local protocol, then $\mathtt{G} \approx \langle \mathtt{G} \rangle$.

Proof. First we consider $\mathtt{G} \lesssim \langle \mathtt{G} \rangle$. The proof is mechanical and is done by induction on the transition rules applied for closed terms of $\mathtt{G}$.

1. (base case) If $\mathtt{G} = \mathtt{end}$ then both $trmG$ and $\langle \mathtt{G} \rangle$ produce an empty set of traces and no rules can be applied.
2. (inductive case) if $\mathtt{G} \xrightarrow{\ell} \mathtt{G}'$ then $\langle \mathtt{G} \rangle \xrightarrow{\ell} \mathtt{G}''$ such that $\mathtt{G}' \approx \mathtt{G}''$.
   (a) if $\mathtt{G} = \mathtt{a(S)\ from\ A\ to\ B}; \mathtt{G}'$
   $\mathtt{G}$ can do $\mathtt{AB!msg}$ or ℓ by $\lfloor$SEND$\rfloor$ or $\lfloor$ASYNC1$\rfloor$ respectively.
   Then $\mathtt{G} \lesssim \langle \mathtt{G} \rangle$ follows by the induction hypothesis (IH) and by the definition of encoding
   (b) $\mathtt{G} = \mathtt{rec\ t}\ \{\mathtt{G}'\}$
   $\mathtt{G} \xrightarrow{\ell} \mathtt{G}''$
   By $\lfloor$REC$\rfloor$ $\mathtt{G}'[\mathtt{rec\ t}\ \{\mathtt{G}'\}/\mathtt{continue\ t}] \xrightarrow{\ell} \mathtt{G}''$
   By IH $\langle \mathtt{G}'[\mathtt{rec\ t}\ \{\mathtt{G}'\}/\mathtt{continue\ t}] \rangle \xrightarrow{\ell} \mathtt{G}'''$ s.t $\mathtt{G}'' \approx \mathtt{G}'''$
   By Lemma 4.5
   $\langle \mathtt{G}' \rangle [\mathtt{rec\ t}\ \{\langle \mathtt{G}' \rangle\}/\mathtt{continue\ t}] \approx \langle \mathtt{G}'[\mathtt{rec\ t}\ \{\mathtt{G}'\}/\mathtt{continue\ t}] \rangle$
   Thus, $\langle \mathtt{G}' \rangle [\mathtt{rec\ t}\ \{\langle \mathtt{G}' \rangle\}/\mathtt{continue\ t}] \xrightarrow{\ell} \mathtt{G}''''$ s.t $\mathtt{G}''' \approx \mathtt{G}''''$
   By $\lfloor$REC$\rfloor$$\mathtt{rec\ t}\ \{\langle \mathtt{G}' \rangle\} \xrightarrow{\ell} \mathtt{G}''''$
   By Lemma 4.5 $\mathtt{rec\ t}\ \{\langle \mathtt{G}' \rangle\} \approx \mathtt{unfold}(\mathtt{rec\ t}\ \{\langle \mathtt{G}' \rangle\}) = \langle \mathtt{G} \rangle$
   (c) $\mathtt{G} = \mathtt{choice\ at\ A}\ \{\mathtt{G}_i\}_{i \in \{1,..,n\}}$
   From $\lfloor$CHOICE$\rfloor$$\mathtt{G} \xrightarrow{\ell} \mathtt{G}'$ with $\mathtt{G}_i \xrightarrow{\ell} \mathtt{G}'$
   From IH $\langle \mathtt{G}_i \rangle \xrightarrow{\ell} \mathtt{G}''$ s.t $\mathtt{G}'' \approx \mathtt{G}'$ From $\mathtt{flatten}(\mathtt{G}) \approx \mathtt{G}$ it follows that $\mathtt{flatten}(\mathtt{G}_i) \xrightarrow{\ell} \mathtt{G}'''$ s.t $\mathtt{G}''' \approx \mathtt{G}''$
   From $\lfloor$CHOICE$\rfloor$ it follows $\langle \mathtt{G} \rangle \xrightarrow{\ell} \mathtt{G}'''$

Now we consider $\langle \mathtt{G} \rangle \lesssim \mathtt{G}$. The proof is by induction on the definition of encoding of closed terms of $\mathtt{G}$.

1. (base case) If $\langle \mathtt{G} \rangle = \mathtt{end}$ then both $\mathtt{G}$ and $\langle \mathtt{G} \rangle$ produce an empty set of traces and no rules can be applied.
2. (inductive case) if $\langle \mathtt{G} \rangle \xrightarrow{\ell} \mathtt{G}'$ then $\mathtt{G} \xrightarrow{\ell} \mathtt{G}''$ such that $\mathtt{G}' \approx \mathtt{G}''$.
   (a) $\langle \mathtt{a(S)\ from\ A\ to\ B}; \mathtt{G} \rangle = \mathtt{a(S)\ from\ A\ to\ B}; \langle \mathtt{G} \rangle$
   $\langle \mathtt{G} \rangle$ can do $\mathtt{AB!msg}$ or ℓ by $\lfloor$SEND$\rfloor$ or $\lfloor$ASYNC1$\rfloor$ respectively.
   Then $\mathtt{G} \lesssim \langle \mathtt{G} \rangle$ follows by the IH and by the definition of encoding

