Synthesising Implicit Contracts[*]

Gordon J. Pace
Departament of Computer Science
University of Malta, Malta
gordon.pace@um.edu.mt

Fernando Schapachnik
Departamento de Computación, FCEyN
Universidad de Buenos Aires, Argentina
fschapachnik@dc.uba.ar

ABSTRACT

In regulated interactive systems, one party's behaviour may impose restrictions on how others may behave when interacting with it. These restrictions may be seen as *implicit contracts* which the affected party has to conform to and may thus be considered inappropriate or excessive if they overregulate one of the parties. In this paper we characterise such implicit contracts and present an algorithmic way of synthesising them using a formalism based on contract automata to regulate interactive action-based systems.

Categories and Subject Descriptors

F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Theory, Languages

Keywords

Automated Legislative Drafting,Contract Synthesis

1. INTRODUCTION

Consider the contract that binds a customer and a bank, which stipulates that opening new accounts is free of charge. And yet, at the moment of opening an account, the bank requires the release of personal information and allowance to send the customer promotional material. The bank is not strictly breaching the contract, but maybe it is asking "too much". Can this "too much" be quantified? As another example consider an airline, which compensates for missed connections due to delays by providing the traveller with food and lodging. However, the airline has a policy of not providing this service unless the customers explicitly demands for it. In a way, the airline is turning its unconditional obligation of providing aid into a conditional or restricted one: given that the customer asks for help, support will be provided.

[*]Partially supported by UBACyT 20020100200103.

In interactive systems involving different parties, the behaviour of each party is inherently affected by the behaviour of the others. In particular, other parties may restrict or enforce behaviour resulting in the affected party having to behave in a different manner than if it were to act independently. In interactive systems regulated by contracts, each party thus may be seen to be restricted through the explicitly agreed upon contracts, but furthermore by the implicit constraints induced through the nature of interaction and the other parties' behaviour. These implicit constraints can be seen as 'invisible' (or unspoken) agreements or contracts a party has to agree to and adhere to if they choose to participate in the interaction.

As the imposed behaviour gets stricter, it drifts from being compliant up to the point of being close to breaching the contract. Can this drifting be measured? As a first step, in this paper we develop techniques so that, given two interacting parties bounded by a contract, we can infer the implicit contract being enforced by one party on the other.

This approach is useful also during contract negotiation: given multiple service providers, a party may base her choice of provider not only upon the signed agreement, but also taking into consideration the implicit contract imposed on her.

To present these notions, we use *contract automata* (Section 2), a formalism based on the notion of synchronising automata from computer science, that provides well understood and clear semantics to model interaction among parties. An automata-based formalism allows to model not only deontic formulae but also the choices that each party has and how decisions affect the other parties.

Implicitly enforced behaviour of a party is presented in Section 3, together with an algorithm to compute it. The next section explains the extra challenges that arise when considering non-deterministic systems, and why, consequently, the algorithm only copes with deterministic ones. Finally, Section 5 surveys related work before concluding the article in Section 6.

2. INTERACTING TWO-PARTY SYSTEMS

In [6] we presented a model for contracts between interacting parties, based on the notion of synchronous composition [1] and multi-action labels on transitions.[1] This section summarises the key aspects of our model.

DEFINITION 2.1. *A multi-action automaton S is a 4-tuple with components $\langle \Sigma,\ Q,\ q0,\ \rightarrow \rangle$, where Σ is the alphabet of actions, Q is the set of states, $q0 \in Q$ 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 $(q, A, q') \in \rightarrow$,*

[1]The use of multi-actions allows the modelling of contracts such as ones in which both parties have different obligations at the same time.

$\mathrm{acts}(q)$ *to be the set of all action sets on the outgoing transitions from q (defined to be* $\{A \mid \exists q' \cdot q \xrightarrow{A} q'\}$*).We will write* $\overset{w}{\Rightarrow}$ *for the transitive closure of* $\rightarrow$ *(with* $w \in (2^{\Sigma})^*$*), and for deterministic automata, we will write* $q \overset{w}{\Rightarrow}$ *to indicate the unique state* q' *such that* $q \overset{w}{\Rightarrow} q'$.

