Modelling session types using contracts \textsuperscript{*} \textsuperscript{†}

Giovanni Bernardi, Matthew Hennessy
Foundations and Methods Group
School of Computer Science and Statistics
University of Dublin, Trinity College
bernargi@cs.tcd.ie, matthew.hennessy@cs.tcd.ie

ABSTRACT

Web services are one of the most widely used technologies for service-oriented computing. In particular, they support client-server protocols whose specifications are written in XML languages as such as WSCL or WSDL. Notwithstanding the wide adoption of web services, it is not yet clear which formalism should be used to reason about the protocols they support. Session types and contracts are two formalisms used to study client-server protocols, both promoted as good formal methods for web services. In this paper we study the relationship between contracts and session types. The main result is the existence of a fully abstract model of session types; this model is based on a natural interpretation of these types into a subset of contracts.

1. INTRODUCTION

Communication between processes in a distributed system often consists of a structured dialogue, following a protocol which specifies the format of the messages interchanged and, at least for binary communication, the direction of the messages. Session types, ST, have been introduced as an approach to the static analysis of the participants of such dialogues. They allow structured sequences of non-uniform messages to be interchanged between the participants. For example, using the notation of [8], the type $\langle \text{Int} \rangle$,$\langle \text{Real} \rangle$ END specifies the output of a value of type Int followed by the input of a value of type Real, after which the dialogue is terminated. Flexibility in the permitted sequencing of messages by a process is accommodated by two choice operators; the branching type $\langle T_1, T_2 \rangle$ offers a choice to the partner in the dialogue between behaving as prescribed by the type $T_1$ or as prescribed by the type $T_2$. On the other hand the choice type $\langle T_1, T_2 \rangle$ allows the process itself to behave as prescribed by either $T_1$ or $T_2$.

Sub-typing, [6], which we denote $\equiv_{ST}$, also increases the flexibility of the type system; if $T_1 \equiv_{ST} T_2$ then any participant that communicates along a channel at type $T_2$ may be used in a situation where a channel at type $T_1$ was expected. Intuitively this pre-order between session types is generated by allowing more possibilities in branching types and restricting them in choice types.

Web services are distributed components which can be combined using standard communication protocols and machine-independent message formats to provide services to clients. To encourage reusability, descriptions of their behaviour are typically made available in searchable UDDI repositories. In papers such as [11, 11, 11, 11, 11] a language of contracts has been proposed for describing this behaviour which, despite a very different surface syntax, is very similar in style to session types. In particular there is the sequencing of messages $\alpha_1, \alpha_2$, an external choice between behaviours $\sigma_1 + \sigma_2$ reminiscent of the branching type $\langle T_1, T_2 \rangle$, and an internal choice between allowed behaviours $\sigma_1 \oplus \sigma_2$, reminiscent of the choice type $\langle T_1, T_2 \rangle$.

The object of this paper is to study the precise relationship between session types and contracts for web services. In particular for first-order session types [11, 11, 11, 11, 11] we show that the theory of session types, $(ST, \equiv_{ST})$, can be captured precisely using a natural pre-order over a subclass of contracts.

Contracts for web services serve two roles. A contract $\sigma$ may describe the behaviour of a server offering some specific service. Dually a contract $\rho$ may describe the behaviour expected of a client who wishes to avail of a particular service. Central to the theory of contracts for web services is the idea of compliance between such contracts, formalised as an asymmetric relation $\rho \triangleright \sigma$; it has been defined in a variety of ways in papers such as [11, 11, 11, 11, 11]. Viewed as constraints on behaviour, session types are much more constraining than contracts; we therefore isolate a subset of contracts, which we call session contracts SC. This set is the range of the translation function $M_t$ which maps session types into contracts and provides our fully abstract model. The compliance relation leads to two natural pre-orders on session contracts:

- the server pre-order: $\sigma_1 \sqsubseteq_{\text{SC}} \sigma_2$ if for every (client) session contract $\rho$, $\rho \triangleright \sigma_1$ implies $\rho \triangleright \sigma_2$
- the client pre-order: $\rho_1 \sqsubseteq_{\text{SC}} \rho_2$ if for every (server) session contract $\sigma$, $\rho_1 \triangleright \sigma$ implies $\rho_2 \triangleright \sigma$.

