Embedding Session Types in HML*

Laura Bocchi
Imperial College, London
l.bocchi@imperial.ac.uk

Romain Demangeon
Imperial College, London
rdemang@gmail.com

Abstract Recent work on the enhancement of multiparty session types with logical annotations enable the effective verification of properties on (1) the structure of the conversations, (2) the sorts of the messages, and (3) the actual values exchanged. In [3] we extend this work to enable the specification and verification of mutual effects of multiple cross-session interactions. Here we give a sound and complete embedding into the Hennessy-Milner logic to justify the expressiveness of the approach in [3] and to provide it with a logical background that will enable us to compare it with similar approaches.

1 Introduction

The Hennessy-Milner Logic (HML) is an expressive modal logic with a strong semantic characterisation [10] that enables the specification of arbitrary behavioural properties of processes. Recent work on the enhancement of multiparty session types with logical annotations [4, 3] addressed key challenges for logical specifications of processes, which were unexplored in the context of HML, such as the tractability of specifications of multiparty choreographies.

The work in [4, 3] is based on multiparty session types [11, 4, 6] and inherits the same top-down approach. The key idea is that conversations are built as the composition of units of design called *sessions* which are specified from a global perspective (i.e., as a global type). Each global type is then *projected* into one local type for each participant, making the responsibilities of each endpoint explicit. This approach enables: (1) the effective verification of properties such as session fidelity, progress, and error freedom, and (2) the modular local verification (i.e., of each principal) of global properties of multiparty interactions.

The direct use of HML for the same purpose would require to start from endpoint specifications and then to check their mutual consistency, and would not offer the same tractability. Starting from global assertions, instead, results in significant concision, while still enjoying generality in the modelling and verification of choreographies.

By enhancing multiparty session types with logical annotations, [4] enables the effective verification of properties on the actual values exchanged, other than the properties on the sorts of the messages guaranteed by [11, 6]. For instance, global type G in (1) describes, following a similar syntax to [6], a conversation where role $\mathtt{S}$ sends role $\mathtt{C}$ an integer and then continues as specified by global type G'. Following [4], assertion $\mathscr{G}$ in (1) can be obtained by annotating global type G; assertion $\mathscr{G}$ further prescribes that the exchanged value, say y, must be greater than 10. Note that y is bound in $\mathscr{G}'$ and the fact $\{y > 10\}$ can be relied on in the subsequent interactions occurring in $\mathscr{G}'$.

$$G = \mathtt{S} \to \mathtt{C} : (\mathtt{int}).G' \qquad \mathscr{G} = \mathtt{S} \to \mathtt{C} : (y : \mathtt{int})\{y > 10\}.\mathscr{G}' \tag{1}$$

*This work has been partially sponsored by the project Leverhulme Trust award Tracing Networks, Ocean Observatories Initiative and EPSRC EP/K011715/1, EP/K034413/1 and EP/G015635/1.

N. Yoshida, W. Vanderbauwhede (Eds.): Programming Language
Approaches to Concurrency- and Communication-cEntric
Software 2013 (PLACES'13)
EPTCS 137, 2013, pp. 53–62, doi:10.4204/EPTCS.137.5

In [3] we extended [4] with the capability to refer to *virtual states* local to each network principal, hence expressing not only properties confined to the single multiparty sessions, but also stateful specifications incorporating mutual effects of multiple sessions run by a principal.

$$\mathtt{S} \to \mathtt{C} : (y{:}\mathtt{int})\{y > 10 \wedge y = \mathtt{S.x}\}\langle \mathtt{S.x{+}{+}} \rangle \tag{2}$$

Consider now the protocol in (2). The description of this simple distributed application implies behavioural constraints of greater depth than the basic communication actions. The (sender-side) *predicate and effect* for the interaction step, $\{y > 10 \wedge y = \mathtt{S.x}\}\langle \mathtt{S.x{+}{+}} \rangle$, asserts that the message *y* sent to each client must equal the current value of S.x, a state variable *x* allocated to the *principal* serving as S; and that the local effect of sending this message is to increment S.x. In this way, S is specified to send incremental values across *consecutive* sessions. The resulting global specifications are called *multiparty stateful assertions* (MPSAs), and model the skeletal structure of the interactions of a session, the constraints on the exchanged messages and on the branches to be followed, and the *effects* of each interaction on the virtual state.

In order to obtain a clear understanding of the status of the logical methodology proposed in [3], it is useful to relate its notion of assertion to a more standard approach in process logic. This enables us to integrate different methods catering for different concerns, for which we may need a common logical basis. In this paper we consider the HML with predicates in [4, 2], and we justify the relevance of the stateful logical layer of [3] by embedding the behaviours of each role in a session – i.e., the projections of MPSAs – into a HML formula. In this way, the required predicates will hold if a process and its state perform reductions and updates matching those of the specification.

$$\forall y : \mathtt{Nat}, [s_{\mathtt{C}}(y)](y = \mathtt{S.x} \wedge [\mathtt{S.x{+}{+}}]\mathtt{true}) \tag{3}$$

(3) is the formula corresponding to the behaviour of S in (2) on channel *s*, where $[\ell]\phi$ means "if a process and its state perform the action ℓ, the resulting pair satisfies ϕ". Communications and state updates are treated as actions of a labelled transition system.