The synchronous composition of two automata S_1 *and* S_2 *(with* $S_i = \langle Q_i,\ q0_i,\ \rightarrow_i \rangle$*), both with alphabet* Σ *and synchronising over alphabet G, written* $S_1 \|_G S_2$*, is defined to be* $\langle Q_1 \times Q_2,\ (q0_1, q0_2), \rightarrow \rangle$*, where* $\rightarrow$ *is the classical synchronous composition relation defined in [1]. Actions in G are called* shared *while actions in* G^c *are called* local*. We will assume that all systems are well-formed, i.e., do not deadlock.*

We can now define contracts to be automata with each state tagged with the clauses which will be in force at that point. The contracts will be able to refer to both presence and absence of actions.

Contract clauses are either (i) obligation clauses of the form $\mathcal{O}_p(a)$ or $\mathcal{O}_p(!a)$, which say that party p is obliged to perform or not perform action a respectively; or (ii) permission clauses which can be either of the form of $\mathcal{P}_p(a)$ or $\mathcal{P}_p(!a)$ (party p is permitted to perform, or not perform action a respectively). Given an alphabet of actions Σ, we write $\Sigma!$ to refer to the alphabet extended with actions preceded with an exclamation mark ! to denote their absence: $\Sigma! \overset{df}{=} \Sigma \cup \{!a \mid a \in \Sigma\}$. We will use variables x, y and z to stand for either presence or absence of an action e.g. $\mathcal{P}_p(x)$ would match both $\mathcal{P}_p(a)$ and $\mathcal{P}_p(!a)$.

DEFINITION 2.2. *A* contract automaton *is a total and deterministic multi-action automaton* $S = \langle Q,\ q0,\ \rightarrow \rangle$*, together with a total function* $\mathrm{contract} \in Q \rightarrow 2^{\mathit{Clause}}$ *assigning a set of clauses to each state.*
A contract automaton $\mathcal{A}$ *is said to be p-specific if all the clauses refer to party p i.e.* $\mathcal{O}_p(x)$ *or* $\mathcal{P}_p(x)$*.*
Two contract automata $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are said to be* deontically equivalent*, written* $\mathcal{A}_1 \equiv \mathcal{A}_2$*, if for all traces* $w \in (2^{\mathit{Clause}})^*$*, the clauses of the reached states are equal:* $\mathrm{contract}_1(q0_1 \overset{w}{\Rightarrow}) = \mathrm{contract}_2(q0_2 \overset{w}{\Rightarrow})$.

2.1 Contract Satisfaction

Given a two-party system (S_1, S_2), and a contract automaton $\mathcal{A}$, we can now define whether or not either party is violating the contract when a particular state is reached or a transition is taken.

DEFINITION 2.3. *Functions* $O_p(q_{\mathcal{A}})$ *and* $F_p(q_{\mathcal{A}})$ *give the set of actions respectively obliged to be performed and obliged not to be performed by party p. They are defined in terms of the contract clauses in the state. Action set A is said to be viable for party p in a contract automaton state* $q_{\mathcal{A}}$*, written* $\mathrm{viable}_p(q_{\mathcal{A}}, A)$*, if (i) all her obliged actions are included in A but; (ii) no actions which the party is obliged not to perform are included in A:*
$\mathrm{viable}_p(q_{\mathcal{A}}, A) \overset{df}{=} O_p(q_{\mathcal{A}}) \subseteq A \wedge F_p(q_{\mathcal{A}}) \cap A = \emptyset$

Since we would like to be able to place blame in the case of a violation, we parametrise contract satisfaction and violation by party. It is also worth noting that while obligation to perform an action, for instance, is violated in a transition which does not include the action, permission is violated by a state in which the opportunity to perform the permitted action is not present. The satisfaction predicate will thus be overloaded to be applicable to both states and transitions. The predicate $\mathit{sat}_p^{\mathcal{A}}(X)$ will denote that the contract automaton $\mathcal{A}$, reaching state X or traversing transition X, does not constitute a violation for party p.

