TECHNICAL REPORT

Report No. CS2014-01
Date: October 2014

# Extending Contract Automata with Reparations

Shaun Azzopardi
Gordon J. Pace
Fernando Schapachnik

Department of Computer Science
University of Malta
Msida MSD 06
MALTA

Tel: +356-2340 2519
Fax: +356-2132 0539
http://www.cs.um.edu.mt

# Extending Contract Automata with Reparations

Shaun Azzopardi
Department of Computer Science
University of Malta
Msida, Malta

Gordon J. Pace
Department of Computer Science
University of Malta
Msida, Malta

Fernando Schapachnik
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 are 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 different ways reparation can be added to an automaton-based contract approach, extended contract automata with three forms of such clauses: (i) catch-all reparations for violation; (ii) reparations for specific violations; and (iii) hierarchical contracts for local reparation reasoning.*

# Extending Contract Automata with Reparations[*]

Shaun Azzopardi
Department of Computer Science
University of Malta
Msida, Malta

Gordon J. Pace
Department of Computer Science
University of Malta
Msida, Malta

Fernando Schapachnik
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 are 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 different ways reparation can be added to an automaton-based contract approach, extended contract automata with three forms of such clauses: (i) catch-all reparations for violation; (ii) reparations for specific violations; and (iii) hierarchical contracts for local reparation reasoning.*

## 1 Introduction

Contracts are legal agreements between two or more parties that regulate their behaviour within some context by specifying what each party should or can do at each part of some process. An essential element of any contract specification logic is the notion of reparation — what should be done in the case of the 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 [GR06].

Compared to logic-based approaches, automata-based ones typically provide a less structured, yet more operational mechanisms for specification and analysis. A number

[*]Partially supported by UBACyT 20020130200032BA

of automata-based proposals have also been developed for deontic reasoning [PS12, DCMS13]. Although some of these frameworks do allow for reparations, their current scope is limited and a general understanding of the design-space for dealing with reparations in an graph or automaton-based approach is still lacking. As opposed to structured deontic logics, in which reparations are usually related to sub-expressions, the lack of structure in graph-based representations presents challenges as to how reparations should be integrated.

In this paper we look into different ways of adding reparations to graph-based contract representations focussing, in particular, on contract automata [PS12]. Contract automata regulate interactivity between parties based on synchronous composition. Explicitly modelling interactivity allows richer reasoning on norm satisfaction: e.g., the obligation of a passenger to put his luggage in the cargo of a plane can be violated if the passenger does not hand his luggage, but also if the crew does not accept it.

We identify three different approaches to adding reparations as part of the contract descriptions. The first uses a catch-all approach to violation handling, partitioning transitions into those which can be taken when the contract is respected and others which are taken upon a violation, thus acting as a form of reparation. Although resulting in clean and simple specifications, this lacks the ability to identify which norms have been violated. This is addressed in the second approach, in which transitions are tagged with expressions identifying which norms have to be respected and/or violated for the transition to be enabled. Finally, we present a more structured hierarchical approach inspired by notations such as Statecharts [MLS97], which allows for clustering sub-graphs for common and complex reparations in a structured manner.

The approaches are illustrated using an airline check-in desk case study [Azz14], extended from [FPS09]. Due to lack of space, we only outline the formalisation of the different semantics, giving just sufficient details to enable comparing the approaches from the perspective of tractability of verification. In Section 2 we recall the fundamentals of contract automata necessary for the rest of the paper. We then present three different approaches to extending contract automata with reparations in Section 3, which are discussed together with other approaches from the literature in Section 4.