We explain how specifications handling several roles in several sessions can be soundly and completely embedded, through the use of an *interleaving* of formulae, exploring all the possible orders in which the actions coming from different sessions can be performed, and ensuring that predicates are always satisfied.

2 HML Embedding

Logical layer We propose an embedding of our analysis into Hennessy Milner Logic (HML), together with soundness and completeness results. The analysis in [3] be seen as the superposition of two analyses: a session type system and a logical layer. The former ensures that a process is able to perform some visible actions described by the specification and can be mechanically, yet tediously encoded in HML, for instance, by using a "surely/then" modality [2]. Our contribution focuses on the embedding of the latter, namely on *predicate safety*, ensuring that stateful predicates will be satisfied. As consequence, the completeness result we propose (Proposition 7), states that if a process abides to the session-type component L of a local assertion $\mathscr{L}$ (obtained by erasing all predicates in $\mathscr{L}$) and satisfies the logical encoding of $\mathscr{L}$, then it is provable against $\mathscr{L}$.

MPSAs We focus here on local assertions, each referring to a specific role and deriving, via projection, from a global assertion as in [3] – e.g., as (2). Local assertions are defined by the grammar below and are ranged over by $\mathscr{L}$.

$$\mathscr{L} ::= \mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I} \mid \mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}$$
$$\mu t\{y : A'\}(x : S).\mathscr{L} : A \mid t(y : A') \mid \mathsf{end}$$

Selection $\mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}$ models an interaction where the role sends $\mathtt{p}$ a branch label l_i and a message x_i of sort U_i (e.g., `int`, `bool`, etc., and local assertion for delegation) and continues as $\mathscr{L}_i$, with being A_i predicates[1] and E_i state updates. Branching $\mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}$ is dual to selection. We use guarded recursion defining a recursion parameter x initially set equal to a value satisfying the initialisation predicate A', where y is the free variable of A', and with A being an invariant predicate. Recursive call $t(y : A')$ instantiates a new iteration of t where the recursion parameter takes a value satisfying A', with y free variable of A'.

[3] uses local assertions as a basis for the verification of a processes, ranged over by P.

$$P ::= \mathbf{0} \mid \overline{u}[\mathtt{n}](y).P \mid u[\mathtt{i}](y).P \mid k[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I} \mid k[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$$
$$P \mid Q \mid (\mu X(x).P)\langle e\rangle \mid X\langle e\rangle$$

A process can be an idle process $\mathbf{0}$, a session request/accept, a guarded command [9], a branching, a parallel composition of processes, a recursive definition and invocation. Session request $\overline{u}[\mathtt{n}](y).P$ multicasts a request to each session accept process $u[\mathtt{i}](y).P$ (with $i \in \{2,..,n\}$) by synchronisation through a shared name u and continuing as P. Guarded command and branching processes represent communications through an established session k. Guarded command $k[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I}$ acts as role $\mathtt{p}$ in session k and sends role $\mathtt{q}$ one of the labels l_i. The choice of the label is determined by boolean expressions e_i, assuming $\vee_{i\in I} e_i = \mathsf{true}$ and $i \neq j$ implies $e_i \wedge e_j = \mathsf{false}$. Each label l_i is sent with the corresponding expression e_i' which specifies the value for x_i, assuming e_i' and x_i have the same type. Branching $k[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$ plays role $\mathtt{q}$ in session k and is ready to receive from $\mathtt{p}$ one of the labels l_i and a value for the corresponding x_i, then behaves as P_i after instantiating x_i with the received value. In guarded command (resp. branching), the local state of the sender (resp. receiver) is updated according to update E_i; in both processes each x_i binds its occurrences in P_i and E_i.

The judgements are of the form $\mathscr{C}; \Gamma \vdash P \triangleright \Delta$ where:

- $\mathscr{C}$ is the assertion environment that is the set of preconditions built, during the verification, as the incremental conjunction of the predicates occurring in the branchings,
- Γ determines which types of sessions can be initiated by a process by mapping shared channels to global assertions (e.g., if $\Gamma(a) = \mathtt{I}(\mathscr{G})$ then P can be invited to join a session specified by $\mathscr{G}$),
- Δ is the session environment mapping sessions that P has joined, say $s[\mathtt{p}]$, to local types.

We write omit Γ (resp. $\mathscr{C}$) in the judgment when it is the empty mapping (resp. `true` precondition).

HML Here, the behaviour prescribed for P is modelled using the standard HML with the first-order predicates as in [2]. We use the same type of predicate A as in MPSAs. We associate this HML with a LTS where actions ℓ model communications and state updates.

$$\ell ::= s[\mathtt{p},\mathtt{q}](x) \mid \overline{s[\mathtt{p},\mathtt{q}]}(x) \mid E$$

Namely, $s[\mathtt{p},\mathtt{q}](x)$ is an input action, $\overline{s[\mathtt{p},\mathtt{q}]}(x)$ is an output action, and E is a state update. We let states to be ranged over by $\sigma, \sigma', \ldots$ and we write $\sigma' = \sigma\,\mathtt{after}\,\ell$ for the state σ' obtained by updating σ as

[1] As in [4, 3] we assume that the validity of closed formulae is decidable.