Permission.
If party p is permitted[2] to perform shared action a, then the other party $\overline{p}$ must provide p with at least one viable outgoing transition which contains a but does not include any forbidden actions. Permission to perform local actions cannot be violated.

In the case of a single permission, this can be expressed as follows:

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

Similarly, if party p is permitted to not perform action a, then the other party $\overline{p}$ must provide p with at least one viable outgoing transition which does not include a nor any forbidden action. Permission to perform local actions can never be violated. In the case of a single permission, this can be expressed as follows:

$$\begin{aligned}
&(q_1, q_2)_{q_{\mathcal{A}}} \vdash_{\overline{p}} \mathcal{P}_p(!a) \overset{df}{=} \\
&\quad a \in G \implies \exists A \in \mathit{acts}(q_{\overline{p}}),\ \exists A' \subseteq G^c \cdot \\
&\qquad a \notin A \ \wedge\ \mathit{viable}_p(q_{\mathcal{A}}, A \cup A')
\end{aligned}$$

While actual obligation violations occur when an action is not performed, violations of a permission occur when no appropriate action is possible.
To combine all permissions in a state, we simply take the conjunction of all conditions:

$$\mathit{sat}_p^P((q_1, q_2)_{q_{\mathcal{A}}}) \overset{df}{=} \forall \mathcal{P}_p(x) \in q_{\mathcal{A}} \cdot (q_1, q_2)_{q_{\mathcal{A}}} \vdash_{\overline{p}} \mathcal{P}_p(x)$$

We say that $((q_1, q_2)_{q_{\mathcal{A}}}, A) \vdash_{\overline{p}} \mathcal{P}_p(a)$ if $A \in \mathit{acts}(q_{\overline{p}})$ is the action set that exists to satisfy $(q_1, q_2)_{q_{\mathcal{A}}} \vdash_{\overline{p}} \mathcal{P}_p(a)$. We use similar definitions for $((q_1, q_2)_{q_{\mathcal{A}}}, A) \vdash_p \mathcal{P}_p(a)$ and absence of actions.

Obligation.
Obligations bring in constraints on both parties. Given that party p is obliged to perform action a in a state means that (i) party p must include the action in any outgoing transition in the composed system in which it participates; and (ii) the other party $\overline{p}$ must provide a viable synchronisation action set which, together with other asynchronous actions performed by p, allows p to perform *all* its obligations, positive and negative. Obligation to not perform action a ($\mathcal{O}_p(!a)$) can be similarly expressed. We combine all positive and negative obligations in the following definition:

$$\begin{aligned}
&\mathit{sat}_p^O((q_1, q_2)_{q_{\mathcal{A}}} \xrightarrow{A} (q_1', q_2')_{q'_{\mathcal{A}}}) \overset{df}{=} \mathit{viable}_p(q_{\mathcal{A}}, A) \\
&\mathit{sat}_{\overline{p}}^O((q_1, q_2)_{q_{\mathcal{A}}}) \overset{df}{=} \\
&\qquad \exists A \in \mathit{acts}(q_{\overline{p}}),\ \exists A' \subseteq G^c \cdot \mathit{viable}_p(q_{\mathcal{A}}, A \cup A')
\end{aligned}$$

We can now define the rest of the deontic modalities:

- Party p not being permitted to perform an action is equivalent to p being obliged not to perform the action:
$$!\mathcal{P}_p(a) \overset{df}{=} \mathcal{O}_p(!a) \qquad !\mathcal{P}_p(!a) \overset{df}{=} \mathcal{O}_p(a)$$
- Party p not being obliged to perform an action is equivalent to p being permitted not to perform the action:
$$!\mathcal{O}_p(a) \overset{df}{=} \mathcal{P}_p(!a) \qquad !\mathcal{O}_p(!a) \overset{df}{=} \mathcal{P}_p(a)$$

[2]Some people are more comfortable with the term *right* instead of *permission* when it imposes some type of behaviour on the other party. However, as with all terms in deontic literature, many slight and not-so-slight variations of its meaning are used by different authors. For instance, do they last in time? Is the other party responsibility unavoidable or only subject to "best-effort"? While in a philosophical essay it might make sense to dispute over the appropriate name, we use formal semantics to dissolve all doubts.