## 2 Background

In this section we give an overview of contract automata (see [PS12] full details), a model that enables reasoning about the interactivity between parties in the context of a contract. The behaviour of each party 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 $\Sigma$ 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:*

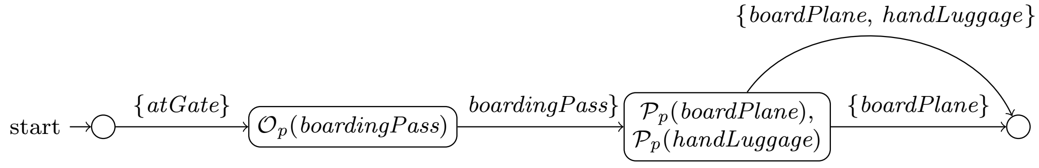
$$\frac{(q_1 \xrightarrow{A}_1 q_1')}{(q_1, q_2) \xrightarrow{A} (q_1', q_2)}\ A \cap G = \varnothing \qquad \frac{(q_2 \xrightarrow{A}_2 q_2')}{(q_1, q_2) \xrightarrow{A} (q_1, q_2')}\ A \cap G = \varnothing$$

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

Contract automata are then multi-action[1] 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[2] 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.*


start
{atGate}
$\mathcal{O}_p(boardingPass)$
boardingPass}
$\mathcal{P}_p(boardPlane)$, $\mathcal{P}_p(handLuggage)$
{boardPlane}
{boardPlane, handLuggage}

[1]Otherwise concurrent obligations can never be satisfied [PS12].

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

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.

**Definition 3.** *A regulated two-party system synchronizing over the set of action sets $G$ 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 specifying 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* $(S_1 \|_G S_2) \|_\Sigma \mathscr{A}$.

## 2.1 Satisfaction

Before defining satisfaction of a contract we consider what it means for an action set to be *viable* i.e., contain all obliged but no prohibited actions.

**Definition 4.** *An action set $A$ is viable for party $p$ in state $q_{\mathscr{A}}$, written* $\mathrm{viable}_p(q_{\mathscr{A}}, A,)$, *if it contains all the obliged actions but no forbidden ones.*

$$\begin{aligned}&\mathrm{viable}_p(q_{\mathscr{A}}, A, \ ) = \\ &\quad \{a \mid \mathcal{O}_p(a) \in contract(q_{\mathscr{A}})\} \subseteq A \wedge \{a \mid \mathcal{O}_p(!a) \in contract(q_{\mathscr{A}})\} \cap A = \varnothing\end{aligned}$$

**Obligations**: An obligation for party $p$ to do an action $a$ is satisfied by $p$ if it includes $a$ in all of the outgoing transitions of $p$. However $\overline{p}$ must also allow $a$ to be performed:

$$\begin{aligned}&sat_p^{\mathcal{O}}((q_1, q_2)_{q_{\mathscr{A}}} \xrightarrow{A} (q_1', q_2')_{q'_{\mathscr{A}}}) = viable_p(q_{\mathscr{A}}, A) \\ &sat_{\overline{p}}^{\mathcal{O}}((q_1, q_2)_{q_{\mathscr{A}}}) = \exists A \in acts(q_{\overline{p}}), A' \subseteq G^c \cdot viable_p(q_{\mathscr{A}}, A \cup A')\end{aligned}$$

**Permissions**: 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 $a$.

$$\begin{aligned}&(q_1, q_2)_{q_{\mathscr{A}}} \vdash_{\overline{p}} \mathcal{P}_p(a) \equiv \\ &\qquad a \in G \implies \exists A \in acts(q_{\overline{p}}), A' \subseteq G^c \cdot a \in A \wedge viable_p(q_{\mathscr{A}}, A \cup A')\end{aligned}$$

Thus, a party satisfies the permissions at a state if it respects all the permissions of shared actions of the other party, by providing a viable action set.

$$sat_{\overline{p}}^{\mathcal{P}}((q_1, q_2)_{q_{\mathscr{A}}}) = \forall \mathcal{P}_{\overline{p}}(a) \in acts(q_{\mathscr{A}}) \cdot (q_1, q_2)_{q_{\mathscr{A}}} \vdash_p \mathcal{P}_{\overline{p}}(a)$$

## 2.2 Compliance, Violation and Culpability

We can extend the operational semantics of contract automata by tagging transitions with the compliance state of each party: $\xrightarrow[(\psi, \psi')]{A}$ where $\psi$ and $\psi'$ are one of ✓, $\times$ and R indicating satisfaction, violation without the possibility of reparation and violation but going to a reparation state respectively[3].