These relations are unsound with respect to session sub-
typing, but by combining these pre-orders we obtain full abstraction, that is a sound and complete model for session types. The full abstraction theorem is the main result of this paper, and it shows that first-order session types can be smoothly embedded into the theory of contracts, which thus appears to be a more general formalism. This suggests that, as formal method for web services, contracts should be preferred over first-order session types. The paper is organised as follows. In the next section we give the definition of session types and the sub-typing between them; this material is taken directly from [6], although our definition is based on first-order types; however, we allow a primitive sub-typing relation between the basic types.

In the subsequent section we study contracts. Our language for contracts is a subset of the language proposed in [10]; we provide a novel co-inductive formulation of the notion of compliance; nevertheless, the resulting compliance relation coincides with that used in both in [10] and [1]. The sub-typing relation between session types is defined co-inductively and the connection we eventually make between contracts and session types will depend on co-inductive characterisations of the set-based pre-orders on contracts.

In Section 2 we focus on the subset of contracts called session contracts $SC$, this time giving co-inductive characterisations to both the restricted server pre-order $\preceq^{SC}_{srv}$ and the restricted client pre-order $\preceq^{SC}_{cst}$ over them. Due to the very restricted nature of these contracts, these co-inductive characterisations are purely in terms of their syntax.

In Section 3 we tackle the central question of the paper. Having defined the (obvious) translation of session types into session contracts, we explain why the two natural pre-orders $\preceq^{SC}_{s} \subseteq \preceq^{SC}_{c}$ are unsound relative to the sub-typing on session types. Finally, we prove that when combined they provide a sound and complete model. Thereafter we use the interpretation to show how to define a satisfactory notion of conformance between types and protocols.

The paper concludes with a brief look at related work.

2. SESSION TYPES

The syntax of terms for session types is given by the language $L_{ST}$ in Figure 1. It presupposes a denumerable set of labels $L$, ranged over by $l$, and a set of basic or ground types $BT$ types ranged over by $t$. We also use a denumerable set of variables $\text{Vars}$, ranged over by $X$, in order to express recursive types.

The use of variables leads to the usual notion of free and bound occurrences of variables in terms in the standard manner; we say that a term is closed if it contains no free variables. We also assume the standard notion of capture avoidance substitution of terms for free variables.

We use $ST$ to denote the set of closed terms with guarded recursion, and we refer to the elements in $ST$ as session types.

**Definition 2.1. Unfolding** [2] For all $T \in ST$, define $\text{unf}(T)$ as follows:

$$\text{unf}(T) = \begin{cases} \text{unf}(T \setminus \{X \mapsto T \}) & \text{if } T = \mu X. T' \\ T & \text{otherwise} \end{cases}$$

Intuitively, $\text{unf}(T)$ unfolds top-level recursive definitions until a type constructor appears, which is not $\mu$.

2.1 Sub-typing

There are three sources for the sub-typing relation over types. The first is some predefined pre-order over the basic types, $t_1 \preceq t_2$, which intuitively says that all data-values of type $t_1$ may be safely used where values of $t_2$ are expected. Throughout the paper we use the set of basic types $BT = \{\text{Bounded}, \text{Bool}, \text{Int}, \text{Real}, \text{Num}, \text{Random}\}$, with a sub-typing relation on them define as follows: $\text{Int} \preceq \text{Real} \preceq \text{Num} \preceq \text{Random}, \text{Bool} \preceq \text{Random}$. Moreover, generally, if $[t]$ denotes the set of values of the basic type $t$ then we can define $\preceq_{bg}$ by letting $t_1 \preceq_{bg} t_2$ whenever $[t_1] \subseteq [t_2]$. The other sources are two constructs of the language: the branch construct allows sub-typing by extending the set of labels involved, while in the choice construct the set of labels may be restricted. For example if $m \leq n$ we have

$$\kappa(1 : S_1, \ldots, m : S_m) \quad \text{subtype of} \quad \&\{1 : S_1, \ldots, n : S_n\}$$

$$\oplus\{1 : S_1, \ldots, m : S_n\} \quad \text{subtype of} \quad \oplus\{1 : S_1, \ldots, m : S_m\}$$

Moreover, we will have the standard co-variance/contra-variance of input/output types [17], extended to both the branch and choice constructs.