(b) $\langle \texttt{rec t } \{\texttt{G}\}\rangle = \texttt{unfold}(\texttt{rec t } \{\langle \texttt{G}\rangle\}) = \langle \texttt{G}\rangle[\texttt{rec t } \{\langle \texttt{G}\rangle\}/\texttt{continue t}]$
From IH: $\texttt{G}[\texttt{rec t } \{\texttt{G}\}/\texttt{continue t}] \approx \langle \texttt{G}\rangle[\texttt{rec t } \{\langle \texttt{G}\rangle\}/\texttt{continue t}]$
Thus, if $\langle \texttt{G}\rangle[\texttt{rec t } \{\langle \texttt{G}\rangle\}/\texttt{continue t}] \xrightarrow{\ell} \texttt{G}'$
then $\texttt{G}[\texttt{rec t } \{\texttt{G}\}/\texttt{continue t}] \xrightarrow{\ell} \texttt{G}''$ s.t $\texttt{G}' \approx \texttt{G}''$
From $\lfloor$REC$\rfloor$ rule: $\texttt{rec t } \{\texttt{G}\} \xrightarrow{\ell} \texttt{G}''$

(c) $\langle \texttt{G}\rangle = \langle \texttt{choice at A } \{\texttt{G}_\texttt{i}\}_{i\in\{1,..,n\}}\rangle = \texttt{choice at A } \{\texttt{flatten}(\langle \texttt{G}_\texttt{i}\rangle)\}_{i\in\{1,..,n\}}$
From $\lfloor$CHOICE$\rfloor\langle \texttt{G}\rangle \xrightarrow{\ell} \texttt{G}'$ with $\texttt{flatten}(\langle \texttt{G}_\texttt{i}\rangle) \xrightarrow{\ell} \texttt{G}'$
By Lemma 4.5 $\texttt{flatten}(\langle \texttt{G}\rangle) \approx \langle \texttt{G}\rangle$ it follows that
$\langle \texttt{G}_\texttt{i}\rangle \xrightarrow{\ell} \texttt{G}''$ s.t $\texttt{G}'' \approx \texttt{G}'$
From IH it follows that $\texttt{G} \xrightarrow{\ell} \texttt{G}'''$ s.t $\texttt{G}''' \approx \texttt{G}'' \approx \texttt{G}'$.

B From Global Protocols to Global Types

Proposition 4.8 (Correspondence of Global Protocols and Global Types): Let G be a Scribble global protocol, then $\texttt{G} \approx [\![\texttt{G}]\!]$.

Proof. First, we consider $\texttt{G} \lesssim [\![\texttt{G}]\!]$. The proof is done by induction (on the depth of the tree) on the transition rule applied.

1. (Base case) If $\texttt{G} = \texttt{end}$ then both G and $[\![\texttt{G}]\!]$ produce an empty set of traces.
2. (Inductive case) if $\texttt{G} \xrightarrow{\ell} \texttt{G}'$ and we have to prove that $[\![\texttt{G}]\!] \xrightarrow{\ell} [\![\texttt{G}']\!]$.

– if $\texttt{G} = \texttt{a(S) from A to B; G}''$ then we either have a send action by $\lfloor$SEND$\rfloor$ or ℓ transition by $\lfloor$ASYNC1$\rfloor$
  - $\lfloor$SEND$\rfloor$ $\texttt{G} \xrightarrow{\texttt{AB!a}\langle \texttt{S}\rangle} \texttt{transit :a(S) from A to B; G}''$
    By (1) $[\![\texttt{G}]\!] = \texttt{A} \to \texttt{B} : \{\texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!]\}$ and
    (2) $[\![\texttt{transit :a(S) from A to B; G}'']\!] = \texttt{A} \rightsquigarrow \texttt{B} : \texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!]$ and
    (3) $\lfloor$SELECT$\rfloor_{MPST}$: $\texttt{A} \to \texttt{B} : \{\texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!]\} \xrightarrow{\texttt{AB!a}\langle \texttt{S}\rangle} \texttt{A} \rightsquigarrow \texttt{B} : \texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!]$
    we have $[\![\texttt{G}]\!] \xrightarrow{\texttt{AB!a}\langle \texttt{S}\rangle} [\![\texttt{G}']\!]$
  - $\lfloor$ASYNC1$\rfloor\texttt{a(S) from A to B; G}'' \xrightarrow{\ell} \texttt{a(S) from A to B; G}''$ By (1) $[\![\texttt{G}]\!] = \texttt{A} \to \texttt{B} : \{\texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!]\}$ and $[\![\texttt{G}']\!] = \texttt{A} \to \texttt{B} : \texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}''']\!]$ By (2) $[\![\texttt{G}'']\!] \xrightarrow{\ell} [\![\texttt{G}''']\!]$, which follows from the premise $\texttt{G}'' \xrightarrow{\ell} \texttt{G}'''$ of the $\lfloor$ASYNC1$\rfloor$ and by IH and (3) $\texttt{B} \notin subj(\ell)$, which follows from the premise of $\lfloor$ASYNC1$\rfloor$:
    we can apply the $\lfloor$ASYNC1$\rfloor_{MPST}$ rule: $[\![\texttt{G}]\!] \xrightarrow{\ell} [\![\texttt{G}']\!]$

– if $\texttt{G} = \texttt{transit :a(S) from A to B; G}''$
We proceed as in the above case. We either have a receive action by the rule $\lfloor$RECV$\rfloor$ or ℓ transition by the rule $\lfloor$ASYNC2$\rfloor$.
  - $\lfloor$RECV$\rfloor\texttt{G} \xrightarrow{\texttt{AB?a}\langle \texttt{S}\rangle} \texttt{G}'$ where $\texttt{G}' = \texttt{G}''$
    By (1) $[\![\texttt{G}]\!] = \texttt{A} \rightsquigarrow \texttt{B} : \texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!]$ and $[\![\texttt{G}']\!] = [\![\texttt{G}'']\!]$ and
    (3) $\lfloor$BRANCH$\rfloor_{MPST}$: $\texttt{A} \rightsquigarrow \texttt{B} : \texttt{a}\langle \texttt{S}\rangle.[\![\texttt{G}'']\!] \xrightarrow{\texttt{AB?a}\langle \texttt{S}\rangle} [\![\texttt{G}'']\!]$
    therefore $[\![\texttt{G}]\!] \xrightarrow{\texttt{AB?a}\langle \texttt{S}\rangle} [\![\texttt{G}']\!]$