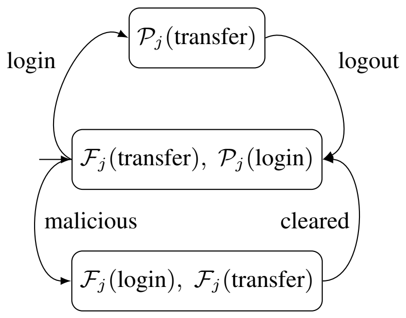
$\mathcal{P}_j$(transfer)
login
logout
$\mathcal{F}_j$(transfer), $\mathcal{P}_j$(login)
malicious
cleared
$\mathcal{F}_j$(login), $\mathcal{F}_j$(transfer)

Figure 1: Internet banking contracts

- Prohibition contract clauses $\mathcal{F}_p(a)$ and $\mathcal{F}_p(!a)$, prohibiting party p from performing and not performing a respectively, can be expressed in terms of permission:
$$\mathcal{F}_p(a) \overset{df}{=} !\mathcal{P}_p(a) \qquad \mathcal{F}_p(!a) \overset{df}{=} !\mathcal{P}_p(!a)$$
- Prohibition to perform an action is equivalent to obligation not to perform the action: $\mathcal{F}_p(x) = \mathcal{O}_p(!x)$.

General contract satisfaction.

It is defined as: $sat_p(X) \overset{df}{=} sat_p^P(X) \wedge sat_p^O(X)$.

Example 1: If the contract of Jane (p) with her bank permits her to withdraw money, permits her not to deposit money, obliges her to pay a fee, and obliges her not to steal ($\mathcal{P}_p(w)$, $\mathcal{P}_p(!d)$, $\mathcal{O}_p(f)$, $\mathcal{O}_p(!s)$), the bank ($\overline{p}$) should provide at least one transition that contains both a w and an f action and contains neither a d nor an s.

Example 2: Now consider John signing a contract with his bank. The contract says that (i) whenever John is logged into his Internet banking account he is to be permitted to make money transfers; and (ii) if a malicious attempt to log in to his account is identified, logging in and making transfers will be prohibited until the situation is cleared. The two statements can be expressed in the contract automaton shown in Figure 1.

2.2 Contract Strength

DEFINITION 2.4. *A party p is said to be incapable of breaching a contract in a regulated two-party system $R = \langle S_1, S_2\rangle_G^{\mathcal{A}}$, written* breachIncapable$_p(R)$*, if p cannot be in violation in any of the reachable states and transitions of R. We write* breachIncapable(R) *if it holds for both parties.*

A contract automaton $\mathcal{A}'$ is said to be stricter than *contract automaton $\mathcal{A}$ for party p (or $\mathcal{A}$ said to be* more lenient *than $\mathcal{A}'$ for party p), written $\mathcal{A} \sqsubseteq_p \mathcal{A}'$, if for any systems S_1 and S_2, p being incapable of breaching $\mathcal{A}'$ implies that p is incapable of breaching $\mathcal{A}$. We say that two contract automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent for party p, written $\mathcal{A} =_p \mathcal{A}'$, if $\mathcal{A} \sqsubseteq_p \mathcal{A}'$ and $\mathcal{A}' \sqsubseteq_p \mathcal{A}$. Similarly, we define strictly more lenient relation $\sqsubset_p$ as $\sqsubseteq_p \setminus =_p$. We define global contract strictness $\mathcal{A} \sqsubseteq \mathcal{A}'$ to hold if $\mathcal{A} \sqsubseteq_p \mathcal{A}'$ holds for all parties p, and similarly global contract equivalence $\mathcal{A} = \mathcal{A}'$.*

