Embedding Session Types in HML

Romain Demangeon[1] and Laura Bocchi[2]⋆

Queen Mary, University of London
romaind@eecs.qmul.ac.uk
University of Leicester
lb148@mcs.le.ac.uk

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 [2] 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 [2] 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 [6] that enables the specification of arbitrary behavioural properties of processes. Recent work on the enhancement of multiparty session types with logical annotations [2, 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 [2, 3] is based on multiparty session types [3, 4, 7] 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., a global type). Each global type is then *projected*, 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, hence 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, [3] enables the effective verification of properties on the actual values exchanged, other than the properties on the sorts of the messages guaranteed by [4, 7]. For instance, global type G in (1) describes, following a similar syntax to [4], a conversation where role $\mathtt{S}$ sends role $\mathtt{C}$ an integer and then continues as specified by global type G'. Following [3], assertion $\mathcal{G}$ in (1) can be obtained by annotating global type G; assertion $\mathcal{G}$ further prescribes that the

⋆This work has been partially sponsored by the project Leverhulme Trust award ”Tracing Networks”.

exchanged value, say y, must be greater than 10. Note that y is bound in $\mathcal{G}'$ and the fact $\{y > 10\}$ can be relied on in the subsequent interactions.

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

In [2] we extended [3] with the capability to refer to *virtual states* local to each network principal, hence expressing not only properties confined to the single multi-party 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 $\mathtt{S.x}$, a state variable x allocated to the *principal* serving as $\mathtt{S}$; and that the local effect of this message send is to increment $\mathtt{S.x}$. In this way, $\mathtt{S}$ is specified to send incremental values across *consecutive* sessions. The resulting global specifications are called *multi-party 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 [2], it is useful to relate its notion of assertions to a more standard approach in process logic. This would enable us to integrate different methods catering for different concerns, for which we may need a common logical basis. In this paper we consider HML with predicates in [1, 3], and we justify the relevance of the stateful logical layer of [2] 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 $\mathtt{S}$ 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

MPSAs. We focus here on local assertions, ranged over by $\mathcal{L}$. $\mathcal{L}$ refers to a specific role; we assume it derives, via projection, from some global assertion as in [2] – e.g., as (2).

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

Selection $\mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$ 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). A_i are predicates[1] and E_i state updates. We use guarded recursion, defining a parameter x initially set equal to a value satisfying the initialisation predicate A', with A being an invariant predicate. Judgements are of the form $\Gamma \vdash P : \Delta$ where Γ is a shared environment and Δ is the session environment (see [2] for more details).

HML. [2] uses local assertions as a basis for the verification of a processes, ranged over by P. Here, the behaviour prescribed for P is modelled using the standard HML with the first-order predicates as in [1]. 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. Namely, $P, \sigma \xrightarrow{\ell} P', \sigma'$ if either $P \xrightarrow{\ell} P'$ and $\sigma' = \sigma$ or $P = P'$ and $\sigma' = \sigma\, \mathtt{after}\, \ell$. We use ϕ to denote HML-formulae, which are built from predicates, implications, universal quantifiers, conjunctions and *must* modalities. We remark that 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.

$$\phi ::= \mathtt{true} \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \mid [\ell]\phi \mid A \mid \forall x : S.\phi \quad \ell ::= s[\mathtt{p},\mathtt{q}](x) \mid \overline{s[\mathtt{p},\mathtt{q}]}(x) \mid E$$

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

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

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

Fig. 1. Logical rules

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

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

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

The embedding of selection (4), is a conjunction of 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 (4), the assertion is used as an hypothesis and no update appear.

[1] As in [2, 3] we assume that the validity of closed formulae is decidable.

3 Soundness

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 $s[\mathtt{p}_2]\ :\ \mathtt{p}_1?(x\ :\ \mathtt{Nat}).\mathsf{end}$, $k[\mathtt{q}_1]\ :\ \mathtt{q}_2!(y:\mathtt{Nat}).\mathsf{end}$.

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 preserving their initial orders. Informally, the interleaving of $[1][2]$ and $[A][B]$ is $[1][2][A][B] \wedge [A][B][1][2] \wedge [1][A][2][B] \wedge [A][1][B][2] \wedge [1][A][B][2] \wedge [A][1][2][B]$.

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 $a : \mathtt{I}(\mathcal{G})$, if some $s[\mathtt{p}]$ is received on a, the resulting process should satisfy the encoding on $s[\mathtt{p}]$ of the projection of $\mathcal{G}$ on $\mathtt{p}$:

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

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

The erasing operator $\mathbf{Er}(\mathcal{L})$, which translates an asserted type into its unasserted counterpart is defined by the removal of every assertion from the local types. Unasserted typing rules for the judgements $\vdash P \rhd \Delta$ are easily deduced. Our preciseness result is:

Proposition 1 (Preciseness). *If* $\Gamma \vdash P \rhd \Delta$*, then:* $P,\sigma \models (\mathtt{Inter}(\Delta,\Gamma))$*. If* $\vdash P \rhd \mathbf{Er}(\Delta)$ *and* $P,\sigma \models (\mathtt{Inter}(\Delta,\Gamma))$ *then* $\Gamma \vdash P \rhd \Delta$*.*

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:

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

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 $\mathtt{eval}$, 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 $\|[E]\phi\|_{\mathtt{p}} = [\|E\|_{\mathtt{p}}]\|\phi\|_{\mathtt{p}}$ and $\|A\|_{\mathtt{p}} = [\overline{x_1}(v_1)]\ldots[\overline{x_n}(v_n)]A\{v_1,\ldots,v_n/x_1,\ldots,x_n\}$ where the state variables of A are $x_1,\ldots,x_n$.

Proposition 2 (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, absent from the previous embeddings, can actually 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 [5]). 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:

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

We will not detail here how we proceed. The idea is that we translate every session judgment into formula, and then every formula into a finite automaton where one state corresponds either to a sequence or a recursion, and where transitions correspond to inputs and outputs. The automata are then merged into a a parallel automaton, which is expanded recursively into branch automata and translated back into formulas.

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 inital one, one after [1], one after [3], one after both [1] and [3]. This automata is diamond-shaped, and, as a result, not tree-shaped. Expansion yields an automata 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 to two equivalent automata are equivalent for the HML satisfaction relation.

References

1. Martin Berger, Kohei Honda, and Nobuko Yoshida. Completeness and logical full abstraction in modal logics for typed mobile processes. In *ICALP (2)*, volume 5126 of *LNCS*, pages 99–111. Springer, 2008.
2. Laura Bocchi, Romain Demangeon, and Nobuko Yoshida. A multiparty multisession logic. (to appear in TGC), 2012.
3. Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, volume 6269 of *LNCS*, pages 162–176, 2010.
4. Mario Coppo and Mariangiola Dezani-Ciancaglini. Structured communications with concurrent constraints. In *TGC*, pages 104–125, 2008.
5. Mads Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *TCS*, 126(1):77–96, 1994.
6. Matthew Hennessy and Robin Milner. Algebraic Laws for Non-Determinism and Concurrency. *JACM*, 32(1), 1985.
7. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.