Because of the recursive nature of our collection of types, the formal definition of the sub-typing relation is given co-inductively.

**Definition 2.2. Type simulation** Let $P(X)$ denote the powerset of a set $X$, and let the function $F_{s\preceq_{s}} : P(ST^2) \rightarrow P(ST^2)$ be defined so that $(T, U) \in F_{s\preceq_{s}}(R)$ whenever one of the following holds:

1. if $\text{unf}(T) = \text{end}$ then $\text{unf}(U) = \text{end}$
2. if $\text{unf}(T) \preceq [[t_1]]; S_1$ then $\text{unf}(U) \preceq [[t_2]]; S_2$ and $(S_1, S_2) \in R$ and $t_1 \preceq_{bg} t_2$
3. if $\text{unf}(T) \preceq [[t_1]]; S_1$ then $\text{unf}(U) \preceq [[t_2]]; S_2$ and $(S_1, S_2) \in R$ and $t_2 \preceq_{bg} t_1$
4. if $\text{unf}(T) = \kappa(1 : S_1, \ldots, m : S_m)$ then $\text{unf}(U) = \kappa(1 : S'_1, \ldots, m : S'_n)$ where $m \leq n$ and $(S_i, S'_i) \in R$ for all $i \in \{1, \ldots, m\}$
5. if $\text{unf}(T) = \oplus\{1 : S_1, \ldots, m : S_m\}$ then $\text{unf}(U) = \oplus\{1 : S'_1, \ldots, n : S'_n\}$ where $n \leq m$ and $(S_i, S'_i) \in R$ for all $i \in \{1, \ldots, n\}$

A relation $R$ such that $R \subseteq F_{s\preceq_{s}}(R)$ is called type simulation. The co-inductive sub-typing relation $\preceq_{st}$ is now defined as the greatest fixed point of the equation $X = F_{s\preceq_{s}}(X)$.

In [6] the set of types $ST$ are used to give a typing system for the pi calculus, and appropriate Type Safety and Type Preservation theorems are proved. Here instead our aim is to give a model to the set of types $\langle ST, \preceq_{st} \rangle$ using contracts.
that can behave as internal fulfilment of a contract. We assume this is not in the contract process implementing the contracts – he symbol ther as successful completion of a computation. The decision can be due for instance to an the contract independently from the environment. Such a computation while an empty contract which intuitively can never be fulfilled.

We recall from the literature an example of contracts.

```
Customer = !Request.(!PayDebit.ρ ⊕ !PayCredit.ρ ⊕ !PayCash.1)
ρ' = !Long.?Bool.1
Bank = μx.?Request.(?PayCredit.?Long.?Bool.x + ?PayDebit.?Long.?Bool.x + ?PayCash.x)
```

The contracts above describe the conversation that should take place between a client (which offers the contract Customer) and a bank (which has contract Bank) involved in an on-line payment.

The conversation unfolds as follows: the Customer sends a request to the bank and afterwards it chooses the payment method; the choice is taken by an internal sum and this means that the decision of the Customer is independent from the environment (ie., the Bank contract). If the Customer decides to pay by cash then no other action has to be taken; while if the payment is done by debit or credit card the Customer has to send the card number, this is represented by the output !Long. After the card number has been received the Bank answers with a boolean. Intuitively, this represents the fact that the bank can approve or reject the payment. The Customer protocol finishes after such boolean has been received, while the bank starts anew.

### 3. CONTRACTS

#### 3.1 The contract language

A language for contracts $L_C$ is given in Figure 2. As with session types it uses a denumerable set of recursion variables $\text{Vars}$, here lower case, but also presupposes a set $\text{Act}$ of actions, ranged over by $\alpha$, which processes guaranteeing contracts may perform; as we will see the special action $\checkmark$, which we assume is not in $\text{Act}$, will be used to indicate the fulfilment of a contract.

Intuitively the contract $\alpha.\sigma$ performs the action $\alpha$ and then behaves like $\sigma$; the sum $\sigma + \sigma''$ is ready to behave either as $\sigma'$ or as $\sigma''$, and the choice depends on the external environment. For this reason the operation $+$ is called external sum. The internal sum $\sigma' \parallel \sigma''$ represents a contract that can behave as $\sigma'$ or as $\sigma''$, and the choice is taken by the contract independently from the environment. Such a decision can be due for instance to an $\textit{if}$ statement in the process implementing the contracts. The symbol $\bot$ denotes an empty contract, which intuitively can never be fulfilled, while 1 denotes the contract that is always satisfied.