**Definition 5.** *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 did not satisfy the active contract clauses:*

$$viol_p(q, A) \equiv \neg(\mathrm{sat}_p^{\mathcal{P}}(q) \wedge \mathrm{sat}_p^{\mathcal{O}}(q) \wedge \forall q' \cdot \mathrm{sat}_p^{\mathcal{O}}(q \xrightarrow{A} q'))$$

*We will write $viol(q, A)$ to denote when either party has violated the contract: $viol_p(q, A) \vee viol_{\overline{p}}(q, A)$. By defining $\delta^{\psi}_{\psi'}(p, q, A)$ to be $\psi'$ if $viol_p(q, A)$ and $\psi$ otherwise, we can define the extended operation semantics: for a transition $q \xrightarrow{A} q'$, we can deduce $q \xrightarrow[(\delta^{\checkmark}_{\times}(1,q,A),\, \delta^{\checkmark}_{\times}(2,q,A))]{A} q'$.*

# 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.

$$\begin{tikzcd}
\text{start} \arrow[r] & \boxed{\mathcal{O}_p(proveIdentity)} \arrow[r, "\{showTicket,\, proveIdentity\}"] \arrow[d, "\{showTicket\}"] & \boxed{\phantom{xxxxxxxx}} \\
& \boxed{\mathcal{O}_p(!boardPlane)} &
\end{tikzcd}$$

This approach, however, has a number of drawbacks. Firstly, we have no implicit notion of which transitions are violating ones from the automaton. Secondly, although obligation and prohibition can be emulated in this manner, there is no way we can express a reparation in the case of a permission. Finally, the approach is only partial,

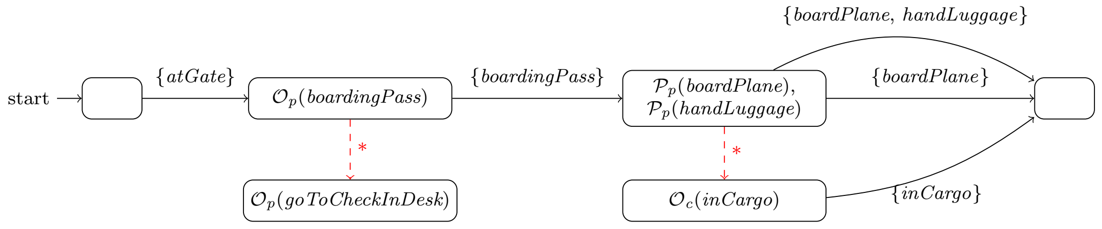
[3]Although we have no notion of reparation yet, we introduce this possible state to be used in the following section.

in that it 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). These limitations indicate the need for reparation to be provided as a first class notion in contract automata. In the rest of this section, we present three different forms of reparation extensions to these automata.

## 3.1 Reparation Automata

Reparations are transitions conditionally taken upon contract violation which can only be detected upon combining the system behaviours. The 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 reparated, 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:[4]


start
$\{atGate\}$
$\mathcal{O}_p(boardingPass)$
$\{boardingPass\}$
$\mathcal{P}_p(boardPlane)$, $\mathcal{P}_p(handLuggage)$
$\{boardPlane,\ handLuggage\}$
$\{boardPlane\}$
$*$
$*$
$\mathcal{O}_p(goToCheckInDesk)$
$\mathcal{O}_c(inCargo)$
$\{inCargo\}$

**Definition 6.** *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 need not be total but must be deterministic. We will write $hasRep(q, A)$ if for some state $q'$ there is a reparation transition $q \xrightarrow{A}_R q'$.*

[4]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.

