Contract Automata with Reparations[1]

Shaun AZZOPARDI [a] Gordon J. PACE [a] Fernando SCHAPACHNIK [b]
[a] {*shaun.azzopardi.10,gordon.pace*}*@um.edu.mt*
University of Malta, Malta
[b] *fschapachnik@dc.uba.ar*
Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Abstract. Although contract reparations have been extensively studied in the context of deontic logics, there is not much literature using reparations in automata-based deontic approaches. Contract automata is a recent approach to modelling the notion of contract-based interaction between different parties using synchronous composition. However, it lacks the notion of reparations for contract violations. In this article we look into, and contrast different ways reparation can be added to an automaton- and state-based contract approach, extending contract automata with two forms of such clauses: catch-all reparations for violation and reparations for specific violations.

Keywords. Deontic Logic, Normative Systems, Interactive Systems, Contracts

1. Introduction

Contracts are agreements between two or more parties that regulate their behaviour by specifying what each party should or can do at each step of some process. An essential element of any contract specification logic is thus the notion of reparation — what should be done in the case of a violation of a part of the contract. In particular, in the context of deontic logics, the notion of *contrary-to-duty* (CTD) dealing with what happens when an obligation clause is violated has been extensively studied in the literature [4].

Compared to logic-based approaches, automata-based ones typically provide a less structured, yet more operational mechanisms for specification and analysis. In this paper we look into different ways of adding reparations to graph-based contract representations focusing, in particular, on contract automata [6]. We identify two different approaches to adding reparations: (i) by partitioning transitions into those which can be taken when the contract is respected and others which are taken upon a violation; and (ii) by tagging transitions identifying which norms have to be respected and/or violated for the transition to be enabled.

The approaches are illustrated using an airline check-in desk case study extended from [3]. Due to lack of space, we only outline the formalisation of the different semantics, giving just sufficient details to enable the comparison of approaches from the perspective of tractability of verification.

[1] Partially supported by UBACyT 20020130200032BA.

2. Background

Due to space reasons, we will only give a brief overview of contract automata — full details can be found in [6]. The behaviour of parties in contract automata is modelled by multi-action automata, which are synchronously composed to model interaction.

Definition 1. *A multi-action automaton is a tuple $S = \langle \Sigma, Q, q_0, \rightarrow \rangle$, where Σ is the alphabet of actions, Q is the set of states, $q_0 \in Q$ is the initial state and $\rightarrow \subseteq Q \times 2^{\Sigma} \times Q$ is the transition relation. We will write $q \xrightarrow{A} q'$ for $(q, A, q') \in \rightarrow$, and $acts(q)$ to be the set of all action sets on the outgoing transitions from q, defined to be $\{A \mid \exists q' \cdot q \xrightarrow{A} q'\}$.*

For two automata $S_i = \langle \Sigma_i, Q_i, q_{0_i}, \rightarrow_i \rangle$, $i \in \{1, 2\}$, their synchronous composition, over alphabet G, is written $S_1 \|_G S_2 \equiv \langle Q_1 \times Q_2, (q_{0_1}, q_{0_2}), \rightarrow \rangle$, where $\rightarrow$ is the classical synchronous composition relation defined below:

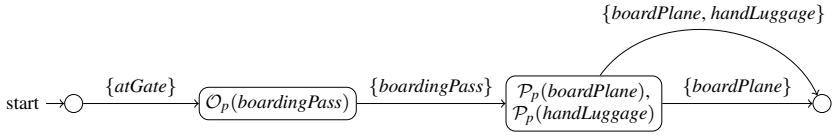
$$\frac{(q_1 \xrightarrow{A}_1 q_1')}{(q_1, q_2) \xrightarrow{A} (q_1', q_2)} \; A \cap G = \varnothing \qquad \frac{(q_2 \xrightarrow{A}_2 q_2')}{(q_1, q_2) \xrightarrow{A} (q_1, q_2')} \; A \cap G = \varnothing$$

$$\frac{(q_1 \xrightarrow{A}_1 q_1'), (q_2 \xrightarrow{B}_2 q_2')}{(q_1, q_2) \xrightarrow{A \cup B} (q_1', q_2')} \; A \cap G = B \cap G \neq \varnothing$$

Contract automata are then multi-action[2] automata with states tagged with sets of clauses ranging over obligations and permissions to perform or not an action: $\mathcal{O}_p(a)$ is the obligation on party p to perform action a, while $\mathcal{O}_p(!a)$ is the obligation not to perform a (hence a prohibition). Permission is similarly written as $\mathcal{P}_p(x)$. The type *Clause* thus contains terms of the form $\mathcal{O}_p(x)$ or $\mathcal{P}_p(x)$, where x is a possibly negated action a or $!a$.