DEFINITION 2.5. *Given two contract clauses C and C', the relation over contract automata $[C \to C'] \subseteq \mathcal{CA} \times \mathcal{CA}$ relates two contract automata $\mathcal{A}$ and $\mathcal{A}'$ if $\mathcal{A}$ is equivalent to $\mathcal{A}'$ except possibly for a number of instances of clause C replaced by C'.*

We extend the notion of strictness to contract clauses. We say that clause C' is stricter than clause C for party p, written $C \sqsubseteq_p C'$, if for any contract automata $\mathcal{A}$ and $\mathcal{A}'$ such that $(\mathcal{A}, \mathcal{A}') \in [C \to C']$, it follows that $\mathcal{A} \sqsubseteq_p \mathcal{A}'$. We similarly extend the notion of strictness for all parties $\sqsubseteq$.

2.3 Restricted Permission

In this article we introduce restricted permissions in a rather abstract way. The intended meaning is to permit some action provided some other action takes place in the same action set. In real cases one would like various different types of predicates in order to express the condition that should hold for the action to be permitted. Typical cases include propositional predicates over variables that hold in states (e.g., "John is permitted to withdraw money (wm) if there is a positive balance (pb)"), over sequences of precedent actions (e.g., "Mary is permitted to use the service (us) provided she had acquired a ticket (at) and presented it in the front office (pfo) in the same day"), etc.

It is important to note that we do not aim for a formalism that people would use directly to express their contracts, but rather for a formalism where those contracts could be compiled into, much like Kripke structures, or the program control structures used in program analysis (e.g. see [8]).

To simplify presentation of the examples, we use multi-actions for actions which take place in a narrow time interval. For example, action set $\{pb, wm\}$ encodes John's withdrawal when he has a positive balance, action set $\{at, pfo, us\}$ can represent Mary's rightful usage of the service, while $\{at, us\}$, $\{pfo, us\}$ and $\{wm\}$ are examples of violations.

Restricting Permissions.

We want to capture the notion of party p being permitted do to action a only if action set $C \subseteq 2^{\Sigma!}$ is present as well. We will write it $\mathcal{P}_p(C \rhd a)$ and call C the condition. More precisely $\mathcal{P}_p(C \rhd a)$ holds iff there is no way of doing a without doing C as well, and there is also a transition which makes it possible to do $\mathcal{P}_p(a)$ and also contains C.

Note that the C set can contain either local or synchronised actions. That allows to express requirements to the same individual (e.g., "you need to raise your hand before asking questions"), or even more complex interactions requiring the other party involvement (e.g., "you need to raise your hand and be acknowledged before asking questions").

$$\begin{aligned}
&\textit{requires}_p(q, C, a) \overset{df}{=} \\
&\quad \forall A \in \textit{acts}(q),\ a \in A \ \cdot\ ((C \cap (\Sigma_p \cup G)) \subseteq A) \\
&(q_1, q_2)_{q_{\mathcal{A}}} \vdash_p \mathcal{P}_p(C \rhd a) \overset{df}{=} \\
&\quad \textit{requires}_p(q_p, C, a) \\
&(q_1, q_2)_{q_{\mathcal{A}}} \vdash_{\overline{p}} \mathcal{P}_p(C \rhd a) \overset{df}{=} \\
&\quad \textit{requires}_{\overline{p}}(q_{\overline{p}}, C, a) \wedge \\
&\quad \exists A \in \textit{acts}(q_{\overline{p}}),\ \exists A' \subseteq G^c \cdot \\
&\qquad \textit{viable}_p(q_{\mathcal{A}}, A \cup A') \wedge C \subseteq (A \cup A') \wedge \\
&\qquad ((q_1, q_2)_{q_{\mathcal{A}}}, A) \vdash_{\overline{p}} \mathcal{P}_p(a)
\end{aligned}$$

Party p complies with $\mathcal{P}_p(C \rhd a)$ if all of its outgoing transitions that contain an a also contain the actions in C that are either local or shared. On the other hand, party $\overline{p}$ needs to satisfy the same, plus allowing at least one viable transition containing the whole C and at the same time permitting a.

Note that an alternative definition could also be given — one where there is no need for a transition to effectively exist. Also note that the given definition trivialises if C is the empty set, because its role as requirement becomes void.