- $\lfloor$ASYNC2$\rfloor \mathtt{G} \xrightarrow{\ell} \mathtt{G}'$ where $\mathtt{G}' = \mathtt{transit}\ :\mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}''$
  By (1) $[\![\mathtt{G}]\!] = \mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S}\rangle.[\![\mathtt{G}'']\!]$ and $[\![\mathtt{G}']\!] = \mathtt{A} \to \mathtt{B} : \{\mathtt{a}\langle \mathtt{S}\rangle.[\![\mathtt{G}''']\!]\}$ By (2) $[\![\mathtt{G}'']\!] \xrightarrow{\ell} [\![\mathtt{G}''']\!]$, which follows from the premises $\mathtt{G}'' \xrightarrow{\ell} \mathtt{G}'''$ of the $\lfloor$ASYNC1$\rfloor$ and by the induction hypothesis and
  (3) $\mathtt{A}, \mathtt{B} \notin subj(\ell)$, which follows from the premise of $\lfloor$ASYNC2$\rfloor$:
  we can apply the $\lfloor$ASYNC2$\rfloor_{MPST}$ rule: $[\![\mathtt{G}]\!] \xrightarrow{\ell} [\![\mathtt{G}']\!]$

– if $\mathtt{G} = \mathtt{choice\ at\ A}\ \{\mathtt{G}_j^b\}_{j\in\{1,..,n\}})$
  By $\lfloor$CHOICE$\rfloor$ we have $\mathtt{G} \xrightarrow{\ell} \mathtt{G}'$ where by the rule premise we have for $\mathtt{G}'$ that $\mathtt{a}_i(\mathtt{S}_i)\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}'' \xrightarrow{\ell} \mathtt{G}'$ for $(i \in I)$ which brings us back to the first case.
– if $\mathtt{G} = \mathtt{rec\ t}\ \{\mathtt{G}''\}$ the thesis directly follows by induction since
  (1) by $\lfloor$REC$\rfloor \mathtt{G} \xrightarrow{\ell} \mathtt{G}'$ where $\mathtt{G}[\mathtt{rec\ t}\ \{\mathtt{G}\}/\mathtt{continue\ t}] \xrightarrow{\ell} \mathtt{G}'$
  (2) $[\![\mathtt{G}]\!] = \mu\mathtt{t}[\![\mathtt{G}'']\!]$
  By $\lfloor$REC$\rfloor$ $[\![\mathtt{G}'']\!][\mu\mathtt{t}.[\![\mathtt{G}'']\!]/\mathtt{t}]) \xrightarrow{\ell} [\![\mathtt{G}']\!]$
  (3) From IH, $\mathtt{G}' \lesssim [\![\mathtt{G}']\!]$ and therefore $\mathtt{G} \lesssim [\![\mathtt{G}]\!]$

Now we consider $[\![\mathtt{G}]\!] \lesssim \mathtt{G}$.
The proof is done by induction on transition rules applied to the encoding of $\mathtt{G}$.

1. $[\![\mathtt{G}]\!] = \mathtt{end}$ then both $[\![\mathtt{G}]\!]$ and $\mathtt{G}$ then no rules can be applied.
2. if $[\![\mathtt{G}]\!] = \mathtt{A} \to \mathtt{B} : \{\mathtt{a}\langle \mathtt{S}\rangle.[\![\mathtt{G}]\!]\}$, then we either have a send action by $\lfloor$SELECT$\rfloor_{MPST}$ or ℓ transition by $\lfloor$ASYNC1$\rfloor$.
   – $\lfloor$SELECT$\rfloor_{MPST}[\![\mathtt{G}]\!] \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} [\![\mathtt{G}']\!]$
   By $\mathtt{G} = \mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}''$ and $\mathtt{G}' = \mathtt{transit}\ :\mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}''$ and $\lfloor$SEND$\rfloor$ it follows that $\mathtt{G} \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} \mathtt{G}'$
   – $\lfloor$ASYNC1$\rfloor_{MPST}[\![\mathtt{G}]\!] \xrightarrow{\ell} [\![\mathtt{G}']\!]$ where
   $[\![\mathtt{G}]\!] = \mathtt{A} \to \mathtt{B} : \{\mathtt{a}\langle \mathtt{S}\rangle.[\![\mathtt{G}'']\!]\}$ and $[\![\mathtt{G}']\!] = \mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S}\rangle.[\![\mathtt{G}'']\!]$
   By (1) the rule premise $[\![\mathtt{G}'']\!] \xrightarrow{\ell} [\![\mathtt{G}''']\!]$ and by (2) IH it follows that $\mathtt{G}'' \xrightarrow{\ell} \mathtt{G}'''$.
   Given also that $\mathtt{A}, \mathtt{B} \notin subj(\ell)$, we can apply $\lfloor$ASYNC1$\rfloor$. Thus, $\mathtt{G} \xrightarrow{\ell} \mathtt{G}'$
3. if $[\![\mathtt{G}]\!] = [\![\mathtt{choice\ at\ A}\ \mathtt{a}_j(\mathtt{S}_j)\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}_j]\!] = \mathtt{A} \to \mathtt{B} : \{\mathtt{a}_j\langle \mathtt{S}_j\rangle.[\![\mathtt{G}_j]\!]\}_{j\in\{1,..,n\}}$
   Then by $\lfloor$CHOICE$\rfloor$ we have that $[\![\mathtt{G}]\!] \xrightarrow{\ell} [\![\mathtt{G}']\!]$ when $[\![\mathtt{a}_i(\mathtt{S}_i)\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}_i]\!] \xrightarrow{\ell} [\![\mathtt{G}']\!]$ for $i \in I$.
   Thus, we have to prove that
   if $[\![\mathtt{a}_i(\mathtt{S}_i)\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}_i]\!] \xrightarrow{\ell} [\![\mathtt{G}']\!]$ then $\mathtt{a}_i(\mathtt{S}_i)\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}_i \xrightarrow{\ell} [\![\mathtt{G}']\!]$, which follows from a).
4. if $[\![\mathtt{G}]\!] = \mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S}\rangle.[\![\mathtt{G}'']\!]$ $[\![\mathtt{G}]\!]$ can do a receive action by $\lfloor$BRANCH$\rfloor_{MPST}$ or ℓ transition by $\lfloor$ASYNC2$\rfloor$.
   – $\lfloor$BRANCH$\rfloor_{MPST}[\![\mathtt{G}]\!] \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} [\![\mathtt{G}']\!]$
   By $\mathtt{G} = \mathtt{transit}\ :\mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}''$ and
   $\mathtt{G}' = \mathtt{transit}\ :\mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{A}\ \mathtt{to}\ \mathtt{B}; \mathtt{G}''$ and $\lfloor$RECV$\rfloor$ it follows that $\mathtt{G} \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} \mathtt{G}'$