$$\frac{P,\sigma \models \phi_1 \qquad P,\sigma \models \phi_2}{P,\sigma \models \phi_1 \wedge \phi_2} \qquad \frac{}{P,\sigma \models \texttt{true}} \qquad \frac{\text{if } P,\sigma \models \phi_1 \text{ then } P,\sigma \models \phi_2}{P,\sigma \models \phi_1 \Rightarrow \phi_2}$$

$$\frac{\text{For all } P',\sigma' \text{ s.t. } P,\sigma \xrightarrow{\ell} P',\sigma', P',\sigma' \models \phi}{P \models [\ell]\phi} \qquad \frac{\sigma \vdash_{bool} A}{P,\sigma \models A} \qquad \frac{\text{For all values } v \text{ of type } T, P,\sigma \models \phi[v/x]}{P,\sigma \models \forall x : T.\phi}$$

Figure 1: Logical rules

prescribed by E. $P,\sigma \xrightarrow{\ell} P',\sigma'$ if either: (a) ℓ is an input or output action, $P \xrightarrow{\ell} P'$ and $\sigma' = \sigma$, or (b) ℓ is an update action, $P = P'$, and $\sigma' = \sigma\,\texttt{after}\,\ell$.

We use ϕ to denote HML-formulae, which are built from predicates, implications, universal quantifiers, conjunctions and *must* modalities. The logic used in this *safety embedding* is positive: if we remove the implication symbol, there is no negation, no existential quantifier, no disjunction and no may modality. Additionally, the implication will always appear as $A \Rightarrow \phi$ meaning that modalities never appear in the negative side.

$$\phi ::= \texttt{true} \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \mid [\ell]\phi \mid A \mid \forall x : S.\phi$$

The satisfactions rules (Figure 1) are fairly standard. For a pair P,σ to satisfy a predicate A, written $P,\sigma \models A$, A has to hold with respect to σ, denoted by $\sigma \vdash_{bool} A$, meaning that $\sigma(A)$ is a tautology for the boolean logic.

The embedding of local types we propose is parameterised with a session channel $s[\mathtt{p}]$. Predicates appearing in input prefixes are embedded as premises in implications, as in (5), and predicates in output prefixes have to be satisfied, as in (4), yielding:

$$\|\mathtt{q}!\{l_i(x_i : S_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}\|^{s[\mathtt{p}]} = \bigwedge_{i\in I} \forall x_i : S_i, [\overline{s[\mathtt{p},\mathtt{q}]}(x_i)](A_i \wedge [E_i]\|\mathscr{L}_i\|^{s[\mathtt{p}]}) \tag{4}$$

$$\|\mathtt{q}?\{l_j(x_j : S_j)\{A_j\}\langle E_j\rangle.\mathscr{L}_j\}_{j\in J}\|^{s[\mathtt{p}]} = \bigwedge_{j\in J} \forall x_j : S_j, [s[\mathtt{q},\mathtt{p}](x_j)](A_j \Rightarrow \|\mathscr{L}_j\|^{s[\mathtt{p}]}) \tag{5}$$

The embedding of selection (4), is a conjunction of the formulae corresponding to the branches: for each value sent on the session channel, predicates should be satisfied and, if the state is updated, the embedding of the continuation should hold. For branching types (5), the assertion is used as an hypothesis and no update appears.

3 Soundness

For the sake of clarity, we divide our proofs into two parts, one proving *simple* preciseness, that is soundness and completeness when the specification is a single session type, the other proves the *full* completeness, for any specification. This corresponds to the two challenges we tackle in our approach: the translation of a type into a formula, and the handling of the possible interleaving of concurrent types.

Simple Preciseness We postpone the introduction of interleavings to focus on proving our result for single types, obtaining a *simple* preciseness result.

The following lemma states that a process cannot perform an action on a channel that does not appear in its type, that a process that does not perform any action does not change the set of formulae it satisfies, that satisfaction of assertions is stable by reduction and that validity of satisfaction judgements is stable by well-typed substitutions.

Lemma 1 *If $\mathscr{C};\Gamma \vdash P \triangleright \Delta$ and $s[\mathtt{p}] \notin \Delta \cup \Gamma$, then there is no P' s.t. $P,\sigma \xrightarrow{\ell_s} P',\sigma$ for any action ℓ_s of the form $s![\mathtt{p},\mathtt{q}](x)$ or $s![\mathtt{q},\mathtt{p}](x)$.*

Similarly, if $a : \mathtt{I}(\mathscr{G}) \notin \Gamma$, there are no P' and $s[\mathtt{p}]$ such that $P,\sigma \xrightarrow{a(s[\mathtt{p}])} P',\sigma$.
If $P_1,\sigma \models \phi$ and P_2 cannot make any action, then $P_1 \mid P_2,\sigma \models \phi$.
If $P,\sigma \models A$ and $P \xrightarrow{\ell} P'$, then $P',\sigma \models A$.
If $P,\sigma \models \phi$ and $x : S, v : S$ are not bound in P,σ and ϕ, then $P[v/x],\sigma \models \phi[v/x]$.

