Science of Computer Programming 109 (2015) 61–95

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Vicious circles in contracts and in logic ☆

Massimo Bartoletti [a,*], Tiziana Cimoli [a], Paolo Di Giamberardino [a], Roberto Zunino [b]

[a] Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari, Italy
[b] Dipartimento di Matematica, Università degli Studi di Trento, Italy

ARTICLE INFO

Article history:
Received 17 February 2014
Received in revised form 5 December 2014
Accepted 19 January 2015
Available online 7 February 2015

Keywords:
Contracts
Propositional contract logic
Event structures

ABSTRACT

Contracts are formal promises on the future interactions of participants, which describe the causal dependencies among their actions. An inherent feature of contracts is that such dependencies may be circular: for instance, a buyer promises to pay for an item if the seller promises to ship it, and vice versa. We establish a bridge between two formal models for contracts, one based on games over event structures, and the other one on Propositional Contract Logic. In particular, we show that winning strategies in the game-theoretic model correspond to proofs in the logic.

1. Introduction

Contracts are formal descriptions of the behaviour of programs or services, used in the software development process to maintain consistency between specification and implementation. Contracts have recently gained an increasing relevance in the design and implementation of concurrent and distributed systems, as advocated e.g. by the OASIS reference model for service-oriented architectures [1]. This interest is witnessed by the proliferation of formal systems for contracts that appeared in the literature in the last few years. Such formalisms have been devised by adapting and extending models of concurrent systems, such as Petri nets [2,3], event structures [4,5], process algebras [6–10], timed automata [11,12], and by extending various logics, such as modal [13], intuitionistic [14,15], linear [14], and deontic [16,17] logics (just to cite a few recent approaches). On a more applied side, tools have been developed to put some of such formalisms in practice. For instance, Scribble [18] can be used to specify the overall interaction protocol (named *choreography*) of a distributed application formed by a set of communicating services. By projecting such a choreography on each of these services [10], we obtain the specification of the interaction behaviour expected from each single service involved in the application. These projections have the form of *session types* [19,20], i.e. contracts which regulate the inputs/outputs each service is expected to perform. Scribble allows the designer of a distributed application to verify properties of the choreography (e.g. the absence of deadlocks), and to automatically construct monitors ensuring that each service participating in the application respects its contract [21].

A main motivation for using contracts lies in that large distributed applications are typically constructed by composing services published by different organizations. The larger an application is, the greater the probability that some of its components deviates from the expected behaviour (either because of unintentional bugs, or maliciousness of a competing

☆ Work partially supported by Aut. Region of Sardinia under grants L.R.7/2007 CRP-17285 (TRICS) and P.I.A. 2010 "Social Glue", by MIUR PRIN 2010-11 project "Security Horizons", and by EU COST Action IC1201 "Behavioural Types for Reliable Large-Scale Software Systems" (BETTY).

* Corresponding author at: Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari, via Ospedale 72, 09124 Cagliari, Italy.
E-mail address: bart@unica.it (M. Bartoletti).

http://dx.doi.org/10.1016/j.scico.2015.01.005
0167-6423/

organisation). Hence, it becomes crucial to protect oneself from other services' misbehaviour. Standard enforcement mechanisms (e.g., execution monitoring, static analysis, and model checking techniques) do not apply because of the total lack of control on code run by mutually untrusted, distributed services. Contracts may offer protection by legally binding services to behave as prescribed, or otherwise be blamed for a contract breach [22].

When contracts are used to support the reliability of distributed applications, the choice of the contract model (i.e. the formalism used to express and reason about contracts) is critical; just to give an example, assuming or not the trustworthiness of service brokers [23] (i.e. the entities which compose services) affects the kind of properties enforceable in different contract models [24]. To drive software architects in the choice of the most suitable contract model for novel applications, it would be desirable to clearly establish the actual properties and the relations among different models. This is not an easy task, because the constellation of formalisms proposed in the literature is wide and heterogeneous, and most works focus on studying a single formalism, rather than developing comparisons between different ones.

1.1. Vicious circles in contracts and in logic

In this paper, we relate two recent models for contracts — one based on game-theoretic notions and the other one on logic.

The game-theoretic model we consider was originally introduced in [24], and it is based on event structures (ES) [25]. The model assumes a set of *participants* (abstractions of services), who promise through their contracts to perform some *events*. These promises can be formalised as *enablings* of the form $X \vdash e$, meaning that an event e must be performed once all the events in the set X have been performed. For instance, consider a system composed by two participants, Alice (A) and Bob (B), associated with two events a and b, respectively, and assume that A is promising to fire a, while B is promising to fire b only after a has happened. Such promises are formalised by two ES with the following enablings:

$$\emptyset \vdash a \qquad \{a\} \vdash b \tag{1}$$

Contracts also define the goals of participants, in terms of a *payoff function* which associates each sequence of events with one of three outcomes: positive, negative, or neutral. In our Alice–Bob example, the payoff of A is positive in the sequences where she obtains b (i.e. b, ab, ba), negative in those where a is done while b is not, and neutral in the empty sequence. The payoffs of B are similar, with the role of a and b inverted.

The interaction among participants is modelled as a concurrent game [26], where participants "play" by performing events, trying to reach their goals. Each participant plays according to some strategy; roughly, a strategy is *winning* for a participant A if it allows A to fulfil her goals (i.e. reach a positive payoff) in all possible plays conforming to such strategy, and such that the other participants have no *obligations* (i.e. enabled events not yet performed). In our Alice–Bob example (1), a possible strategy for A would be just to fire a, and one for B would be to fire b once a has happened. These strategies are winning for both A and B: roughly, the only play conforming to such strategies where no one has pending obligations is ab, and in such play both participants have reached their goals.

Two key notions in this model are *agreement* and *protection*. Intuitively, a set of contracts has the agreement property whenever each participant has a winning strategy. Instead, protection is the property of the contract of a single participant A ensuring that, whenever she interacts with others (possibly adversaries), she has at least one strategy guaranteeing non-negative payoffs in plays without pending promises. The contracts in (1) admit an agreement, but A is *not* protected by her contract. Indeed, when the contract of A interacts with a contract which promises nothing, in all the plays where A has no pending promises (i.e. where she has performed a) her payoff is negative, regardless of her strategy.

When promises are modelled as Winskel's ES [25], it is shown in [24] that agreement and protection cannot hold at the same time. To give an intuition of why they are mutually exclusive, consider the following variation of the Alice–Bob contracts: A, which was not protected in (1), now requires b to happen before a.

$$\{b\} \vdash a \qquad \{a\} \vdash b \tag{2}$$

The contracts in (2) protect both A and B, but they do *not* enjoy the agreement. Indeed, the only admissible play in (2) is the empty one, because of the *vicious circle* determined by the mutual dependency between the two events a and b.

The other model of contracts [15] we consider in this paper is based on intuitionistic propositional logic (IPC): promises are represented by logical formulae, and obligations are derived through the entailment relation of the logic. The kind of vicious circles described in (2) can also happen in the logic-based model. For instance, consider the following formula of intuitionistic propositional logic (IPC):

$$(a \to b) \wedge (b \to a) \tag{3}$$

which says that b is provable provided that one has a proof of a, and *vice versa*, a is provable through a proof of b. Because of the mutual dependency between a and b, it turns out that from (3) neither a nor b can be deduced in the proof system of IPC.