– $\lfloor \text{ASYNC2} \rfloor_{MPST} [\![\mathtt{G}]\!] \xrightarrow{\ell} [\![\mathtt{G}']\!]$ where
$[\![\mathtt{G}]\!] = \mathtt{A} \rightsquigarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S} \rangle.[\![\mathtt{G}'']\!]$ and $[\![\mathtt{G}']\!] = \mathtt{A} \rightarrow \mathtt{B} : \mathtt{a}\langle \mathtt{S} \rangle.[\![\mathtt{G}'']\!]$
By (1) the rule premise $[\![\mathtt{G}'']\!] \xrightarrow{\ell} [\![\mathtt{G}''']\!]$ and by (2) IH it follows that $\mathtt{G}'' \xrightarrow{\ell} \mathtt{G}'''$.
Given also that $\mathtt{A}, \mathtt{B} \notin subj(\ell)$, we can apply $\lfloor \text{ASYNC2} \rfloor$. Thus, $\mathtt{G} \xrightarrow{\ell} \mathtt{G}'$

5. if $[\![\mathtt{G}]\!] = \mu \mathtt{t}.[\![\mathtt{G}'']\!]$ the thesis directly follows by induction.

C From Local Protocols to Local Types

Proposition 4.12 (Correspondence of Local Protocols and Local Types): Let $\mathtt{T}$ be a Scribble local protocol, then $\mathtt{T} \approx (\!|\mathtt{T}|\!)$.

Proof. First, we consider $\mathtt{T} \lesssim [\![\mathtt{T}]\!]$.

The proof is done by induction (on the depth of the tree) on the transition rule applied.

1. (Base case) If $\mathtt{T} = \mathtt{end}$ then both $\mathtt{T}$ and $(\!|\mathtt{T}|\!)$ produce an empty set of traces.
2. (Inductive case) $\mathtt{T} \xrightarrow{\ell} \mathtt{T}'$ and we have to prove that $(\!|\mathtt{T}|\!) \xrightarrow{\ell} (\!|\mathtt{T}'|\!)$. We proceed by case analysis on the structure of $\mathtt{T}$
   (a) if $\mathtt{T} = \mathtt{a}(\mathtt{S})\ \mathtt{to}\ \mathtt{B}; \mathtt{T}'' \xrightarrow{\mathtt{AB!a}\langle \mathtt{S} \rangle} \mathtt{T}''$ by $\lfloor \text{SEND} \rfloor$
   $(\!|\mathtt{T}|\!) = \mathtt{B}!\{\mathtt{a} : \langle \mathtt{S} \rangle.(\!|\mathtt{T}''|\!)\} \xrightarrow{\mathtt{AB!a}\langle \mathtt{S} \rangle} (\!|\mathtt{T}''|\!)$ by $\lfloor \text{LSEL} \rfloor$
   (b) if $\mathtt{T} = \mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{B}; \mathtt{T}'' \xrightarrow{\mathtt{AB?a}\langle \mathtt{S} \rangle} \mathtt{T}''$ by $\lfloor \text{RECV} \rfloor$
   $(\!|\mathtt{T}|\!) = \mathtt{B}?\{\mathtt{a} : \langle \mathtt{S} \rangle.(\!|\mathtt{T}''|\!)\} \xrightarrow{\mathtt{AB?a}\langle \mathtt{S} \rangle} (\!|\mathtt{T}''|\!)$ by $\lfloor \text{LBRA} \rfloor$
   (c) if $\mathtt{T} = \mathtt{choice\ at\ A}\ \{\mathtt{T}_i\}_{i \in I}) \xrightarrow{\ell} \mathtt{T}'$
   Depending on the structure of $\mathtt{T}_i$, this case folds back to previous cases a) and b).
   if $\mathtt{T}_i = \mathtt{a}_i(\mathtt{S}_i)\ \mathtt{from}\ \mathtt{B}; \mathtt{T}'' \xrightarrow{\mathtt{AB!a}\langle \mathtt{S} \rangle} \mathtt{T}'' = \mathtt{T}'$ then $(\!|\mathtt{T}_i|\!) \xrightarrow{\mathtt{AB!a}\langle \mathtt{S} \rangle} (\!|\mathtt{T}'|\!)$ by $\lfloor \text{LSEL} \rfloor$
   if $\mathtt{T}_i = \mathtt{B}?\{\mathtt{a}_i : \langle \mathtt{S}_i \rangle.(\!|\mathtt{T}''|\!)\} \xrightarrow{\mathtt{AB?a}\langle \mathtt{S} \rangle} = \mathtt{T}'' = \mathtt{T}'$ then $(\!|\mathtt{T}_i|\!) \xrightarrow{\mathtt{AB?a}\langle \mathtt{S} \rangle} (\!|\mathtt{T}'|\!)$ by $\lfloor \text{LBRA} \rfloor$
   (d) if $\mathtt{T} = \mu \mathtt{t}.\mathtt{T}''$ the thesis directly follows by induction.