Proof By induction on ϕ, as our processes and formulas abide a Barendregt convention, the case $\forall y.\phi$ is easy as $y \neq x$ and $y \neq v$. The only interesting cases are assertion and must modality:

- Case $\phi = A$. The logic rules notifies that $\sigma(A)$ is a tautology, so any instantiation of its free variable should be so. Thus $\sigma(A)\{v/x\}$ is a tautology and any process (in particular $P\{v/x\}$) and the state σ form a pair that satisfies it.
- Case $\phi = [\alpha]\phi'$. We prove, by induction on the reduction rules, that if $P \xrightarrow{\alpha} P'$, then $P\{v/x\} \xrightarrow{\alpha\{v/x\}} P'\{v/x\}$ and use the induction hypothesis.

We state, thanks to the previous lemmas, the following 'simple' soundness for simple local types, that is for Δ with one single local type:

Proposition 2 (Simple Soundness) *If $\mathscr{C} \vdash P \triangleright s[\mathtt{p}] : \mathscr{L}$, then $(P,\sigma) \models \mathscr{C} \Rightarrow \|\mathscr{L}\|^{s[\mathtt{p}]}$.*

In order to state simple completeness we define unasserted types. Unasserted types are built from:

$$L ::= \ \mathtt{p}?\{l_i(U_i).L_i\}_{i\in I} \mid \mathtt{p}!\{l_i(U_i).L_i\}_{i\in I} \mid \mu t.L \mid t \mid \mathsf{end}$$

An unasserted local type can be obtained from an asserted local type using an erasing operator. The erasing operator $\mathbf{Er}(\mathscr{L})$ is defined by the removal of every assertion, update and variable from $\mathscr{L}$. Unasserted typing rules for the judgements $\vdash P \triangleright \Delta$ are easily deduced from the asserted ones.

Proposition 3 (Simple Completeness) *For all $\mathscr{L}$, if $\vdash P \triangleright s[p] : \mathbf{Er}(\mathscr{L})$ and $P,\sigma \models \mathscr{C} \Rightarrow \|\mathscr{L}\|^{s[\mathtt{p}]}$ then $\mathscr{C} \vdash P \triangleright s[p] : \mathscr{L}$.*

Proof By induction on the typing judgement $\vdash P \triangleright s[\mathtt{p}] : \mathbf{Er}(\mathscr{L})$:

- Case *branching*. We have $\mathscr{L} = \mathtt{p}_0?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}$. Let $i \in I$ and suppose $\mathscr{C}$ holds. We have from the hypothesis $\vdash P \triangleright \mathtt{p}_0?\{l_i(U_i).L_i\}_{i\in I}$. The unasserted typing rules give that $P = s[\mathtt{p}_0,\mathtt{p}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$, and $\vdash P_i \triangleright s[\mathtt{p}] : L_i$. We know that $P,\sigma \models \mathscr{C} \implies \|\mathscr{L}\|^s$, which is:

$$P,\sigma \models \bigwedge_{i\in I} \forall x_i.[\overline{s}(x_i)](A_i \implies [E_i]\|\mathscr{L}_i\|^s \wedge (A_i \wedge \mathscr{C})$$

  rules, that P can perform $s(x_i)$ to $P_i,\sigma \models (A_i \implies [E_i]\|\mathscr{L}_i\|^{s[\mathtt{p}]} \wedge A_i)$. We see that σ can perform E_i to $\sigma\,\mathtt{after}\,E_i$, meaning that we have $P_i,\sigma\,\mathtt{after}\,E_i \models (A_i \implies \|\mathscr{L}_i\|^{s[\mathtt{p}],\mathscr{I}})$, we use the induction hypothesis to get $\mathscr{C} \wedge A_i \vdash P_i \triangleright \mathscr{L}_i$. To sum up, for all i, $\mathscr{C}, A_i \vdash P_i \triangleright s[\mathtt{p}] : \mathscr{L}_i$. We use the proof rule for branching to prove $\mathscr{C} \vdash P \triangleright s[p] : \mathscr{L}$.
- Case *selection*. We have $\mathscr{L} = \mathtt{p}_0!\{l_i(U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}$. Suppose $\mathscr{C}$ holds and $\sigma \models \mathscr{I}$. We have from the hypothesis $\vdash P \triangleright s[\mathtt{p}] : \mathtt{p}_0?\{l_i(U_i).L_i\}_{i\in I}$. The unasserted typing rules give $P = s[\mathtt{p},\mathtt{p}_0]!\{e_j \mapsto l_j\langle e'_j\rangle(x_j)\langle E_j\rangle;P_j\}_{j\in J}$, and $\vdash P_j \triangleright s[\mathtt{p}] : L_j$. We know that $P,\sigma \models \mathscr{C} \implies \|\mathscr{L}\|^s$, which is $P \models \bigwedge_{i\in I} \forall x_i.[\overline{s}(x_i)]A_i \wedge \|\mathscr{L}_i\|^{s[\mathtt{p}]} \wedge (A_i \wedge \mathscr{C})$. In particular, as $\mathscr{C}$ holds, $P \models \overline{[s[\mathtt{p},\mathtt{p}_0]}(x_j)]A_j \wedge$

$$[\ell_1]\phi_1 \rtimes [\ell_2]\phi_2 = [\ell_1](\phi_1 \rtimes [\ell_2]\phi_2) \wedge [\ell_2]([\ell_1]\phi_1 \wedge \phi_2)$$
$$[\ell_1]\phi_1 \rtimes (\phi_{2,1} \wedge \phi_{2,2}) = [\ell_1](\phi_1 \rtimes \phi_{2,1}) \wedge [\ell_1](\phi_1 \rtimes \phi_{2,2})$$
$$\phi \rtimes \mathtt{true} = \phi$$
$$\phi \rtimes (\phi_1 \wedge \phi_2) = (\phi \rtimes \phi_1) \wedge (\phi \rtimes \phi_2)$$
$$(\phi_1 \wedge \phi_2) \rtimes \phi = (\phi_1 \rtimes \phi) \wedge (\phi_2 \rtimes \phi)$$
$$\forall x : T.\phi_1 \rtimes \phi_2$$
$$(A \Rightarrow \phi_1) \rtimes \phi_2 = A \Rightarrow (\phi_1 \rtimes \phi_2)$$