An operational semantics for the closed terms of the language $L_C$ is given in Figure 3. The judgements are of the form $\sigma \xrightarrow{\alpha} \sigma'$ where $\sigma$, $\alpha$ and $\sigma'$ are closed terms of the language $L_C$. The judgement $\sigma \xrightarrow{\alpha} \sigma'$ expresses that the contract $\sigma$ is resolved to the contract $\sigma'$ by some internal computation, while $\sigma \xrightarrow{\checkmark} \sigma'$ represents the reporting of the successful completion of a computation.

Let $\xrightarrow{\top}$ denote the reflexive transitive closure of $\xrightarrow{\top}$.

We let $C$ denote the set of all terms $\sigma$ of $L_C$ which are closed, with guarded recursion, and convergent. We refer to these terms as contracts.

We recall from the literature an example of contracts.

```
Customer = !Request.(!PayDebit.ρ ⊕ !PayCredit.ρ ⊕ !PayCash.1)
ρ' = !Long.?Bool.1
Bank = μx.?Request.(?PayCredit.?Long.?Bool.x + ?PayDebit.?Long.?Bool.x + ?PayCash.x)
```

The contracts above describe the conversation that should take place between a client (which offers the contract Customer) and a bank (which has contract Bank) involved in an on-line payment.

The conversation unfolds as follows: the Customer sends a request to the bank and afterwards it chooses the payment method; the choice is taken by an internal sum and this means that the decision of the Customer is independent from the environment (ie., the Bank contract). If the Customer decides to pay by cash then no other action has to be taken; while if the payment is done by debit or credit card the Customer has to send the card number, this is represented by the output !Long. After the card number has been received the Bank answers with a boolean. Intuitively, this represents the fact that the bank can approve or reject the payment. The Customer protocol finishes after such boolean has been received, while the Bank starts anew.

#### 3.2 Client-Server interactions and the compliance relation

Contracts are expressive enough to encode XML based languages such as WS-BPEL activities and WSCL diagrams, and in it is shown how to assign contracts to a subset of CCS processes. Intuitively, if a process, such as a server, is assigned a contract $\sigma$ then it guarantees to support the behaviour described in $\sigma$. The interaction between servers and clients can be described at the level of their contracts, by defining a binary operation $\rho \parallel \sigma$ between their contracts and describing the evolution of the contracts as they interact.

This interacting semantics is given in Figure 4 where the judgements are of the form $\rho \parallel \sigma \xrightarrow{\top} \rho' \parallel \sigma'$. It presupposes a binary relation on $\text{Act}$, $\alpha \bowtie \beta$, which intuitively means that the action $\alpha$ can synchronise with action $\beta$.

The relation $\bowtie$ can be instantiated in various ways depending on the particular set of actions $\text{Act}$; the only general property we require of it is that it be $\textit{finitary}$, that is for every action $\alpha$ the set $\{ \beta \mid \alpha \bowtie \beta \}$ is finite.

For instance, to interpret session types as contracts we will define $\textit{Act}$ to be the set $\{ ?t, t! \mid t \in \text{BT} \} \cup \{ ?, 1! \mid 1 \in L \}$ with $\bowtie$ determined by

\[
\alpha \bowtie \beta \text{ whenever } \begin{cases} \alpha = ?t, \beta = t! & t' \leq_E t \\ \alpha = t!, \beta = ?t' & t \leq_E t' \\ \alpha = ?, \beta = 1 & \alpha = 1, \beta = ?1 \end{cases}
\]

Having described how interactions between clients and
servers affect their contracts, let us describe, by means of a relation, when a client (guaranteeing a contract $\rho$) can safely interact with a server (guaranteeing a) contract $\sigma$. Indeed, we shall formalise the meaning of “safely”.

The central notion is that of compliance between contracts. This is defined co-inductively and uses the predicate on contracts $\rho \xhookrightarrow{c} \sigma$ which intuitively means that the contract $\rho$ has already been satisfied. Our definition is a variation on that of compliance in [7,8,10].