Definition 2. *A contract automaton is a total and deterministic multi-action automaton $S = \langle \Sigma, Q, q_0, \rightarrow \rangle$, together with a total function contract $\in Q \rightarrow 2^{Clause}$. $\rightarrow$ is a subset of $Q \times 2^{\Sigma} \times Q$ and is total i.e. $\forall q \in Q, A \in 2^{\Sigma} \cdot \exists q' \in Q$ s.t. $q \xrightarrow{A} q'$.*

Below is a contract automaton[3] modelling the clause: *"When the passenger arrives at the gate he must present his boarding pass. After that he can board at any time. He is allowed one piece of hand luggage."*



Contract automata are closed under synchronous composition, which acts as a form of conjunction. The overall behaviour of a regulated system consists of the behaviour of the parties synchronously composed with a contract automaton, which allows for a contract satisfaction predicate to be defined at each state.

Definition 3. *A regulated two-party system synchronizing over the set of actions G is a tuple $R = \langle S_1, S_2 \rangle_G^{\mathscr{A}}$, where $S_i = (\Sigma_i, Q_i, q_{0_i}, \rightarrow_i)$ is a multi-action automaton specify-*

[2] Otherwise concurrent obligations can never be satisfied [6].

[3] Throughout the paper, we leave out transitions going to a sink state used to make the automata shown total as required.

ing the behaviour of party i, and $\mathscr{A}$ is a contract automaton. The behaviour of a regulated two-party system R*, written* $[\![\mathrm{R}]\!]$*, is defined to be the automaton* $(S_1 \|_G S_2) \|_\Sigma \mathscr{A}$.

An action set *A* is said to be viable for party p in state $q_{\mathscr{A}}$ of contract automaton $\mathscr{A}$, written *viable*$_p(q_{\mathscr{A}}, A)$, if it contains all the obliged actions but no forbidden ones. Based on this, we can define contract satisfaction as follows[4]:

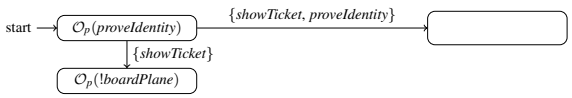
- An obligation for party p to do an action a is satisfied by p by a transition, written $sat_p^{\mathcal{O}}(q \xrightarrow{A} q')$, if it includes a in all of the outgoing transitions of p. However, the other party, noted $\overline{p}$, must also allow a to be performed.
- A party p always respects its own permission to do an action. However, the permission for a party p to perform a can be violated by $\overline{p}$ if it does not provide a viable transition that includes a. A party satisfies the permissions at a state $(q_1, q_2, q_{\mathscr{A}})$ of the regulated two-party system, written as $sat_{\overline{p}}^{\mathcal{P}}((q_1, q_2, q_{\mathscr{A}}))$, if it respects all the permissions of shared actions of the other party, by providing a viable action set.

We can extend the operational semantics of contract automata by tagging transitions with the compliance state of each party: $\xrightarrow[(\psi,\,\psi')]{A}$ where ψ and ψ' are one of $\checkmark$, $\times$ and R indicating satisfaction, violation without the possibility of reparation and violation but going to a reparation state respectively[5].

Definition 4. *A state of a regulated system and action set pair (q, A) is said to be violating for party p, written $viol_p(q, A)$ if the state or the transition do not satisfy the active contract clauses: $\neg(\mathrm{sat}_p^{\mathcal{P}}(q) \wedge \mathrm{sat}_p^{\mathcal{O}}(q) \wedge \forall q' \cdot \mathrm{sat}_p^{\mathcal{O}}(q \xrightarrow{A} q'))$. We write $viol(q, A)$ to denote that either party has violated the contract: $viol_p(q, A) \vee viol_{\overline{p}}(q, A)$. We define $\delta^{\psi}_{\psi'}(p, q, A)$ to be ψ' if $viol_p(q, A)$, and ψ otherwise. The extended operational semantics for a transition $q \xrightarrow{A} q'$ tagged with violation information is $q \xrightarrow[(\delta^{\checkmark}_{\times}(1,q,A),\, \delta^{\checkmark}_{\times}(2,q,A))]{A} q'$.*

3. Handling Reparations in Contract Automata

Since the semantics of contract automata continue enforcing a contract even after violation, they can simulate reparations in a limited way — consider the contrary-to-duty clause: *"The passenger is obliged to show a means of identification when presenting the ticket, and would otherwise be prohibited from boarding"*. This can be partially emulated using the contract automaton shown below.