Figure 2: Rules for interleaving

$\|\mathscr{L}_j\|^{s[\mathtt{p}]} \wedge (A_j))$ We know from the shape of P, given above, and the reduction rules, that P can perform $\overline{s[\mathtt{p},\mathtt{p}_0]}(x_j)$ to $P_j \models (A_j \wedge \|\mathscr{L}_j\|^s)$, meaning that A_j holds. Also, σ can perform E_j to $\sigma\,\mathtt{after}\,E_j$. To sum up, we have $\mathscr{C} \implies A_j$, $P_j \models \mathscr{C} \implies \|\mathscr{L}_j\|^{s[\mathtt{p}]}$ and $\vdash P_j \triangleright s[\mathtt{p}] : \mathscr{L}_j$, we use the induction hypothesis to get $\mathscr{C} \vdash P_j \triangleright : \mathscr{L}_j$ and this allows us to use the proof rule for selection to prove $\mathscr{C} \vdash P \triangleright s[\mathtt{p}] : \mathscr{L}$.

- Case *parallel*. No assertion appear in the parallel rule and we can use Lemmas 1.1 and 1.2 to state that exactly one side of the parallel composition satisfies the formula (along with the same state σ). As a consequence, we use the induction hypothesis twice and conclude.
- Case *end*. $\mathscr{L} = \mathtt{end}$, so this case is trivial.

Full preciseness Full preciseness is done using the previous simple results, and additional lemmas handling interleavings.

To obtain soundness for typing judgements involving specifications, we have to introduce *interleavings* of formulae, treating the fact that one process can play several roles in several sessions. As a simple example both $s[\mathtt{p}_1,\mathtt{p}_2]?(x).k![\mathtt{q}_1,\mathtt{q}_2]\ \langle 10\rangle$ and $k![\mathtt{q}_1,\mathtt{q}_2]\langle 10\rangle.s[\mathtt{p}_1,\mathtt{p}_2]?(x)$ can be typed with $s[\mathtt{p}_2] : \mathtt{p}_1?(x : \mathtt{Nat}).\mathtt{end}$, $k[\mathtt{q}_1] : \mathtt{q}_2!(y : \mathtt{Nat}).\mathtt{end}$.

Interleaving is not a new operator *per se* and can be seen as syntactic sugar, describing shuffling of must modalities. The main rule for interleaving is: $[\ell_1]\phi_1 \rtimes [\ell_2]\phi_2 = [\ell_1](\phi_1 \rtimes [\ell_2]\phi_2) \wedge [\ell_2]([\ell_1]\phi_1 \wedge \phi_2)$. When interleaving two or more formulae containing modalities, we obtain a conjunction of formulae, each one representing a different way of organising all modalities in a way that preserves their initial orders. Informally, the interleaving of $[1][2]$ and $[A][B]$ is $[1][2][A][B] \wedge [A][B][1][2] \wedge [1][A][2][B] \wedge [A][1][B][2] \wedge [1][A][B][2] \wedge [A][1][2][B]$.

The full rules for interleaving are given in Figure 3.

We encode a pair Δ,Γ into a complex formula $\mathtt{Inter}(\Delta,\Gamma)$, defined as the interleaving of the formulae obtained by encoding the local types of Δ on their corresponding channels and the formulae corresponding to Γ, built as follows: for each channel $a : \mathtt{I}(\mathscr{G})$, if some $s[\mathtt{p}]$ is received on a, the resulting process should satisfy the encoding on $s[\mathtt{p}]$ of the projection of $\mathscr{G}$ on $\mathtt{p}$:

$$\mathtt{Inter}(s_1[\mathtt{p}_1],\ldots,s_n[\mathtt{p}_n];a_1 : \mathtt{I}(\mathscr{G}_1),\ldots,a_m : \mathtt{I}(\mathscr{G}_m)) = \|T_1\|^{s_1[\mathtt{p}_1]} \rtimes \ldots \rtimes \|T_n\|^{s_n[\mathtt{p}_n]} \rtimes \phi_1 \rtimes \ldots \rtimes \phi_m$$

where $\phi_i = \forall s_i'.\forall \mathtt{p}_i'.[a_i(s_i'[\mathtt{p}_i])]\|\mathscr{G}_i \upharpoonright \mathtt{p}_i'\|^{s_i'[\mathtt{p}_i']}$.

Lemma 4 (Shuffling correctness)

If $P_1 \models \phi_1$ *and* $P_2 \models \phi_2$ *and if* $\mathtt{free}(\phi_1) \cap \mathtt{free}(P_2) = \mathtt{free}(\phi_2) \cap \mathtt{free}(P_1) = \mathtt{free}(P_1) \cap \mathtt{free}(P_2) = \mathtt{free}(\phi_1) \cap \mathtt{free}(\phi_2) = \emptyset$, *then* $P_1 \mid P_2 \models \phi_1 \rtimes \phi_2$.

