Compliance of binary session contracts

Franco Barbanera[1]

Department of Mathematics and Computer Science
University of Catania
Catania, Italy

Ugo de'Liguoro[2]

Department of Computer Science
University of Torino
Torino, Italy

Abstract

The relation of compliance formalises the notion of client satisfaction in client/server interactions. We describe three weakenings of such a relation in the context of *session contracts*, a session-based restriction of the theory of contracts.

Keywords: Contracts, sessions, reversible computations, orchestration.

1 Introduction

Session types and contracts are two formalisms used to study client/server protocols. Session types have been introduced in [13] as a tool for statically checking safe message exchanges through channels. Contracts, on the other hand, as proposed in [10,14,11], are a subset of CCS without τ, that address the problem of abstractly describing behavioural properties of systems by means of process algebra. In between these two formalisms lie *session contracts*[3] as introduced in [4,3,8,9]; this is a formalism interpreting the session types into a subset of contracts.

In the theory of contracts, as well as in the formalism of session contracts, the notion of *compliance* plays a central role. A client ρ is defined as being compliant with a server σ (written as $\rho \dashv \sigma$) whenever *all* of its *requests* are satisfied by the

[1] Email: barba@dmi.unict.it

[2] Email: ugo.deliguoro@unito.it

[3] They were dubbed *session behaviours* in [4,3]. For sake of uniformity and since *session contract* sounds more appealing, we adhere here to this name.

server. In this note we survey on some recent results of ours concerning weaker notions of compliance that are more permissive (and hence stronger than) the original concept, although such concepts remain decidable.

2 Compliant session contracts

Session contracts are expressions generated by the grammar:

$$\sigma, \rho ::= \mathbf{1} \mid \sum_{i\in I} a_i.\sigma_i \mid \bigoplus_{i\in I} \overline{a}_i.\sigma_i \mid x \mid \mathsf{rec}\ x.\sigma$$

where the a_i and $\overline{a}_i$ are pairwise distinct input/output actions, and σ is not a variable in $\mathsf{rec}\ x.\sigma$.

We take the equi-recursive standpoint and consider any expression of the form $\mathsf{rec}\ x.\sigma$ as identical to $\sigma\{\mathsf{rec}\ x.\sigma/x\}$. The capabilities of session contract expressions are defined via the LTS:

$$\sum_{i\in I} a_i.\sigma_i \xrightarrow{a_i} \sigma_i \qquad \bigoplus_{i\in I} \overline{a}_i.\sigma_i \longrightarrow \overline{a}_j.\sigma_j \qquad \overline{a}.\sigma \xrightarrow{\overline{a}} \sigma$$

where the simple arrow represents an internal action; so $+$ and $\oplus$ are external and internal choice from CCS without τ, respectively. Finally the communication semantics is given by:

$$\frac{\rho \xrightarrow{\alpha} \rho' \qquad \sigma \xrightarrow{\overline{\alpha}} \sigma'}{\rho \| \sigma \longrightarrow \rho' \| \sigma'} \qquad \frac{\rho \longrightarrow \rho'}{\rho \| \sigma \longrightarrow \rho' \| \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\rho \| \sigma \longrightarrow \rho \| \sigma'}$$

where α ambiguously represents an input or an output capability and $\overline{\overline{a}} = a$. Note that the parallel composition $\|$ is not in the syntax of session contracts, and it is essentially a testing operator.

Definition 2.1 A *client* ρ and a *server* σ are *compliant*, written $\rho \dashv \sigma$, if

$$\forall \rho', \sigma'.\quad \rho\|\sigma \xrightarrow{*} \rho'\|\sigma' \not\longrightarrow \quad \Longrightarrow \quad \rho' = \mathbf{1}$$

where $\xrightarrow{*}$ is the reflexive and transitive closure of $\longrightarrow$ and $\not\longrightarrow$ is its complement.