**Definition 3.1.** (Compliance relation) Let $F_{\mathcal{C}} : \mathcal{P}(\mathcal{C}^2) \rightarrow \mathcal{P}(\mathcal{C}^2)$ be the function defined so that $(\rho, \sigma) \in F_{\mathcal{C}}(R)$ whenever both the following hold:

(i) if $\rho \parallel \sigma \xhookrightarrow{c} \rho'$ then $\rho \xhookrightarrow{c}$

(ii) if $\rho \parallel \sigma \xhookrightarrow{c} \rho'$, then $(\rho', \sigma) \in R$

If $R \subseteq F_{\mathcal{C}}(R)$ then we say that $R$ is a co-inductive compliance relation. Let $\downarrow$ denote the greatest solution of the equation $X = F_{\mathcal{C}}(X)$. We call this solution the compliance relation. If $\rho \vdash \sigma$ we say that the contract $\rho$ complies with the contract $\sigma$.

Notice that there is an asymmetry in the relation $\rho \vdash \sigma$; the intention is that any client running contract $\rho$ when interacting with a server running contract $\sigma$ will be satisfied, in the sense that either the interaction between client and server will go on indefinitely, or, if the interaction gets stuck, the client will end on its own in a state in which it is satisfied, $\rho \xhookrightarrow{c}$.

In order that the relation $\vdash$ captures the intuition described above, it is crucial that $\mathcal{C}$ contains no divergent terms. Had we admitted them, then for every $\rho$ the relation $\{ (\rho', \mu.x.\mu.x) \mid \rho \xhookrightarrow{c} \rho' \}$ would have been a perfectly fine co-inductive compliance. Note, though, that the client $\rho$ would by no means be satisfied by the server.

Note that according to our definition of compliance, the client need not ever perform $\checkmark$. For example, suppose $\alpha \bowtie \beta$ and consider the set $\{ (\mu.x.\alpha.x, \mu.y.\beta.y) \}$. It is a co-inductive compliance relation, and the client contract, $\mu.x.\alpha.x$, does not perform $\checkmark$ at all.

The fact that NIL can not be satisfied is formally expressed by the fact that NIL $\not\vdash \sigma$ for every contract $\sigma$. On the other hand $1$ is always satisfied because, for every $\sigma$, we have $1 \vdash \sigma$.

The compliance relation is determined purely by the client contract performing the success action $\checkmark'$, therefore $1$ complies with every contract $\sigma$; this means that a client whose contract is $1$ is satisfied by any server. On the other hand a server with contract $1$ is equivalent to a server with contract NIL.

$$
\rho, \sigma ::= 1 \mid \sum_{i \in I} \eta_i.\sigma_i \mid \bigoplus_{i \in I} \eta_i.\sigma_i \mid \eta.\sigma \mid \mu x.\sigma \mid x \mid \mu x.\sigma
$$

We impose the additional proviso that in a term the $\eta_i$’s are pair-wise different, and that the set $I$ is finite and nonempty.

**Figure 5: Session contract grammar.**

### 4. Session Contracts

Here we specialise the contract language to a sub-language which will be the target of our interpretation of the session types from Section 2. This is the topic of the first subsection. We then go on to introduce the server pre-order and to show how it applies to this sub-language; in particular we prove that it can be characterised co-inductively, by using purely syntactical criteria. In the final subsection we give a similar co-inductive characterisation to a similar pre-order, the sub-client pre-order.

#### 4.1 Session contracts

The syntax for the language $\mathcal{L}_{\text{SC}}$ is given in Figure 5. We work relative to a structural equivalence, generated by the following identities:

$$
\sigma + \rho = \rho + \sigma \quad \sigma + (\sigma' + \sigma'') = (\sigma + \sigma') + \sigma''
$$

This justifies the use of the general summation constructs for internal and external choices, which emphasises the intended restrictions in the language. We use $\mathcal{SC}$ to denote the closed terms of $\mathcal{L}_{\text{SC}}$ with guarded recursion. We refer to these terms as session contracts. Note that $\mathcal{SC}$ is a subset of the more general language of contracts $\mathcal{C}$, but external choices are restricted to inputs on labels, and internal choices are restricted to outputs on labels. We have chosen $1$ to be the base contract, for reasons which will become apparent. Moreover, we already reasoned that a server contract $1$ has the same behaviour as NIL.