Conversely, if $P_1 \mid P_2 \models \phi_1 \rtimes \phi_2$, *and* $\mathtt{free}(\phi_1) \cap \mathtt{free}(P_2) = \mathtt{free}(\phi_2) \cap \mathtt{free}(P_1) = \mathtt{free}(P_1) \cap \mathtt{free}(P_2) = \mathtt{free}(\phi_1) \cap \mathtt{free}(\phi_2) = \emptyset$, *then* $\mathtt{free}(\phi_1) \subseteq \mathtt{free}(P_1)$ *and* $\mathtt{free}(\phi_2) \subseteq \mathtt{free}(P_2)$.

Proof We proceed by double structural induction over the pair (ϕ_1, ϕ_2).

- The most interesting case is when both formula are modalities: $\phi_1 = [\alpha_1]\phi_1'$ and $\phi_2 = [\alpha_2]\phi_2'$. The formula $\phi_1 \rtimes \phi_2$ is $[\alpha_1](\phi_1' \rtimes \phi_2) \wedge [\alpha_2](\phi_1' \rtimes \phi_2)$. We prove that $P_1 \mid P_2$ satisfies the first formula (the other part is similar). First the condition of $\texttt{free}(P_2) \cap \texttt{free}(\phi_1)$ ensures that there is no P_2' such that $P_2 \xrightarrow{\alpha_1} P_2'$. As a consequence, if $P_1 \mid P_2 \xrightarrow{\alpha_1} P'$, it means that $P_1 \xrightarrow{\alpha_1} P_1'$. By hypothesis, $P_1' \models \phi_1'$ and we use the induction hypothesis to get $P_1' \mid P_2 \models (\phi_1' \rtimes \phi_2)$.
- The other cases are treated by destructing one construct, following the definition, and using the induction hypothesis.

Lemma 5 (Description of free names) *If* $\mathscr{C}, \Gamma \vdash P \triangleright \Delta$ *then* $\texttt{free}(P) \subseteq \texttt{free}(\Delta) \cup \texttt{free}(\Gamma)$

Easily done by induction on the typing judgement.

Lemma 6 (Nature of an interleaving)

Let $\Delta = \{s_k[\mathtt{p}_k] : \mathtt{q}_k \overset{!}{?} \{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.T_{k,i}\}_{i\in I}\}_k$ *and* $\Gamma = \{a_j : \mathtt{I}(\mathscr{G}_j)\}_j$ *be well-formed, then the formula* $\texttt{Inter}(\Delta, \Gamma)$ *is equivalent to a formula guarded by several* $\forall$ *operators guarding a conjunction of formulae, each one starting with a modality, and this modalities are in bijection with the pairs of* $(s_k[\begin{smallmatrix}\mathtt{p}_k, \mathtt{q}_k \\ \mathtt{q}_k, \mathtt{p}_k\end{smallmatrix}], l_{k,i})$ *and* $(a_j, \emptyset)$.

Proof By induction on the typing judgment:

- Case *selection*. In this case we have $P = s[\mathtt{p}, \mathtt{p}_0]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$ and $\Delta = \Delta', s[\mathtt{p}] : \mathtt{p}_0?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathscr{L}_i\}_{i\in I}$. We use Lemma 6 to state the formula we are trying to validate using P is a conjunction on several formulas, all beginning with a different modality from the pairs $(s_k[\mathtt{p}_k], l_{k,i})$ and $(a_j, \emptyset)$. As P is only able to perform an action $s[\mathtt{p}, \mathtt{p}_0]?$, all formulas starting with a modality associated to a different name are automatically satisfied, and we have to prove that for each i:

$$P, \sigma \models \mathscr{C} \implies$$

$$\|T_i\|^{s[\mathtt{p}]}, \mathtt{S} \rtimes \bigtimes_{s_k[\mathtt{p}_k]:T_k \in \Delta'} \|T_k\|^{s_k[\mathtt{p}_k], \mathscr{I}}, \mathtt{S} \rtimes \bigtimes_{a_j : \mathscr{G}_j[\mathtt{p}_j] \in \Gamma} \forall s_j.[a_j(s_j[\mathtt{p}_j])] \|\mathscr{G}_j|_{p_j}\|^{s_j[\mathtt{p}_j], \mathscr{I}}, \mathtt{S}$$

  We conclude in a way similar to the one followed in the proof of Proposition 2.
- Case *branching*. We have $P = s[\mathtt{p}, \mathtt{p}_0]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I}$. We use Lemma 6 to state the formula we are trying to validate using P is a conjunction on several formulas, all beginning with a different prefix. As P is only able to perform an action $s[\mathtt{p}, \mathtt{p}_0]!$, all formulas starting with a different modality are automatically satisfied, and we have to prove We conclude using the proof of Proposition 2.
- Case *session reception*. We have $P = a(s).P'$ and $\Gamma = a : \mathscr{G}[\mathtt{p}], \Gamma'$. We use Lemma 6 to state the formula we are trying to validate using P is a conjunction on several formulas, all beginning with a different modality. As P is only able to perform an action on a, all formulas starting with a modality associated to a different name are automatically satisfied, and we have to prove that P satisfies $\forall s[\mathtt{p}], [a(s)]\texttt{Inter}(\Gamma'; \Delta, s : \mathscr{G}|_{\mathtt{p}})$. As P is able to receive $s[\mathtt{p}]$ on a, we use the induction hypothesis to conclude.
- Case *parallel composition*. Easily done by using Lemmas 4 and 5 and the fact that both Γ and Δ are split multiplicatively in the rule for parallel composition we use.