Now we consider $(\!|\mathtt{T}|\!) \lesssim \mathtt{T}$.

The proof is done by induction on transition rules applied to the encoding.

1. (Base case) If $(\!|\mathtt{T}|\!) = \mathtt{end}$ then both $(\!|\mathtt{T}|\!)$ and $\mathtt{T}$ produce an empty set of traces.
2. (Inductive case) $(\!|\mathtt{T}|\!) \xrightarrow{\ell} (\!|\mathtt{T}'|\!)$ and we have to prove that $\mathtt{T} \xrightarrow{\ell} \mathtt{T}'$. We proceed by case analysis on the structure of $(\!|\mathtt{T}|\!)$

– if $(\!|\mathtt{T}|\!) = \mathtt{B}!\{\mathtt{a} : \langle \mathtt{S} \rangle.(\!|\mathtt{T}''|\!)\}$
$\mathtt{B}!\{\mathtt{a} : \langle \mathtt{S} \rangle.(\!|\mathtt{T}''|\!)\} \xrightarrow{\mathtt{AB!a}\langle \mathtt{S} \rangle} (\!|\mathtt{T}''|\!)$ by $\lfloor \text{LSEL} \rfloor$
$\mathtt{T} = \mathtt{a}(\mathtt{S})\ \mathtt{to}\ \mathtt{B}; \mathtt{T}'' \xrightarrow{\mathtt{AB!a}\langle \mathtt{S} \rangle} (\!|\mathtt{T}''|\!)$ by $\lfloor \text{SEND} \rfloor$

– if $(\!|\mathtt{T}|\!) = \mathtt{B}?\{\mathtt{a} : \langle \mathtt{S}\rangle.(\!|\mathtt{T}''|\!)\}$
  $\mathtt{B}?\{\mathtt{a} : \langle \mathtt{S}\rangle.(\!|\mathtt{T}''|\!)\} \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} (\!|\mathtt{T}''|\!)$ by $\lfloor$LBRA$\rfloor$
  $\mathtt{T} = \mathtt{a}(\mathtt{S})\ \mathtt{from}\ \mathtt{B}; \mathtt{T}'' \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} \mathtt{T}''$ by $\lfloor$RECV$\rfloor$
– if $(\!|\mathtt{T}|\!) = \mathtt{B}?\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.(\!|\mathtt{T}_i|\!)\}_{i\in I}$
  $\mathtt{B}?\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.(\!|\mathtt{T}_i|\!)\}_{i\in I} \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} (\!|\mathtt{T}_j|\!)(j \in I)$
  By $\lfloor$RECV$\rfloor$ and the structure of $\mathtt{T}_i$ we have that $\mathtt{a}_i(\mathtt{T}_i)\ \mathtt{to}\ \mathtt{B}; \mathtt{T}_i \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} \mathtt{T}_i$ and therefore we can apply $\lfloor$CHOICE$\rfloor$
  Thus, $\mathtt{T} \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} \mathtt{T}_j$
– if $(\!|\mathtt{T}|\!) = \mathtt{A}!\{\mathtt{a}_i : \langle \mathtt{S}_i\rangle.(\!|\mathtt{T}|\!)\}_{i\in I}$ the case is analogical to the previous one.
– if $(\!|\mathtt{T}|\!) = \mu\mathtt{t}.\mathtt{T}''$ the thesis directly follows by induction.

Proposition 4.13 (Correspondence of Configurations): Let $(\mathtt{T}_1, \ldots, \mathtt{T}_n, w)$ be a configuration of Scribble local protocols, then $(\mathtt{T}'_1, \ldots, \mathtt{T}'_n, w) \approx ((\!|\mathtt{T}_1{}'|\!), \ldots, (\!|\mathtt{T}'_n|\!), w')$.

Proof. The proof is by induction on the number of transition steps. Inductive hypothesis: $(\mathtt{T}_1, \ldots, \mathtt{T}_n, w) \approx ((\!|\mathtt{T}_1|\!), \ldots, (\!|\mathtt{T}_n|\!), w)$
Now we want to prove that if $(\mathtt{T}_1, \ldots, \mathtt{T}_n, w) \xrightarrow{\ell} (\mathtt{T}'_1, \ldots, \mathtt{T}'_n, w')$ then
$((\!|\mathtt{T}_1|\!), \ldots, (\!|\mathtt{T}_n|\!), w) \xrightarrow{\ell} ((\!|\mathtt{T}'_1|\!), \ldots, (\!|\mathtt{T}'_n|\!), w')$
We do a case analysis on the transition label ℓ:
(1) if $\ell = \mathtt{AB!a}\langle \mathtt{S}\rangle$
By $\mathtt{T}_B \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} \mathtt{T}_B$ and Proposition 4.12 it follows: $(\!|\mathtt{T}_B|\!) \xrightarrow{\mathtt{AB!a}\langle \mathtt{S}\rangle} (\!|\mathtt{T}_B|\!)$
By definition of configuration of local protocols:
$w'_{\mathtt{AB}} = w_{\mathtt{AB}} \cdot \mathtt{a}(\mathtt{T}) \wedge (w_{ij} = w'_{ij})_{\text{for } ij \neq \mathtt{AB}}$.
(2) if $\ell = \mathtt{AB?a}\langle \mathtt{S}\rangle$
By $\mathtt{T}_B \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} \mathtt{T}'_B$ and Proposition 4.12 it follows: $(\!|\mathtt{T}_B|\!) \xrightarrow{\mathtt{AB?a}\langle \mathtt{S}\rangle} (\!|\mathtt{T}'_B|\!)$
By definition of configuration of local protocols:
$w'_{\mathtt{AB}} = w_{\mathtt{AB}} \cdot \mathtt{a}(\mathtt{S}) \wedge (\Rightarrow w_{ij} = w'_{ij})_{ij \neq \mathtt{AB}}$
In (1) and (2) we have by definition that $\mathtt{T}_i = \mathtt{T}'_i$(for $\neq \mathtt{AB}$), which by the inductive hypothesis implies that $(\!|\mathtt{T}_i|\!) = (\!|\mathtt{T}'_i|\!)$
Then by the definition of configuration of local protocols (from (1) and (2)) it follows that $((\!|\mathtt{T}_1|\!), \ldots, , \ldots, (\!|\mathtt{T}_n|\!), w) \xrightarrow{\ell} ((\!|\mathtt{T}'_1|\!), \ldots, (\!|\mathtt{T}'_n|\!), w')$.