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):

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

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

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{A}_N q_{3_{\mathscr{A}}}'}{q \xrightarrow[(\delta_{\times}^{\checkmark}(1,q,A),\, \delta_{\times}^{\checkmark}(2,q,A))]{A} q'}\ \mathit{viol}(q, A) \land \neg \mathit{hasRep}(q, A)$$

Reparation automata enable the description of reparations but are still limited in a number of ways. Unlike contract automata, they are not closed under synchronous composition, since non-determinism can result from the composition. This happens when a state has more than one active clause since it is impossible to distinguish which of these has been violated. This is one of the major shortcomings of reparation automata, which we seek to address in extended reparation automata.

## 3.2 Extended Reparation Automata

Some situations require identifying which combination of clauses was violated, as each may be reparated 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 as cargo; 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 ($\mathcal{V}_p(c)$ and $\neg\mathcal{V}_p(c)$):

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

To illustrate the use of this approach, recall the reparation automaton given in the example discussed earlier. Reparation automata can only define one reparation transition for the action *boardPlane*, however a party could be in violation in multiple ways, i.e., either the passenger was not allowed the hand luggage, or the coat, or both. This can be modelled using extended reparation automata as shown below:[5]

$$\begin{tikzcd}
\boxed{\mathcal{P}_p(\mathit{report})} & & \boxed{\begin{array}{c}\mathcal{P}_p(\mathit{boardPlane})\\ \mathcal{P}_p(\mathit{handLuggage})\\ \mathcal{P}_p(\mathit{coat})\end{array}} \arrow[ll, "\mathcal{V}_c(\mathcal{P}_p(\mathit{coat})) \triangleright \{\mathit{boardPlane}\}"'] \arrow[rr, "\mathit{ok} \triangleright \{\mathit{boardPlane}\}"] \arrow[dl, "\mathcal{V}_c(\mathcal{P}_p(\mathit{handLuggage})) \triangleright \{\mathit{boardPlane}\}"'] \arrow[dr, "\mathcal{V}_c(\mathcal{P}_p(\mathit{coat})) \wedge \mathcal{V}_c(\mathcal{P}_p(\mathit{handLuggage})) \triangleright \{\mathit{boardPlane}\}"] & & \boxed{\mathcal{O}_p(\mathit{switchOffDevices})} \\
 & \boxed{\mathcal{O}_c(\mathit{inCargo})} & & \boxed{\begin{array}{c}\mathcal{P}_p(\mathit{report}),\\ \mathcal{O}_c(\mathit{inCargo})\end{array}} &
\end{tikzcd}$$

**Definition 7.** *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.*

We can now give the tagged semantics of extended reparation automata as follows (we write $(q, A) \vdash V$ to denote that violation expression $V \in \mathrm{VExp}$ is satisfied when action set $A$ is taken from state $q$):

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{\mathit{ok} \triangleright A} q_{3_{\mathscr{A}}}'}{q \xrightarrow[(\checkmark,\checkmark)]{A} q'}\ (q, A) \vdash \mathit{ok}$$

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{V \triangleright A} q_{3_{\mathscr{A}}}'}{q \xrightarrow[(\delta_{\mathrm{R}}^{\checkmark}(1,q,A),\ \delta_{\mathrm{R}}^{\checkmark}(2,q,A))]{A} q'}\ V \neq \mathit{ok},\ (q, A) \vdash V$$

The totality of the transition relation means that no transitions are unmitigatable violating ones. We can address this problem by having a violation state, in which violations with no reparation are sent to. Alternatively, one could modify the semantics of extended reparation automata to allow for partial transitions and treat

[5]The expression *ok* is used to denote that none of the clauses in the source state are violated by either of the parties.

missing transitions or the transition $(ok, A)$ as a catch-all when $A$ takes place and no other transition is activated. The semantics given above are, however, more compositional and thus preferred. Given that one can now differentiate between a norm being violated or not, extended reparation automata are closed under synchronous composition.

## 3.3 Hierarchical Reparation Automata

One aspect of logics lost when one uses automata is that of structure. For instance, the reparation for an airline not to allow a passenger to board might be to have to offer the passenger overnight accommodation, and (later) book them on the next available flight. By looking at the automata describing this behaviour, it is not possible to know whether the reparation is the overnight accommodation offer or whether it extends to the booking on the next available flight or whether it extends even further. A standard way of introducing structure and nesting in an automaton-based formalism is to use *hierarchical automata* (e.g. see [MLS97]) to be able to encapsulate a whole automaton in a single state, resulting in different levels of behavioural detail thus giving a means of grouping behaviour together.