The definitions just given can be generalised so that party p, in order to do action a, is restricted to comply with not one give set of conditions, but one from a set of sets as in $\mathcal{P}_p(C_1 \vee \ldots \vee C_n \rhd a)$.

3. CONTRACT SYNTHESIS

Through their very nature, interacting systems affect the behaviour of each other through imposed synchronisation, with a party's behaviour sometimes forcing or disabling certain behaviour from the other party.

3.1 Implicitly Enforced Behaviour

Since synchronised actions cannot take place without the participation of both parties, and both parties must ensure progress, the behaviour of one party can force the other to behave in a particular way. In this manner, a party may force the co-party to behave as though a particular contract was being enforced. For instance, if airport management staff only allow a synchronised *boardAircraft* action only after a passenger performs a *bodySearch* action, from the point of view of the passenger it is as though he or she is prohibited from boarding the aircraft unless he or she allows a body search to take place.

DEFINITION 3.1. *Given a party p with behaviour S and synchronising on alphabet G, a $\overline{p}$-specific and conflict-free contract $\mathcal{A}$ is said to be* implicitly enforced *on $\overline{p}$ if due to the composition with S, $\mathcal{A}$ will not be violated no matter what the behaviour of $\overline{p}$ is:*

$$\forall S' \cdot \text{breachIncapable}(\langle S, S' \rangle_G^{\mathcal{A}})$$

We say that an implicitly enforced contract $\mathcal{A}$ (with behaviour S of party p) is said to be maximal if there is no other implicitly enforced contract $\mathcal{A}'$ such that $\mathcal{A} \sqsubset_{\overline{p}} \mathcal{A}'$.

Although $\sqsubseteq_{\overline{p}}$ is not a total order, implicitly enforced contracts have a unique maximum up to $=_{\overline{p}}$.

THEOREM 3.1. *Any two maximal implicitly enforced contracts of party p with behaviour S are equivalent up to $=_{\overline{p}}$.*

Computing one of these maximal elements for a deterministic system can be done using the algorithm of Definition 3.3. Non-deterministic system impose non-trivial challenges that we discuss in Section 4. Before presenting the algorithm we need the notion of Implicit Conditional Permission.

Implicit Restricted Permission.

Think of a transition labelled $\{a, b, c\}$. In order to do a, the other party $\overline{p}$ needs to do b and c as well. If there is another transition labelled $\{a, d, e\}$, then we can conclude that $\mathcal{P}_{\overline{p}}(\{d, e\} \vee \{b, c\} \rhd a)$ and also $\mathcal{P}_{\overline{p}}(\{a, c\} \rhd b)$, $\mathcal{P}_{\overline{p}}(\{a, b\} \rhd c)$, etc.

Generalising, the idea is that each shared action appearing in an outgoing transition can only be exercised by the other party if she also performs all of the other shared actions that come with the transition.

Now consider that another transition is added, labelled only with a. The algorithm proposed so far would synthesise $\mathcal{P}_{\overline{p}}(\{d, e\} \vee \{b, c\} \vee \emptyset \rhd a)$. However, a restricted permission with an empty condition removes the imposition of doing other actions in order to do a.

Considering all the above gives raise to the algorithm of Definition 3.2.

DEFINITION 3.2. *The algorithm to synthesise the restricted permissions imposed by p to any other party $\overline{p}$ in a given state q_p is as follows.*

1. *Construct the alphabet used in its outgoing transitions (the 'local' alphabet): $\Sigma_{q_p} = \bigcup_{A \in \text{acts}(q_p)} A$.*
2. *For each action $a \in (\Sigma_{q_p} \cap G)$*
   - (a) *Let C be an empty set of sets of actions.*
   - (b) *For each $A \in \text{acts}(q_p)$*
     - i. *if $a \in A$, then add $(A \cap G) - \{a\}$ to C*
   - (c) *Let $C_1 \ldots C_n$ be the members of C.*
   - (d) *If there is not $1 \leq i \leq n$ such that $C_i = \emptyset$, then add $\mathcal{P}_{\overline{p}}(C_1 \vee \ldots \vee C_n \rhd a)$*

The algorithm works as follows: for every shared action a that appears in an outgoing transition from q_p, visit every outgoing transition (step 2b). If the action is part of that transition's action set then we have probably found a new condition for doing a: the rest of the shared actions in that action set (step 2b.i). If a can be done without further restrictions in this transition, then the just mentioned set becomes empty and there is not restriction for doing a. That is checked in step 2d.