- Case *end* is trivial.

We extend the erasing operator to Δ. Namely, $\mathbf{Er}(\Delta)$ maps $s[\mathtt{p}]$ to $\mathbf{Er}(\mathscr{L})$ iff Δ maps $s[\mathtt{p}]$ to $\mathscr{L}$. Our preciseness result is:

Proposition 7 (Preciseness) *If* $\Gamma \vdash P \triangleright \Delta$, *then:* $P, \sigma \models (\mathtt{Inter}(\Delta, \Gamma))$. *If* $\vdash P \triangleright \mathbf{Er}(\Delta)$ *and* $P, \sigma \models (\mathtt{Inter}(\Delta, \Gamma))$ *then* $\Gamma \vdash P \triangleright \Delta$

By induction on the unasserted typing judgment, case branching and selection are treated in a way similar to the proof of Proposition 3, parallel composition is done using Lemmas 4 and 5.

4 Refinements

Embedding to pure HML We are actually able to embed a stateful satisfaction relation $P, \sigma \models \phi$ into a satisfaction relation $P' \models \phi'$ for a standard π-calculus with first-order values, by encoding the store σ into a π-process:

$$\begin{array}{l} \|x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\|_{\mathtt{p}} = \quad \overline{a_1}(v_1) \mid \ldots \mid \overline{a_n}(v_n) \mid \\ \quad !x_1(e).a_1(y_1)\ldots a_n(y_n).(\overline{a_1}(\mathtt{eval}(e[y_1 \ldots y_n / x_1 \ldots x_n])) \mid \overline{a_2}(y_2) \mid \ldots \mid \overline{a_n}(y_n)) \mid \ldots \mid \\ \quad !x_n(e).a_1(y_1)\ldots a_n(y_n).(\overline{a_1}(y_1) \mid \ldots \mid \overline{a_{n-1}}(y_{n-1}) \mid \overline{a_n}(\mathtt{eval}(e[y_1 \ldots y_n / x_1 \ldots x_n]))) \end{array}$$

For each variable x_i in the domain of the state σ, we add an output prefix emitting its content on the channel a_i and we add a replicated module that waits for an update e at x_i, then capture the value of all variables of the current state, replace the variable x_i by evaluating e by `eval`, and then makes available the other ones. Soundness and completeness allow us to state that HML formulae for pairs state/process can be seen as pure HML formulas on the π-processes.

The embedding for the formula is given by

$$\begin{array}{ll} \|[E]\phi\|_{\mathtt{p}} & = [\|E\|_{\mathtt{p}}]\|\phi\|_{\mathtt{p}} \\ \|A\|_{\mathtt{p}} & = [\overline{x_1}(v_1)]\ldots[\overline{x_n}(v_n)]A\{v_1, \ldots, v_n / x_1, \ldots, x_n\} \end{array}$$

where the state variables of A are $x_1, \ldots, x_n$.

Proposition 8 (Preciseness) *If* $P, \sigma \models \phi$, *then* $\|P\|_{\mathtt{p}} \mid \|\sigma\|_{\mathtt{p}} \models \|\phi\|_{\mathtt{p}}$.
If $\|P\|_{\mathtt{p}} \mid \|\sigma\|_{\mathtt{p}} \models \|\phi\|_{\mathtt{p}}$ *then* $P, \sigma \models \phi$

Embedding Recursion Recursion can be encoded at the cost of much technical details. We add to our HML syntax the recursion operators, $\mu X.\phi$ and X (similar to the ones present in the μ-calculus [7]). The main difficulty lies in the interaction between interleaving and recursion: loops coming from different sessions can be interleaved in many different way, and the difficult task is to compute the finite formula which is equivalent to this interleaving. As a small example consider the following session environment (interactions are replaced by integer labels): $s_1[\mathtt{p}_1] : \mu X.1.2.X, s_2[\mathtt{p}_2] : \mu Y.3.4.Y$. The simplest HML formula describing all possible interleavings is:

$$\begin{array}{l} \mu A.([1]\mu B.([2]A \wedge [3]\mu C.([4]B \wedge [2]([1]C \wedge [4].A))) \wedge \\ \quad [3]\mu D.([4].A \wedge [1]\mu E.([2]D \wedge [4]([2]A \wedge [3]E)))) \end{array}$$

We use the following method to obtain a matching HML formula. We use a translation through finite automata. Here is a sketch of the method, which takes as arguments a set session environment Δ:

1. Encode every session judgement $s_i[\mathtt{p}_i] : T_i$ of Δ into a formula ϕ_i, using $\|\mu X.T\|^{s[\mathtt{p}]} = \mu X\|T\|^{s[\mathtt{p}]}$.