D From Sribble to CFSM

Lemma 5.5 (Soundness of the translation). Given a local protocol $\mathtt{T}$, then $\mathtt{T} \approx \mathscr{A}(\mathtt{T})$.

Proof. In the proof we assume $\sigma = \mathtt{unfMap}(\mathtt{T}, \varnothing)$. Also we assume $\mathtt{T} \neq \mathtt{end}$. When $\mathtt{T} = \mathtt{end}$ the lemma is trivially true since $\mathtt{T}$ produces an empty set of traces, δ is an empty relation and q_0, the initial state, is also a final state.

First, we consider $\mathtt{T} \lesssim \mathscr{A}(\mathtt{T})$. Next we prove that if $\mathtt{T} \xrightarrow{\ell} \mathtt{T}'$ then $\exists \mathtt{T}_\mathtt{A}, \mathtt{T}'_\mathtt{A} \in Q$ such that $\mathtt{T}_\mathtt{A}\sigma = \mathtt{T}$ and $\mathtt{T}'_\mathtt{A}\sigma = \mathtt{T}'$, and $(\mathtt{T}_\mathtt{A}, \ell, \mathtt{T}'_\mathtt{A}) \in \delta$.

The proof is by induction on the transition relation for local types. In all cases we assume that $\mathtt{T} = \mathtt{T}_\mathtt{A}\sigma$.

- ⌊SEND⌋ if the reduction is by ⌊SEND⌋ we have
  $\mathtt{T}_\mathtt{A}\sigma = (\mathtt{msg\ to\ B}; \mathtt{T}'_\mathtt{A})\sigma = \mathtt{msg\ to\ B}; (\mathtt{T}'_\mathtt{A}\sigma)$.
  Thus, $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}'_\mathtt{A}\sigma$ where $\ell = \mathtt{msg!AB}$.
  Since $\mathtt{body}(\mathtt{T}_\mathtt{A}) = \mathtt{msg\ to\ B}; \mathtt{T}'_\mathtt{A}$ we proceed by case analysis on $\mathtt{T}'_\mathtt{A}$.
  Case 1: $\mathtt{T}'_\mathtt{A} \neq \mathtt{continue\ t}$;
  By Definition 5.2(1-2) and $\mathtt{body}(\mathtt{T}_\mathtt{A}) = \mathtt{msg\ to\ B}; \mathtt{T}'_\mathtt{A} \Rightarrow (\mathtt{T}_\mathtt{A}, \ell, \mathtt{T}'_\mathtt{A}) \in \delta$.
  Case 2: $\mathtt{T}'_\mathtt{A} = \mathtt{continue\ t}$;
  We have that $\mathtt{T}'_\mathtt{A}\sigma = \mathtt{continue\ t}\ \sigma = \mathtt{T}''$, where $\{\mathtt{t} \mapsto \mathtt{T}''\} \in \sigma$.
  By ⌊SEND⌋ we have $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}''$.
  By Definition 5.2(1-1) and $body\mathtt{T}_\mathtt{A} = \mathtt{msg\ to\ B}; \mathtt{T}'_\mathtt{A}$ it follows that $(\mathtt{T}_\mathtt{A}, \ell, \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\}) \in \delta$ with $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\} \in \mathtt{T}_\mathtt{O}$.
  By Lemma 5.4 we have $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\}\sigma = \mathtt{T}''$ and we conclude the case.
- ⌊RECV⌋ is similar to Case ⌊SEND⌋ and thus we omit.
- ⌊CHOICE⌋ if the reduction is by ⌊CHOICE⌋ we have
  $\mathtt{T}_\mathtt{A}\sigma = (\mathtt{choice\ at\ A}\{\mathtt{T}_{\mathtt{Ai}}\}_{i\in I})\sigma = \mathtt{choice\ at\ A}\{(\mathtt{T}_{\mathtt{Ai}}\sigma)\}_{i\in I}$.
  Case 1: if $\mathtt{T}_{\mathtt{Ai}}\sigma$ has the shape $(\mathtt{msg}_i\ \mathtt{to\ B}; \mathtt{T}'_{\mathtt{Ai}})\sigma = \mathtt{msg}_i\ \mathtt{to\ B}; (\mathtt{T}'_{\mathtt{Ai}}\sigma), \forall i \in I$ then we have $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}'_{\mathtt{Aj}}\sigma$ for some $j \in I$ with $\ell = \mathtt{msg}_j\mathtt{!AB}$.
  Since $\mathtt{body}(\mathtt{T}_\mathtt{A}) = \mathtt{T}_\mathtt{A}$, we proceed by case analysis on $\mathtt{T}'_{\mathtt{Aj}}$.
  Case 1.1: $\mathtt{T}'_{\mathtt{Aj}} \neq \mathtt{continue\ t}$;
  By Definition 5.2(3-a-2) and $\mathtt{body}(\mathtt{T}_\mathtt{A}) = \mathtt{T}_\mathtt{A}$ we have $(\mathtt{T}_\mathtt{A}, \ell, \mathtt{T}'_{\mathtt{Aj}}) \in \delta$.
  Case 1.2: $\mathtt{T}_\mathtt{A}\mathtt{j}' = \mathtt{continue\ t}$;
  (1*) We have that $\mathtt{T}'_{\mathtt{Aj}}\sigma = \mathtt{continue\ t}\ \sigma = \mathtt{T}''$, where $\{\mathtt{t} \mapsto \mathtt{T}''\} \in \sigma$.
  (2*) By ⌊CHOICE⌋ we have $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}''$.
  By Definition 5.2(3-a-2) and $\mathtt{body}(\mathtt{T}_\mathtt{A}) = \mathtt{T}_\mathtt{A} \Rightarrow (\mathtt{T}_\mathtt{A}, \ell, \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\}) \in \delta$ with $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\} \in \mathtt{T}_\mathtt{O}$.
  By Lemma 5.4 we have $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\}\sigma = \mathtt{T}''$.
  Applying the IH to (1*) and (2*) we conclude the case.
  Case 2: if $\mathtt{T}_{\mathtt{Ai}}\sigma$ has the shape $(\mathtt{msg}_i\ \mathtt{to\ B}; \mathtt{T}'_{\mathtt{Ai}})\sigma = \mathtt{msg}_i\ \mathtt{to\ B}; (\mathtt{T}'_{\mathtt{Ai}}\sigma)$ this case is similar to Case 1 and thus we omit.
  Note that since the normal form of local types does not allow for unguarded choice, hence, all possible transitions of $\mathtt{T}_\mathtt{A}\sigma$ are the transitions from Case 1 and Case 2.
- ⌊REC⌋ if the reduction is by ⌊REC⌋ we have then $\mathtt{T}_\mathtt{A}\sigma = (\mathtt{rec\ t}\ \{\mathtt{T}'_\mathtt{A}\})\sigma = \mathtt{T}'_\mathtt{A}\sigma$. We note that $\mathtt{T}'_\mathtt{A}\sigma$ does not contain the term $\mathtt{continue\ t}$ since unguarded recursive variables are not allowed. Hence, $\mathtt{T}'_\mathtt{A}\sigma$ is either send, receive or choice and by IH and ⌊SEND⌋, ⌊RECV⌋, ⌊CHOICE⌋ we conclude this case.