We have used such a nested approach in *hierarchical reparation automata*, in which any state can be refined into two automata — the first giving the contract in force (without reparations) and the second being another contract which will be triggered if any part of the first contract is violated. This nesting can be done to any arbitrary depth to enable the description of reparations triggered in case a reparation itself is violated.

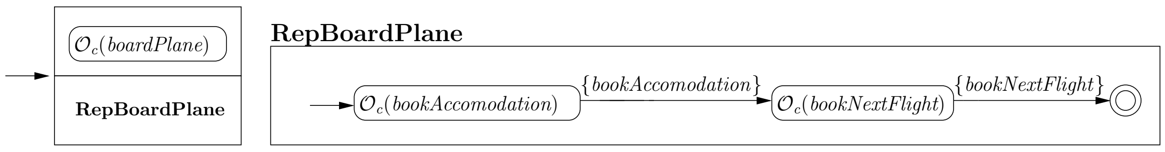
**Definition 8.** *A* hierarchical reparation automaton *$H \in \mathcal{H}$ consists of (i) a multi-action automaton $S = \langle \Sigma, Q, q_0, \rightarrow \rangle$, with: (i) a state partition $Q = Q_0 \cup Q_N$ (with $Q_0 \cap Q_N = \varnothing$) where $Q_0$ are the normal states and $Q_N$ are the nested states; (ii) a set of final states $F \subseteq Q$; (iii) a set of clauses associated to each normal state $contract \in Q_0 \rightarrow 2^{Clause}$; and (iv) two functions used to access the normal contract and the reparation automata in a nested state: $nrm, rep \in Q_N \rightarrow \mathcal{H}$.*

The configuration of a hierarchical reparation automaton consists of a stack of pairs of states and automata specifying where the contract currently lies. In an automaton $H$, the stack just contains $(H,\ q_0(H))$ (which we write as $conf_0(H)$) and advances by (i) moving along the top automaton if no violation is detected, also removing the item from the stack if a final state is reached or adding an item if the state moved into is a nested one; or (ii) moving to the initial state of the reparation automaton of the next item on the stack (if any) in the case of a violation. Two of the rules specifying the behaviour of a regulated system using hierarchical reparation automata are shown below (we write $q$ to denote $(q_1, q_2)_{q_{3_{\mathscr{A}}}})$):

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{A} q_{3_{\mathscr{A}}}'}{(q_1, q_2)_{(H, q_{3_{\mathscr{A}}}):hs} \overset{A}{\Rightarrow} (q_1', q_2')_{(H, q_{3_{\mathscr{A}}}'):hs}} \neg viol(q, A) \wedge q_{3_{\mathscr{A}}}' \notin F(H) \cup Q_N(H)$$

$$\frac{(q_1, q_2) \xrightarrow{A} (q_1', q_2'),\ q_{3_{\mathscr{A}}} \xrightarrow{A} q_{3_{\mathscr{A}}}'}{(q_1, q_2)_{(H, q_{3_{\mathscr{A}}}):(H', q_{4_{\mathscr{A}}}'):hs} \overset{A}{\Rightarrow} (q_1', q_2')_{conf_0(\text{rep}(H')):hs}} viol(q, A) \wedge q_{4_{\mathscr{A}}}' \in Q_N(H')$$

The example given earlier in this section, regarding the reparation for an airline not to allow a passenger on board, can be modelled using a hierarchical reparation automaton as shown below:


$\mathcal{O}_c$(*boardPlane*)
RepBoardPlane
RepBoardPlane
$\mathcal{O}_c$(*bookAccomodation*)
{*bookAccomodation*}
$\mathcal{O}_c$(*bookNextFlight*)
{*bookNextFlight*}

As in reparation automata, we cannot differentiate between the norms being violated and thus hierarchical reparation automaton are not closed under synchronous composition. Although one could tag reparation sub-automata with an expression specifying upon which violation it is to be trigerred (similar to extended reparation automata), this would still not suffice since the normal behaviour automaton may contain multiple states and thus a single norm may be violated from different states.