2. Translate every formula ϕ_i into a finite automata $\mathscr{A}_i$, one state corresponds to a syntactic point between two modalities or a μX, one transition correponds to either $[\ell](A \wedge [E]\circ)$ (output) or $[\ell](A \Rightarrow \circ)$ (input). Every automata is *directed* with a source state corresponding to the head of the formula and leaf states corresponding to recursion variables (or end of protocols).
3. Compute the automata $\mathscr{A}$, the parallel composition of all the $\mathscr{A}_i$, which is still *directed*.
4. Expand the automata $\mathscr{A}$, in order to obtain an equivalent branch automata, that is, an automata such that there is a root (the starting state) and transitions form a tree (back transitions are allowed but only on the same branch). This could be done by recursively replacing sub-automata with several copies of this sub-automata.
5. Translate back the automata into a formula, every state with more than two incoming transition is encoded as a recursion operator.

On our example, we obtain the formulas $\mu X.[1][2].X$ and $\mu Y.[3][4].Y$, each one giving an automaton with 2 states (initial and between $[1]$ (resp. $[3]$) and $[2]$ (resp. $[4]$)). Merging yields automata with 4 states: the initial one, one after $[1]$, one after $[3]$, one after both $[1]$ and $[3]$. These automata are diamond-shaped (hence not tree-shaped). Expansion yields an automaton with 7 states, which is then translated in the formula described above. The preciseness proof relies on the fact that the operation described in 3. and 4. give equivalent automata, and that two formulas translated into two equivalent automata are equivalent for the HML satisfaction relation.

5 Conclusion

Hennessy-Milner logic (HML) is a natural and semantically complete logic for processes which can immediately be applied to the distributed π-calculus in [3]. The HML with hypothetical supposition can faithfully embed the safety aspect of stateful MPSAs: at the same time, the restricted expressive power of MPSAs enables tractable dynamic and static validations. The underlying type structures and linkage among them through local state is a major reason why local types enable both static and runtime verification against rich specifications.

The work [5] investigates a relationship between a dual intuitionistic linear logic and binary session types, and shows that the former defines a proof system for a session calculus which can automatically characterise and guarantee a session fidelity and global progress. In [1], the authors introduce a state layer in a π-caclulus, toward the validation of security properties for protocols. The work [13] further extends [5] to the dependent type theory in order to include processes that communicate data values in functional languages. A recent work [12] encodes dynamic features in [8] in a dependently typed language for secure distributed programming. None of the above works treat either virtual states or logical specifications for interleaved multiparty sessions.

References

[1] Myrto Arapinis, Eike Ritter & Mark Dermot Ryan (2011): *StatVerif: Verification of Stateful Processes*. In: *CSF*, IEEE Computer Society, pp. 33–47, doi:10.1109/CSF.2011.10.

[2] Martin Berger, Kohei Honda & Nobuko Yoshida (2008): *Completeness and Logical Full Abstraction in Modal Logics for Typed Mobile Processes*. In: *ICALP (2), LNCS* 5126, Springer, pp. 99–111, doi:10.1007/978-3-540-70583-3_9.

[3] Laura Bocchi, Romain Demangeon & Nobuko Yoshida (2013): *A Multiparty Multi-session Logic*. In: *TGC, LNCS* 8191, Springer, pp. 97–111, doi:10.1007/978-3-642-41157-1_7.

[4] Laura Bocchi, Kohei Honda, Emilio Tuosto & Nobuko Yoshida (2010): *A Theory of Design-by-Contract for Distributed Multiparty Interactions*. In: *CONCUR, LNCS* 6269, pp. 162–176, doi:10.1007/978-3-642-15375-4_12.

[5] Luis Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *CONCUR, LNCS* 6269, Springer, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[6] Mario Coppo & Mariangiola Dezani-Ciancaglini (2008): *Structured Communications with Concurrent Constraints*. In: *TGC*, pp. 104–125, doi:10.1007/978-3-642-00945-7_7.

[7] Mads Dam (1994): *CTL* and ECTL* as Fragments of the Modal mu-Calculus*. *TCS* 126(1), pp. 77–96, doi:10.1016/0304-3975(94)90269-0.

[8] Pierre-Malo Deniélou & Nobuko Yoshida (2011): *Dynamic Multirole Session Types*. In: *POPL*, pp. 435–446, doi:10.1145/1926385.1926435.

[9] Edsger W. Dijkstra (1975): *Guarded commands, nondeterminacy and formal derivation of programs*. *Commun. ACM* 18, pp. 453–457, doi:10.1145/360933.360975.

[10] Matthew Hennessy & Robin Milner (1985): *Algebraic Laws for Non-Determinism and Concurrency*. *JACM* 32(1), pp. 137–161, doi:10.1145/2455.2460.

[11] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL'08*, ACM, pp. 273–284, doi:10.1145/1328438.1328472.

[12] Nikhil Swamy, Juan Chen, Cedric Fournet, Pierre-Yves Strub, Karthikeyan Bharagavan & Jean Yang (2011): *Secure Distributed Programming with Value-Dependent Types*. In: *ICFP*, ACM, pp. 266–278, doi:10.1145/2034773.2034811.

[13] Bernardo Toninho, Luis Caires & Frank Pfenning (2011): *Dependent Session Types via Intuitionistic Linear Type Theory*. In: *PPDP*, ACM, pp. 161–172, doi:10.1145/2003476.2003499.