We next consider $\mathscr{A}(\mathtt{T}) \lesssim \mathtt{T}$. We prove that given a local protocol $\mathtt{T}_\mathtt{O}$ if $(\mathtt{T}_\mathtt{A}, \ell, \mathtt{T}'_\mathtt{A}) \in \delta$ then $\exists \mathtt{T}$ s.t. $\mathtt{T} = \mathtt{T}_\mathtt{A}\sigma$ and $\mathtt{T} \xrightarrow{\ell} \mathtt{T}'$ and $\mathtt{T}' = \mathtt{T}'_\mathtt{A}\sigma$ with $\sigma = \mathtt{unfMap}(\mathtt{T}_\mathtt{O}, \varnothing)$. We proceed by case analysis on the transitions in δ.

Case 1: $\mathtt{T}_\mathtt{A} = \mathtt{msg\ to\ B}; \mathtt{T}''_\mathtt{A}$ and $\ell = \mathtt{msg?AB}$.
Then $\mathtt{T}' = \mathtt{T}_\mathtt{A}\sigma$ and we have by $\lfloor \textsc{send} \rfloor \mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}''_\mathtt{A}\sigma$.
Case 1.1: if $\mathtt{T}''_\mathtt{A} = \mathtt{T}'_\mathtt{A} \neq \mathtt{continue\ t}$
The hypothesis follows from $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}'_\mathtt{A}\sigma$.
Case 1.2: if $\mathtt{T}''_\mathtt{A} = \mathtt{continue\ t}$
By Definition 5.2 $\mathtt{T}'_\mathtt{A} = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'''_\mathtt{A}\} \in \mathtt{T}_0, \mathtt{t} \in \mathtt{t}$.
By Definition 5.3 and Lemma 5.4 we have $\mathtt{t} \mapsto \mathtt{T}''$ s.t. $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'''_\mathtt{A}\}\sigma = \mathtt{T}''$.
From IH and $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}''_\mathtt{A}\sigma = \mathtt{T}'' = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'''_\mathtt{A}\}\sigma = \mathtt{T}'_\mathtt{A}\sigma$ we conclude the case.
Case 2: $\mathtt{T}_\mathtt{A} = \mathtt{msg\ from\ B}; \mathtt{T}''_\mathtt{A}$ and $\ell = \mathtt{msg!AB}$.
Proceeds in a similar way as Case 2 and thus we omit.
Case 3: $\mathtt{T}_\mathtt{A} = \mathtt{choice\ at}\{\mathtt{msg}_i\ \mathtt{to\ B}; \mathtt{T}_{\mathtt{A}i}\}_{i \in I}$
Then we have by $\lfloor \textsc{choice} \rfloor$
$\mathtt{T}_\mathtt{A}\sigma = \mathtt{choice\ at}\{\mathtt{msg}_i\ \mathtt{to\ B}; \mathtt{T}_{\mathtt{A}i}\sigma\}_{i \in I} \xrightarrow{\mathtt{msg!AB}} \mathtt{T}_{\mathtt{A}j}\sigma$ for some $j \in I$.
Case 3.1: if $\mathtt{T}_{\mathtt{A}j} = \mathtt{T}'_\mathtt{A} \neq \mathtt{continue\ t}$
From IH and $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}'_\mathtt{A}\sigma$ we conclude the case.
Case 3.2: if $\mathtt{T}_{\mathtt{A}j} = \mathtt{continue\ t}$
By Definition 5.2 $\mathtt{T}_{\mathtt{A}j} = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'''_\mathtt{A}\} \in \mathtt{T}_0, \mathtt{t} \in \mathtt{t}$
By Definition 5.3 and Lemma 5.4 we have $\mathtt{t} \mapsto \mathtt{T}''$ s.t. $\mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'''_\mathtt{A}\}\sigma = \mathtt{T}''$.
We have that $\mathtt{T}_\mathtt{A}\sigma \xrightarrow{\ell} \mathtt{T}_{\mathtt{A}j}\sigma = \mathtt{T}'' = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}'''_\mathtt{A}\}\sigma = \mathtt{T}'_\mathtt{A}\sigma$, hence we conclude the case.
Case 4: $\mathtt{T}_\mathtt{A} = \mathtt{choice\ at}\{\mathtt{msg}_i\ \mathtt{from\ B}; \mathtt{T}_{\mathtt{A}i}\}_{i \in I}$
Proceeds in a similar way as Case 3 and thus we omit.
Case 5: $\mathtt{T}_\mathtt{A} = \mathtt{rec}\ \vec{\mathtt{t}}\ \{\mathtt{T}''_\mathtt{A}\}$
Note that the $\mathtt{T}''_\mathtt{A}$ is either message send or message receive. Hence, By applying the IH and Case 1, 2 we conclude the case.