Observe that the interaction among ρ and σ needs not to terminate for $\rho \dashv \sigma$ to hold; the simplest case is that of $\mathsf{rec}\, x.\ a.x \dashv \mathsf{rec}\, x.\ \overline{a}.x$. However compliance is a decidable relation:

Theorem 2.2 *For any ρ and σ it is decidable whether $\rho \dashv \sigma$.*

3 Three weakenings of compliance

3.1 Skipping server outputs

Consider a ballot service whose behaviour is described by the following contract:
$\mathsf{BallotServiceAB} \triangleq \mathsf{rec}\, x.\ \mathtt{Login}.(\overline{\mathtt{Wrong}}.x \oplus \overline{\mathtt{Overload}}.x \oplus \overline{\mathtt{Ok}}.(\mathtt{VoteA} + \mathtt{VoteB}))$.
This service can receive a login from a client, a voter, via the input action $\mathtt{Login}$; if the login is correct the server issues to the client the message $\overline{\mathtt{Ok}}$ enabling the client to vote for either candidates A or B via a continuation consisting of the external choice $+$ of the input actions $\mathtt{VoteA}$ and $\mathtt{VoteB}$. In case the login is incorrect or the service is busy, the messages $\overline{\mathtt{Wrong}}$ or $\overline{\mathtt{Overload}}$ are sent to the client respectively. In both cases the voter is allowed to retry the login by recursion.

On the other hand consider the following client:

$$\mathsf{Voter} \triangleq \mathsf{rec}\, x.\ \overline{\mathtt{Login}}.(\mathtt{Wrong}.x + \mathtt{Overload}.x + \mathtt{Ok}.\overline{\mathtt{VoteA}}).$$

Then $\mathsf{Voter} \dashv \mathsf{BallotServiceAB}$ even if Voter is not the dual of $\mathsf{BallotServiceABC}$. However Voter is not compliant with

$$\begin{array}{l}\mathsf{BallotServiceBehSkp} \ \triangleq \\ \quad \mathsf{rec}\, x.\, \mathtt{Login}.(\overline{\mathtt{Wrong}}.\overline{\mathtt{InfoW}}.x \oplus \overline{\mathtt{Overload}}.x \oplus \overline{\mathtt{Ok}}.\overline{\mathtt{Id}}.(\quad \mathtt{VoteA}.(\mathtt{Va1} + \mathtt{Va2}) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + \mathtt{VoteB}.(\mathtt{Vb1} + \mathtt{Vb2})\ )\ )\end{array}$$

because of the actions $\overline{\mathtt{InfoW}}$ and $\overline{\mathtt{Id}}$, representing infos about the failure of the login and an identifier of the voting transaction, respectively. Such outputs, however, have hardly any control significance, and could be discarded without compromising the client-server interaction. Let us then modify the communication LTS by adding the rule:

$$\frac{\rho \not\Downarrow a \quad \sigma \xrightarrow{\overline{a}} \sigma'}{\rho \| \sigma \xrightarrow{\text{skp}} \rho \| \sigma'}$$

where $\rho \not\Downarrow a$ means that there is no ρ' such that $\rho \xrightarrow{*}\xrightarrow{a} \rho'$, to prevent that the skipped output $\overline{a}$ from the server has a meaningful match on the client side.

Definition 3.1 The session contracts ρ and σ are *skip-compliant* if they satisfy the same condition as in definition 2.1 plus the condition that any infinite reduction of $\rho \| \sigma$ under the relation $\rightarrow \cup \xrightarrow{\text{skp}}$ is not definitely an infinite reduction under $\xrightarrow{\text{skp}}$ alone.

Theorem 3.2 ([2]) *The relation $\dashv^{\mathsf{skp}}$ admits a sound and complete axiomatisation which is decidable.*