Synthesis Algorithm.

The idea behind the algorithm is as follows: Suppose a is a shared action. If a given state of S does not have a-transitions, then any system synchronising with S will be forbidden to perform an a, thus $\mathcal{F}_{\overline{p}}(a)$ should be synthesised. On the other hand, if all of its outgoing transitions have an a, then a is mandatory for the other party, etc.

DEFINITION 3.3. *An implicitly enforced contract of a deterministic system $S = \langle \Sigma, Q, q0, \rightarrow \rangle$ can be computed to be the contract automaton* enforced(S) *in the following manner:*

1. *We start with an automaton (states, initial state and transition relation) identical to S.*
2. *We add a new state Δ to the automaton. Furthermore, for any state q and action set A such that $A \notin \text{acts}(q)$, we add the transition $q \xrightarrow{A} \Delta$. For any action set A, $\Delta \xrightarrow{A} \Delta$. This step is needed to guarantee that the resulting automaton is total, which is required by Definition 2.2.*
3. *For any state q (with $q \neq \Delta$),* contract(q) *contains $\mathcal{F}_{\overline{p}}(a)$ if and only if in the original automaton $\forall A \in \text{acts}(q) \cdot a \notin A$.*
4. *For any state q (with $q \neq \Delta$),* contract(q) *contains $\mathcal{O}_{\overline{p}}(a)$ if and only if in the original automaton $\forall A \in \text{acts}(q) \cdot a \in A$.*
5. *For any state q (with $q \neq \Delta$),* contract(q) *contains $\mathcal{P}_{\overline{p}}(a)$ if and only if in the original automaton $\exists A \in \text{acts}(q) \cdot a \in A$.*
6. *For any state q (with $q \neq \Delta$),* contract(q) *contains the restricted permissions given by Definition 3.2.*

THEOREM 3.2. *The above algorithm synthesises a maximal implicitly enforced contract of party p.*

3.2 Comparing Contracts

In a scenario where party p is negotiating a contract with party $\overline{p}$, or has already engaged into one, p could compare the expected behaviour and the one that $\overline{p}$ would actually bring about as computed in Definition 3.3. Such comparison could highlight plain incompatibilities but also "deviations", such as a permission turned into a restricted one. During contract negotiation, this comparison could be used to select the more convenient option among various bidders.

Getting back to the case of the airline obliged to provide assistance to passengers, suppose airline A satisfies every customer's demand, while airline B also does so but only if the customer calls

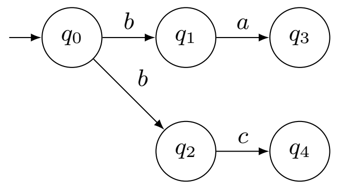
$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, "b"] \arrow[dr, "b"] & q_1 \arrow[r, "a"] & q_3 \\ & & q_2 \arrow[r, "c"] & q_4 \end{tikzcd}$$

Figure 2: Example of non-deterministic system

B's headquarters, thus adding an extra condition on the customer in order to exercise its permission to request assistance. Despite this difference, it can be argued that both airlines satisfy the regulation, yet with varying degrees.

4. DIFFICULTIES FOR NON-DETERMINISTIC SYNTHESIS

Synthesising contracts for non-deterministic systems posses some difficulties. Consider a system S_1 with shared alphabet $G = \{a, b, c\}$, like the one in Figure 2.

Looking at state q_1 we can synthesise $\{\mathcal{O}_2(a), \mathcal{F}_2(c)\}$. State q_2 gives the opposite: $\{\mathcal{O}_2(c), \mathcal{F}_2(a)\}$ (both also give $\mathcal{F}_2(b)$ but it is immaterial for the example).