References

1. Behavioural Types: From Theory to Tools. River Publishers, Delft (2017)
2. Bocchi, L., Lange, J., Yoshida, N.: Meeting deadlines together. In: 26th International Conference on Concurrency Theory. LIPIcs, vol. 42, pp. 283–296. Schloss Dagstuhl (2015)
3. Bocchi, L., Yang, W., Yoshida, N.: Timed multiparty session types. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 419–434. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44584-6_29
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983)
5. Castro, D., Hu, R., Jongmans, S.-S., Ng, N., Yoshida, N.: Distributed programming using role parametric session types in go. In: 46th ACM SIGPLAN Symposium on Principles of Programming Languages, pp. 1–30. ACM (2019)
6. W3C WS-CDL. http://www.w3.org/2002/ws/chor/
7. Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N.: A gentle introduction to multiparty asynchronous session types. In: Bernardo, M., Johnsen, E.B. (eds.) SFM 2015. LNCS, vol. 9104, pp. 146–178. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18941-3_4

8. De Nicola, R., Ferrari, G., Pugliese, R.: Klaim: a kernel language for agents interaction and mobility. IEEE Trans. Softw. Eng. **24**, 315–330 (1998)
9. Demangeon, R., Honda, K., Raymond, H., Neykova, R., Yoshida, N.: Practical interruptible conversations: distributed dynamic verification with multiparty session types and Python. FMSD **46**(3), 197–225 (2015)
10. Deniélou, P.-M., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 194–213. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28869-2_10
11. 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.) ICALP 2013. LNCS, vol. 7966, pp. 174–186. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_18
12. Honda, K., et al.: Structuring communication with session types. In: Agha, G., et al. (eds.) Concurrent Objects and Beyond. LNCS, vol. 8665, pp. 105–127. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44471-9_5
13. Honda, K., Mukhamedov, A., Brown, G., Chen, T.-C., Yoshida, N.: Scribbling Interactions with a Formal Foundation. In: Natarajan, R., Ojo, A. (eds.) ICDCIT 2011. LNCS, vol. 6536, pp. 55–75. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19056-8_4
14. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053567
15. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL 2008, pp. 273–284. ACM (2008)
16. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. JACM **63**, 1–67 (2016)
17. Hu, R., Neykova, R., Yoshida, N., Demangeon, R., Honda, K.: Practical interruptible conversations. In: Legay, A., Bensalem, S. (eds.) RV 2013. LNCS, vol. 8174, pp. 130–148. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40787-1_8
18. Hu, R., Yoshida, N.: Hybrid session verification through endpoint API generation. In: Stevens, P., Wąsowski, A. (eds.) FASE 2016. LNCS, vol. 9633, pp. 401–418. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49665-7_24
19. Hu, R., Yoshida, N.: Explicit connection actions in multiparty session types. In: Huisman, M., Rubin, J. (eds.) FASE 2017. LNCS, vol. 10202, pp. 116–133. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54494-5_7
20. Lange, J., Tuosto, E.: Yoshida, N.: From communicating machines to graphical choreographies. In: POPL, pp. 221–232. ACM (2015)
21. Neykova, R., Bocchi, L., Yoshida, N.: Timed runtime monitoring for multiparty conversations. FAOC **29**, 877–910 (2017)
22. Neykova, R., Hu, R., Yoshida, N., Abdeljallal, F.: A session type provider: compile-time API generation for distributed protocols with interaction refinements in F#. In: 27th International Conference on Compiler Construction, pp. 128–138. ACM (2018)
23. Neykova, R., Yoshida, N.: Let it recover: multiparty protocol-induced recovery. In: CC. ACM (2017, to appear)
24. Ng, N., de Figueiredo Coutinho, J.G., Yoshida, N.: Protocols by default. In: Franke, B. (ed.) CC 2015. LNCS, vol. 9031, pp. 212–232. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46663-6_11

25. Scalas, A., Dardha, O., Hu, R., Yoshida, N.: A linear decomposition of multiparty sessions for safe distributed programming. In: 31st European Conference on Object-Oriented Programming. LIPIcs, vol. 74, pp. 24:1–24:31. Schloss Dagstuhl (2017)
26. Scribble Project.. http://www.scribble.org
27. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D., Philokyprou, G., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58184-7_118
28. Yoshida, N., Hu, R., Neykova, R., Ng, N.: The scribble protocol language. In: Abadi, M., Lluch Lafuente, A. (eds.) TGC 2013. LNCS, vol. 8358, pp. 22–41. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05119-2_3