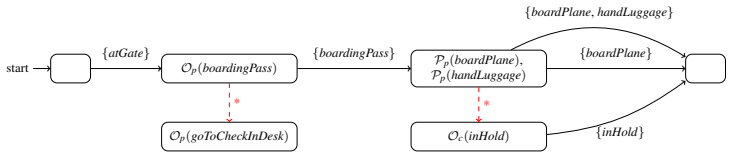
[4]Formal definitions of these notions can be found in [6]. To build an intuition it can be considered that obligation for a requires a to be in every outgoing transition, while permission requires it to be in at least some of them. In both cases the action set that contains a cannot include forbidden actions.

[5]Although we have no notion of reparation yet, we introduce this state to be used in the following section.

However, using this approach does not distinguish between whether the passenger chose not to present a means of identification (and hence the airline can apply the reparation) or whether the airline never even gave the option to the passenger to show the means of identification (by not providing a synchronising action). We thus (i) have no notion of which transitions are violating ones, and (ii) cannot express a reparation of a permission in this manner. These limitations indicate the need for reparation to be provided as a first class notion in contract automata.

Reparation Automata Reparations are transitions conditionally taken upon contract violation which can only be detected upon combining the system behaviours. Our first extension to contract automata hinges on this distinction, providing means of specifying two types of transitions — reparation ones which are taken if a violation has taken place and is to be repaired, and normal ones which are otherwise taken.

Consider a contract which states that: *(i) The passenger is obliged to present his boarding pass or would otherwise be obliged to go back to the check-in desk; after which (ii) he is permitted to board the plane with hand-luggage but if stopped from doing so, the airline company is obliged to put his hand-luggage in the hold and allow him to board.* The reparation automaton for this agreement is given in the figure below — red dashed edges are used to identify reparation transitions:[6]



Definition 5. *A* reparation automaton *is a contract automaton with two transition relations $\rightarrow_N$ (normal) and $\rightarrow_R$ (reparation), each a subset of $Q \times 2^{\Sigma} \times Q$. While the normal relation is to be total and deterministic as in the case of contract automata, the reparation one needs not to be total but must be deterministic. We will write hasRep(q, A) if for some state q' there is a reparation transition $q \xrightarrow{A}_R q'$.*

We can now define the tagged operational semantics of a regulated two party system using the following rules (we write q and q' to denote the combined states $(q_1, q_2, q_{3_{\mathscr{A}}})$ and $(q_1', q_2', q_{3_{\mathscr{A}}}')$ respectively):

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{A}_N q_{3_{\mathscr{A}}}'}{q \xrightarrow[(\checkmark,\checkmark)]{A} q'} \neg viol(q, A)$$

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{A}_R q_{3_{\mathscr{A}}}'}{q \xrightarrow[(\delta_R^{\checkmark}(1,q,A),\ \delta_R^{\checkmark}(2,q,A))]{A} q'} viol(q, A)$$

[6] An asterisk $*$ on a transition is used to denote that any action set not matching any other outgoing transition from the source state would follow this transition. Formally, it would be a set of transitions, one for each uncatered for action set.

$$\frac{(q_1,q_2) \xrightarrow{A} (q_1',q_2'),\ q_{3_{\not A}} \xrightarrow{A}_N q'_{3_{\not A}}}{q \xrightarrow[(\delta^{\checkmark}_{\times}(1,q,A),\ \delta^{\checkmark}_{\times}(2,q,A))]{A} q'} \mathit{viol}(q,A) \wedge \neg \mathit{hasRep}(q,A)$$

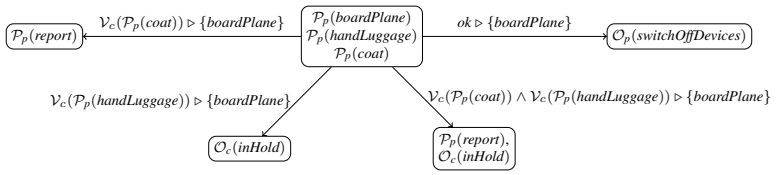
Unlike contract automata, reparation automata are not closed under synchronous composition, since composition may lead to non-determinism. This happens when a state has more than one active clause since it is impossible to distinguish which of these has been violated. Extended reparation automata, address this issue.

Extended Reparation Automata Sometimes one must identify which of the clauses were violated, as each may be repaired in a different way. For instance, if *the passenger is (i) permitted to have one piece of hand luggage, but if not allowed on board, the crew is obliged to send it in the hold; and (ii) he is also permitted to have a coat, but if not allowed on board, he may report the issue.* To be able to identify which clauses the reparation is related to, one can tag reparation transitions with the contract clauses the reparation addresses.