## 4 Discussion and Related Work

We have presented three distinct approaches to adding reparations to contract automata: reparation automata provide a rudimentary means of enabling branching upon a violation as part of a contract. In a reasonably sophisticated reparation setting, 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 by allowing branching on conditions identifying which clauses have been violated. Finally, to add structure to our contracts (and reparations), we introduced hierarchical automata to specify reparation contracts as sub-automata.

Of these approaches we found extended reparation automata to deal best with the airline case study [Azz14], although the structure of most parts of the contract are lost in the formalisation. Unless one tags hierarchical reparation automata with clauses which are to be violated to trigger reparation, extended reparation automata are the only formalism that manages to deal with states with multiple clauses effectively. However employing this increases the size of the automata and makes the representation of larger contracts difficult to generate and understand.

This complexity does not come as a surprise, since reparations, although generally well understood on more structured logical approaches e.g. [GR06, FPS09, Hag01] are generally more challenging in graph-based approaches. For space reasons we survey only a couple that represent the general state-of-the-art: *C-O diagrams* [DCMS13] 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 [BDF14] detect violations at runtime, but do not have a formalisation that allows reasoning about them.

It is interesting to note that in all the presented approaches reparations for the violation of permissions are handled analogously to the more traditionally explored notions of contrary-to-duty and contrary-to-prohibition clauses. At a semantic level, violations of permissions in contract automata occur at the state level (when one party does not allow the necessary interaction to perform the permitted action) as opposed to violations of obligations and prohibitions which occur at the transition level (when, in the case of a prohibition, an action is performed or, in the case of an obligation, it is not). This dual nature may indicate that the reparation of the two should be handled in a similarly dual manner — with reparations for violation of permission being related to a state, while the reparation for an obligation or prohibition to be related with a transition. However, such an approach would substantially increase the complexity of the formalism's syntax (before the dual nature of violation was implicit in the semantics of contract automata, not their syntax). The hierarchical approach fares best here, since the moment of violation triggering the reparation is identified semantically and can thus be addressed differently for permission as opposed to obligation and prohibition in a syntactically transparent manner.

It is interesting to note that extended reparation automata allow for branching on the violation of clauses which are not part of the contract clauses in force at a particular state. Consider, for example, a state in which the only active clause is that the passenger is obliged to prove their identity, and from which one may have outgoing transitions depending on whether or not the passenger permits the airline to record his details on their database — if he does, then they are forbidden from sharing the information with others. Note that the requirement to permit the airline company to collect the passenger's data was never in force, and can be thus seen as a hypothetical permission clause. Although compliance to or violation of hypothetical prohibitions and obligations can be emulated using standard contract automata, similar reasoning about permissions was not possible. In our formalism, the possibility of adopting such hypothetical clauses on transitions may lead to spurious violations being identified (ones which are hypothetical but not actual), which thus requires additional machinery to identify violations with clauses in force on the states, but does not permit a straightforward relationship between reparations and clauses in force.

## References

[Azz14] Shaun Azzopardi. Extending contract automata with reparation, hypothetical and conditional clauses. Technical report, University of Malta, May 2014.

[BDF14] Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari. Automata for service contracts. In *Hot Issues in Security Principles and Trust 2014*, 2014.

[DCMS13] 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.

[FPS09] 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.

[GR06] 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.

[Hag01] Jaap Hage. Contrary to Duty Obligations — A Study in Legal Ontology. In *Legal Knowledge and Information Systems (JURIX 2001)*, December 2001.

[MLS97] Erich Mikk, Yassine Lakhnech, and Michael Siegel. Hierarchical automata as model for statecharts. In *Third Asian Computing Science Conference. Advances in Computing Science – ASIAN'97*, volume 1345 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[PS12] 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.