#### 4.2 The server and the client pre-orders

Now we show how to use the compliance relation so as to order session contracts with respect to their capacity of satisfying clients. As proof method, we give also a co-inductive characterisation of the pre-order.

**Definition 4.1.** (Restricted server pre-order) For $\sigma_1, \sigma_2 \in \mathcal{SC}$ let $\sigma_1 \leq_{\text{sc}} \sigma_2$ whenever $\rho \vdash \sigma_1$ implies $\rho \vdash \sigma_2$ for every $\rho$ in $\mathcal{SC}$.

**Definition 4.2.** (Syntactic sub-server relation) Let $F_{\mathcal{L}}^{\text{syn}} : \mathcal{P}(\mathcal{C}^2) \rightarrow \mathcal{P}(\mathcal{C}^2)$ be defined by letting $(\sigma_1, \sigma_2) \in F_{\mathcal{L}}^{\text{syn}}(R)$ whenever one of the following holds:

(i) $\text{UNF}(\sigma_1) = 1$

(ii) $\text{UNF}(\sigma_2) = \mathcal{L}\text{UNF}(\sigma_1)$ and $\text{UNF}(\sigma_1) = \mathcal{L}\text{UNF}(\sigma_2)$

\footnote{Formally the use of these equalities is justified by the relationship between $\vdash$ and the weak bisimulation equivalence $\overset{\omega}{\approx}$.}
(iii) $\text{UNF}(\sigma_2) = t_2, \sigma_2'$ and $\text{UNF}(\sigma_1) = t_1, \sigma_1'$
with $t_2 \ll_t t_1$ and $\sigma_1' \ll_R \sigma_2'$

(iv) $\text{UNF}(\sigma_2) = \sum_{j \in J} t_j, \sigma_j$ and
$\text{UNF}(\sigma_1) = \sum_{j \in I} t_j, \sigma_j'$ with $I \subseteq J$ and $\sigma_j' \ll_R \sigma_j$.

(v) $\text{UNF}(\sigma_2) = \bigoplus_{j \in J} t_j, \sigma_j$ and
$\text{UNF}(\sigma_1) = \bigoplus_{j \in I} t_j, \sigma_j'$ with $J \subseteq I$ and $\sigma_j' \ll_R \sigma_j$.

If $R \subseteq F_{\ll_R} (\mathcal{R})$ then we say that $R$ is a co-inductive syntactic sub-server relation. Let $\ll_{\text{syn}}$ denote the greatest solution of the equation $X = F_{\ll_{\text{syn}}} (X)$. We call this solution the syntactic sub-server.

We can prove the following result.

**Theorem 4.3.** (Co-inductive characterisation) For session contracts $\sigma_1, \sigma_2, \sigma_1 \ll_{\text{syn}} \sigma_2$ if and only if $\sigma_1 \ll_{\text{syn}} \sigma_2$.

Now we introduce a second pre-order which compares the capacity of clients to be satisfied by servers. We proceed as for the previous pre-order.

**Definition 4.4.** (Restricted client pre-order) For $\rho_1, \rho_2 \in \mathcal{S}$ let $\rho_1 \ll_{\text{clt}} \rho_2$ whenever $\rho_1 \vdash \sigma$ implies $\rho_2 \vdash \sigma$ for every $\sigma \in \mathcal{S}$.

The co-inductive characterisation is given through the following co-inductive relation.

**Definition 4.5.** (Syntactic sub-client relation) Let the function $F_{\ll_{\text{clt}}} : \mathcal{P}(\mathcal{S}) \rightarrow \mathcal{P}(\mathcal{S})$ be defined so that $(\rho_1, \rho_2) \in F_{\ll_{\text{clt}}} (\mathcal{R})$ whenever one of the following is true:

(i) $\text{UNF}(\rho_2) = 1$

(ii) $\text{UNF}(\rho_2) = t_2, \rho_2'$ and $\text{UNF}(\rho_1) = t_1, \rho_1'$
with $t_2 \ll_t t_1$ and $\rho_2' \ll_R \rho_1'$

(iii) $\text{UNF}(\rho_2) = t_2, \rho_2'$ and $\text{UNF}(\rho_1) = t_1, \rho_1'$
with $t_2 \ll t_1$ and $\rho_2' \ll \rho_1'$

(iv) $\text{UNF}(\rho_2) = \sum_{j \in J} t_j, \rho_j$ and
$\text{UNF}(\rho_1) = \sum_{j \in I} t_j, \rho_j'$ with $I \subseteq J$ and $\rho_j' \ll_R \rho_j$.