The solution we adopt is to have transitions tagged not only with an action set, but also with an expression specifying which clauses were violated (or not) by a party p ($\mathcal{V}_p(c)$ and $\neg\mathcal{V}_p(c)$):

$$\text{VExp} ::= \mathcal{V}_{\text{Party}}(\mathit{Clause}) \mid \neg\text{VExp} \mid \text{VExp} \wedge \text{VExp}$$

Recall the reparation automaton given in the example discussed earlier — in which we could only define one reparation transition for the action *boardPlane*, even though violation may happen in multiple ways (the passenger was not allowed the hand luggage, or the coat, or both). This can be modelled using extended reparation automata:[7]



Definition 6. *An extended reparation automaton is a contract automaton where the transition relation is augmented with a boolean expression over clause violation:* $\rightarrow \subseteq Q \times \mathit{VExp} \times 2^{\Sigma} \times Q$. *Totality and determinism of the transition relation is still required — for any action set* A*, the disjunction of the violation expressions on transitions tagged by* A *must be a tautology and any two such expressions must be mutually exclusive.*

The tagged semantics of extended reparation automata is defined as follows[8]:

$$\frac{(q_1,q_2) \xrightarrow{A} (q_1',q_2'),\ q_{3_{\not A}} \xrightarrow{ok \triangleright A} q'_{3_{\not A}}}{q \xrightarrow[(\checkmark,\checkmark)]{A} q'} (q,A) \vdash \mathit{ok}$$

$$\frac{(q_1,q_2) \xrightarrow{A} (q_1',q_2'),\ q_{3_{\not A}} \xrightarrow{V \triangleright A} q'_{3_{\not A}}}{q \xrightarrow[(\delta^{\checkmark}_{\text{R}}(1,q,A),\ \delta^{\checkmark}_{\text{R}}(2,q,A))]{A} q'} V \neq \mathit{ok},\ (q,A) \vdash V$$

[7] The expression *ok* is used to denote that no clauses in the source state are violated by either party.

[8] We write $(q, A) \vdash V$ to denote that $V \in \text{VExp}$ is satisfied when action set A is taken from state q.

The totality of the transition relation means that no transitions are unmitigatable violating ones. We can address this by having a violation state, in which violations with no reparation are sent to, or by modifying the semantics of extended reparation automata to allow for partial transitions and treat missing transitions or the transition (ok, A) as a catch-all when A takes place and no other transition is activated. The semantics adopted, however, are more compositional and thus preferred. The ability to differentiate whether or not a norm is violated, ensures closure under synchronous composition.

4. Discussion and Related Work

We have presented two approaches to adding reparations to contract automata. Reparation automata provide rudimentary means of enabling branching upon a violation as part of a contract. For reasonably sophisticated reparations, however, one needs stronger means to be able to identify which clauses have been violated and by which party, which can be done using extended reparation automata which allow branching on conditions identifying which clauses have been violated. Of the two approaches, extended reparation automata handle better the case study. Extended reparation automata can deal with states with multiple clauses effectively, but employing this increases the size of the automata and makes the representation of larger contracts difficult to generate and understand.

Reparations, although generally well understood on more structured logical approaches e.g. [4, 5] are generally more challenging in graph-based approaches. For isntance, *C-O diagrams* [2] are a graphical contract visualization framework that although being able to cope with violations, cannot consider what action set was responsibly for the violation (similar to our reparation automata). Taking a more pragmatic approach, contract automata for service contracts presented in [1] detect violations at runtime, but do not have a formalisation that allows reasoning about them.

References

[1] Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari. Automata for service contracts. In *Hot Issues in Security Principles and Trust 2014*, 2014.

[2] Gregorio Díaz, María Emilia Cambronero, Enrique Martínez, and Gerardo Schneider. Specification and Verification of Normative Texts using C-O Diagrams. *IEEE Transactions on Software Engineering*, 99:1, 2013.

[3] Stephen Fenech, Gordon J. Pace, and Gerardo Schneider. Automatic Conflict Detection on Contracts. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC'09)*, volume 5684 of *Lecture Notes in Computer Science*. Springer Verlag, 2009.

[4] Guido Governatori and Antonino Rotolo. Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. In *The Australasian Journal of Logic*, volume 4, pages 193–215, 2006.

[5] Jaap Hage. Contrary to Duty Obligations — A Study in Legal Ontology. In *Legal Knowledge and Information Systems (JURIX 2001)*, December 2001.

[6] Gordon J. Pace and Fernando Schapachnik. Contracts for Interacting Two-Party Systems. In *Proceedings of Sixth Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'12)*, volume 94 of *Electronic Proceedings in Theoretical Computer Science*, 2012.