3.2 Sessions that rollback

In [5] we have considered an extension of the contract syntax and semantics by allowing both server and client to rollback to certain marked points; this rollback that synchronously must happen on both sides, is caused by an internal move of either agents. A different motivation for rolling back is to recover from a failure:

$$\begin{array}{rcl}\mathsf{Buyer} & \triangleq & \overline{\mathsf{bag}}.\mathsf{price}.(\overline{\mathsf{card}} \oplus \overline{\mathsf{cash}}) \oplus \overline{\mathsf{belt}}.\mathsf{price}.(\overline{\mathsf{card}} \oplus \overline{\mathsf{cash}}) \\ \mathsf{Seller} & \triangleq & \mathsf{bag}.\overline{\mathsf{price}}.(\mathsf{card} + \mathsf{cash}) + \mathsf{belt}.\overline{\mathsf{price}}.\mathsf{cash}\end{array}$$

where a buyer is willing to purchase either a bag or a belt, and to decide about the payment only after knowing the price. But the seller doesn't provide payment by card after choosing a belt, so that $\mathsf{Buyer} \not\dashv \mathsf{Seller}$. In [6] we extend the contract syntax by an external sum of outputs $\sum_{i \in I} \overline{a}_i.\sigma_i$ and treat both forms of external choice *retractable*, in the sense that if the reduction out of $\rho \| \sigma \xrightarrow{*} \rho' \| \sigma'$ gets stuck because no communication is possible among ρ' and σ', and $\rho' \neq \mathbf{1}$, then the interaction synchronously rollbacks to the latest external choice.

In [6] we define a notion of reduction among pairs of contracts of the shape $\gamma \prec \sigma$ where γ is a stack recording the discarded branches of the past choices in the interaction leading to σ; then a compliance relation $\dashv^{\mathsf{rbk}}$ is defined as in definition 2.1 but w.r.t. the new reduction relation.

Theorem 3.3 *There exists a sound and complete axiomatisation of the relation $\dashv^{\mathsf{rbk}}$, together with an algorithm deciding derivability in the formal system; hence $\dashv^{\mathsf{rbk}}$ is decidable.*

3.3 Orchestrated compliance

It might be the case that client satisfaction cannot be achieved because of just a difference in the order in which the partners exchange information, or because one of them provide some extra un-needed information.

Consider the example of a meteorological data processing system (MDPS) that is permanently connected to a weather station to which it sends, according to its processing needs, particular data requests. After the requests, the MDPS expects to receive the data in the order they were requested:

$$\mathsf{MDPS} \triangleq \mathsf{rec}\, x\,.\,\overline{\mathtt{tempReq}}\,.\,\overline{\mathtt{humReq}}\,.\,\mathtt{temperature}\,.\,\mathtt{humidity}\,.\,x$$

Assume a weather station to be able to send back the asked-for information in the order decided by its sensors, interspersed with information about *wind speed*:

$$\begin{aligned}\mathsf{WeatherStation} \triangleq \mathsf{rec}\, x\,.\,\mathtt{tempReq}\,.\,\mathtt{humReq}\,.(&\overline{\mathtt{temperature}}\,.\,\overline{\mathtt{humidity}}\,.\,\overline{\mathtt{wind}}\,.\,x\\ &\oplus\\ &\overline{\mathtt{humidity}}\,.\,\overline{\mathtt{temperature}}\,.\,\overline{\mathtt{wind}}\,.\,x)\end{aligned}$$

With the standard notion of compliance, it is not difficult to check that $\mathsf{MDPS} \not\dashv \mathsf{WeatherStation}$, since the client MDPS has no input action for the wind data, and since it could occur that the temperature and humidity data are delivered in a different order than expected by MDPS.

A natural solution allowing permutation of input/output action consists of devising a process that could act as a mediator, called *orchestrator* by Padovani [15]. However, due to known results about communicating finite state machines (see e.g. [12]), the resulting compliance relation would be undecidable without suitable constraints to the orchestrators, e.g. they have bounded buffering capacity in Padovani's work. Nonetheless right the example above shows that no bounded buffer would contain the unbounded number of copies of the message $\overline{\mathtt{wind}}$ from the weather station.

In [1] we consider orchestrators with unbounded buffers, whose syntax, with respect to [15], is restricted in order to enforce the fact that, in any session, nondeterminism is due just to the client or the server. We then manage to show that it is decidable to check whether a client ρ is compliant with a server σ by means of a given orchestrator (denoted by $f : \rho \dashv\!\mid \sigma$).