(v) $\text{UNF}(\rho_2) = \bigoplus_{j \in J} t_j, \rho_j$ and
$\text{UNF}(\rho_1) = \bigoplus_{j \in I} t_j, \rho_j'$ with $J \subseteq I$ and $\rho_j' \ll_R \rho_j$.

If $R \subseteq F_{\ll_{\text{clt}}} (\mathcal{R})$ then we say that $R$ is a co-inductive syntactic sub-client relation. Let $\ll_{\text{clt}}$ denote the greatest solution of the equation $X = F_{\ll_{\text{clt}}} (X)$. We call this solution the sub-client relation.

**Theorem 4.6.** (Co-inductive characterisation) Let $\rho, \sigma$ be session contracts. Then $\rho \ll_{\text{clt}} \sigma$ if and only if $\rho \ll_{\text{clt}} \sigma$.

## 5. MODELLING SESSION TYPES

The interpretation of session types as contracts is expressed as a function from the language $L_{\text{ST}}$ in Section 3 to the language $L_{\text{SC}}$ in Section 4. Let $M : L_{\text{ST}} \rightarrow L_{\text{SC}}$ be defined as in Figure 6. It is easy to see that $M$ maps session types, $\mathcal{S}$, to session contracts, $\mathcal{S}$; indeed it defines a bijection between these sets, we can prove that $M$ is injective and surjective. Further, the function $M$ preserves substitution and commutes with $\text{UNF}(\cdot)$.

$$
\begin{align*}
M(S) = \begin{cases}
1 & \text{if } S = \text{END} \\
!t.M(S) & \text{if } S = \lceil t \rceil; S \\
?t.M(S) & \text{if } S = \lfloor t \rfloor; S \\
\sum_{i \in [1,n]} t_i.M(S_i) & \text{if } S = \langle 1 \ldots , x \rangle \\
\bigoplus_{i \in [1,n]} t_i.M(S_i) & \text{if } S = \langle 1 \ldots , x \rangle
\end{cases}
\end{align*}
$$

**Figure 6:** Interpretation function.

The difficulty now is to find a natural pre-order on session contracts which accurately reflects the sub-typing relation on session contracts. There are two obvious candidates, the restricted server pre-order and the restricted client pre-order on session contracts. The difficulty lies in the interpretation of END.

Recall that $M(\text{END}) = 1$. In the restricted server pre-order the session contract $1$ is a least element, being smaller or equal to every other session contract. On the other hand, for session types $\text{END} \ll_{\text{ST}} T$ if and only if $\text{UNF}(T) = \text{END}$. Consequently the relation $\ll_{\text{ST}}$ is an unsound model for sub-typing between session types. For example:

$$1 \ll_{\text{ST}} \text{!t.1, END} \ll_{\text{ST}} \text{!t.1, END}$$

The restricted client pre-order presents the dual issue as it relates every session contract to $1$: it is one of the top element. Once again a model based on $\ll_{\text{ST}}$ would be unsound:

$$\text{!t.1} \ll_{\text{ST}} 1, \text{END} \ll_{\text{ST}} \text{END}$$

The main result of the paper is that the bijection $M$ gives a fully abstract interpretation of sub-typing between session types in terms of session contracts, provided we combine these two set-based pre-orders.

**Definition 5.1.** (Session contract pre-order) For $\sigma_1, \sigma_2 \in \mathcal{S}$ let $\sigma_1 \ll_{\text{ST}} \sigma_2$ whenever $\sigma_1 \ll_{\text{ST}} \sigma_2$.

**Theorem 5.2.** (Full abstraction) For all session types, $T_1 \ll_{\text{ST}} T_2$ if and only if $M(T_1) \subseteq M(T_2)$.

In [3], by using the fully abstract interpretation, we have shown how to formalize the intuitive idea that session types conform to communication protocols.

## 6. CONCLUSIONS

In this paper we have used contracts [5] to give a fully abstract model for first-order session types ordered by their sub-typing relation [6]. This was achieved by identifying a subset of the standard language of contracts, $L_{\text{SC}}$ which we call session contracts. These are ordered using a combination of two natural pre-orders [1], defined in terms of a contracts role in constraining the behaviours of servers and clients respectively. We believe that our work provides the
first fully abstract model of session types in terms of contracts.

The restriction to first-order session types is a severe limitation on our results, and we intend to extend our results to the full language of session types in [6].

Our compliance relation coincides with the notion of strong compliance given there, although the formulation is different. Comparison with earlier work, [7] is complicated by the fact that in these papers compliance judgements take the form \( t_1[\rho] \vdash \perp t_2[\sigma] \) where \( t_1, t_2 \) are finite sets of actions representing in some sense the interfaces of the processes guaranteeing the contracts; moreover, for a contract \( t[\sigma] \) to be valid its interface \( i \) has to contain all the action names that appear in the behaviour \( \sigma \). Let us refer to these pairs \( t[\sigma] \) as constrained contracts. Using the obvious notation for the compliance relations between constrained contracts one can show if \( \rho, \sigma \) are in \( C \) then

- if \( t[\rho] \vdash \perp t[\sigma] \) for some \( i, j \) then \( \rho \vdash \sigma \)
- if \( t[\rho] \vdash \perp t[\sigma] \) for some \( i, j \) then \( \rho \vdash \sigma \)

under the assumption that in our definition of compliance the synchronisation relation \( \bowtie \) is suitably defined. Moreover, it is easy to provide counter examples to the converse of both these points.

Our research has been greatly influenced by the work in [1]. In that paper the focus is the set of session behaviours, which is a proper subset of contracts and a proper superset of our session contracts; using this set the authors provide a sound model for sub-typing on session types. They use an interpretation function \( [\cdot] \) from session types to session behaviours which, in general, is not invertible. For instance: \( i_{\text{Int}} \cdot 1 + ?1_{1} \cdot 1 \) is a session behaviour that has no corresponding session type; this because \( i_{\text{Int}} \cdot 1 \) is a base type while \( 1 \cdot 1 \) is a label. Note, though, that \( [\cdot] = M \cdot 1 \), so the range of \( [\cdot] \) is the set of session contracts and our Theorem [5.2] proves that \( [\cdot] \) provides a complete model, as conjectured in [1].

Indeed, their approach is very similar to ours, in that they provide a co-inductive characterisation of the intersection of the sub-server and sub-client pre-orders over session behaviours. In contrast, we have studied the individual pre-orders independently.

Finally in [8] two interpretations, \( M_{1} \) and \( M_{\text{nil}} \), similar to our \( M \), are given for pairs of session types into pairs of constrained contracts. Their proposed full abstraction result, Theorem 2 of [8], though, appears not to be true; what corresponds to our server pre-order in their paper is denoted by \( \preceq \) and is defined in their Definition 2. According to that definition and the interpretation \( M_{\text{nil}} \)

\[
\emptyset[\text{NIL}] \preceq \langle \ell \rangle[\ell, \text{NIL}] 
\]

Their Theorem 2 therefore implies \&\( \langle \ell : \text{END} \rangle \preceq_{\text{ST}} \text{END} \), which is obviously not true. On the other hand if \( M_{1} \) is used then there are two issues. According to Theorem 2 the pair \( \langle \text{END}, \&\( \langle \ell : \text{END} \rangle \rangle \) is interpreted as \( \emptyset[\emptyset, \ell, \text{NIL}] \). Then

(a) neither \( \emptyset[\emptyset, \text{NIL}] \) nor \( \langle \ell \rangle[\ell, \emptyset, \text{NIL}] \) are constrained contracts, because their interfaces do not contain all the action names which appear in the respective behaviours; moreover

(b) even if the interpretation was correct, Theorem 2 would be false because

\[
\{ \ell \}[\ell, \emptyset, \text{NIL}] \preceq \{ \ell \}[\emptyset, \emptyset, \text{NIL}]
\]

while, as stated above, \&\( \{ \ell : \text{END} \} \) \preceq_{\text{ST}} \text{END} \) is not true.

7. REFERENCES