Theorem 3.4 *Given ρ,σ and f, it is decidable to check whether $f : \rho \dashv\!\mid \sigma$ holds.*

Even more, it is possible to synthesize the set of orchestrators, if any, which can make a given ρ compliant with a given σ. However, not all the synthesized orchestrators are necessarily *respectful*, i.e. produce *reasonable* interactions. A **respectful** orchestrator must be sound (it never sends an element to a server or to a client if the element has not been previously received); client-respectful (all outputs from the client must be eventually delivered); and not definitely server-inputted (it does not just accepts inputs from the server indefinitely from a certain point on). All the above properties making an orchestrator respectful are decidable.

Theorem 3.5 *It is decidable to check whether a given orchestrator f is respectful.*

4 On-going work

In [7] a two-players game-theoretic interpretation on event structures is provided for client-server systems of session contracts. Starting from an observation by Bartoletti, we are currently providing a three-players game interpretation of retractable contracts, according to which the retractable actions correspond to moves of a third player, whose goal is having the first player win. Such a goal resembles that of an orchestrator. In fact we are also working on showing that the winning strategies for the third player in the above mentioned interpretation are in one-to-one correspondence with compliance-enabling orchestrators for client-server systems of session contracts (where just particular input-output actions can actually be orchestrator-driven).

References

[1] Barbanera, F., and U. de' Liguoro, *Orchestrated compliance for session-based client/server interactions*, in: *ICE 2015*, EPTCS (2015), to appear.

[2] Barbanera, F. and U. de' Liguoro, *Loosening the notions of compliance and sub-behaviour in client/server systems*, in: Proceedings 7th *ICE 2014*, EPTCS **166**, 2014, pp. 94–110.

[3] Barbanera, F. and U. de' Liguoro, *Sub-behaviour relations for session-based client/server systems*, Math. Struct. in Comp. Science (2014), to appear, published online.

[4] Barbanera, F. and U. de'Liguoro, *Two notions of sub-behaviour for session-based client/server systems*, in: *PPDP* (2010), pp. 155–164.

[5] Barbanera, F., M. Dezani-Ciancaglini and U. de' Liguoro, *Compliance for reversible client/server interactions*, in: *BEAT*, EPTCS **162**, 2014, pp. 35–42.

[6] Barbanera, F., M. Dezani-Ciancaglini, I. Lanese and U. de' Liguoro, *Retractable contracts*, in: *PLACES*, EPTCS (2015), to appear.

[7] M. Bartoletti, T. Cimoli, G.M. Pinna and R. Zunino, *Contracts as games on event structures*, Accepted for publication in JLAMP.

[8] Bernardi, G. and M. Hennessy, *Modelling session types using contracts*, in: *Proceedings of 27th Annual ACM SAC '12* (2012), pp. 1941–1946.

[9] Bernardi, G. and M. Hennessy, *Modelling session types using contracts*, Math. Struct. in Comp. Science (2014), to appear, published online.

[10] Carpineti, S., G. Castagna, C. Laneve and L. Padovani, *A formal account of contracts for Web Services*, in: *WS-FM*, number 4184 in LNCS (2006), pp. 148–162.

[11] Castagna, G., N. Gesbert and L. Padovani, *A theory of contracts for web services*, ACM Trans. on Prog. Lang. and Sys. **31** (2009), pp. 19:1–19:61.

[12] Cécé, G. and A. Finkel, *Verification of programs with half-duplex communication*, Inf. Comput. **202** (2005), pp. 166–190.
URL http://dx.doi.org/10.1016/j.ic.2005.05.006

[13] Honda, K., V. T. Vasconcelos and M. Kubo, *Language primitives and type disciplines for structured communication-based programming*, in: *ESOP*, LNCS **1381** (1998), pp. 22–138.

[14] Laneve, C. and L. Padovani, *The Must Preorder Revisited: An Algebraic Theory for Web Services Contracts*, in: *CONCUR'07*, LNCS **4703** (2007), pp. 212–225.

[15] Padovani, L., *Contract-Based Discovery of Web Services Modulo Simple Orchestrators*, Theoretical Computer Science **411** (2010), pp. 3328–3347.