That means that after performing b from q_0 party S_2 is either obliged to do action a or action c. However, contract automata are deterministic in our model. One option would be to apply the algorithm of Definition 3.3 to obtain a pseudo-contract automaton $\mathcal{A}$ and then mix the resulting states $q_1^{\mathcal{A}}$ and $q_2^{\mathcal{A}}$ into $q_{1,2}^{\mathcal{A}}$, but that gives the conflicting set $\{\mathcal{O}_2(a), \mathcal{F}_2(c), \mathcal{O}_2(c), \mathcal{F}_2(a)\}$.

Another option would be to determinise S_1 by merging q_1 and q_2 into $q_{1,2}$ and then computing the enforced contract. That would synthesise $\{\mathcal{P}_2(a), \mathcal{P}_2(c)\}$ for the corresponding state. That would also be incorrect: it could be the case that a given S_2 synchronises with S_1, and after doing action b ends up in q_1 being forbidden to do action c.

As can be seen from the given example, synthesising contracts for non-deterministic systems remains an open issue that should be further researched.

5. RELATED WORK

Although contracts are a long-covered topic in deontic literature (e.g., [4, 5, 6]), we are not aware of work on synthesising implicit contracts. Process calculi (e.g., [2]) or even petri nets (e.g., [9]) have been used before to model "contracts". However, the term contract is used there in the sense of *interface*, as way to guarantee services interoperability, but they are not rich enough for other types of contracts because they do not support deontic operators.

The automata community is well aware of many different techniques for synthesising different types of automata (e.g., [7]), but our setting is very different that theirs. There usually is one automaton P called *plant* and a property φ it is expected to comply with. The problem is to synthesise another automaton C called *controller* such that $P||C \models \varphi$, i.e., such that the controller limits the plant so it complies with φ. In our setting there are two important differences: on one hand, there is no "$\models$" operator but degrees of compliance to a contract. On the other hand, both parties' systems are given, and also is the contract. As there are degrees of compliance it is interesting to see how "good" a co-party is. Take for instance the case mentioned earlier of an airline being obliged by international regulation to provide food and lodging to customers whose flight get delayed more than a certain amount hours and demand assistance. Airline A satisfies every customer's demand. Airline B would only do so if customers call B's headquarters, adding an extra condition on the customer in order to exercise its permission to request assistance. Despite this difference, it could be argued that both airlines satisfy the regulation.

Some authors discuss "implicit deontic effects" (e.g., [3]), but the concept treated is entirely different, as an example clarifies: "If agent i orders a product from agent j then he *implicitly* authorises j to demand payment upon delivery of the goods".

6. CONCLUSIONS

We have presented the notion of implicitly enforced behaviour of a party in a two-party contract setting, together with an algorithm to synthesise it. Although the concept is interesting in itself, it becomes more interesting when one starts to compare this implicitly enforced behaviour to what a given contract demands, as a tool to highlight not only potential breaches but also subtle differences.

Such a comparison permits the view of a continuum going from plain incompatibility to full compliance, with a lattice of possibilities in between — for instance, when one party restricts a permission of the other party by only allowing the permitted action after certain extra actions are performed.

To further explore implicitly enforced behaviour, synthesis for non-deterministic systems, as discussed in Section 4, is still an open problem. Future research directions also includes the dual of implicitly enforced behaviours, *intrinsic well-behaviour:* given the behaviour of a system, induce a contract which it obeys under all circumstances. Such a notion could, for instance, be useful to use to figure out how much more one can ask from the other party without them having to change their behaviour.

7. REFERENCES

[1] A. Arnold. Nivat's processes and their synchronization. *Theor. Comput. Sci.*, 281:31–36, June 2002.

[2] M. Bravetti and G. Zavattaro. Contract based multi-party service composition. In *International Symposium on Fundamentals of Software Engineering*, pages 207–222. Springer, 2007.

[3] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[4] G. Governatori and Z. Milosevic. Dealing with contract violations: formalism and domain specific language. In *EDOC Enterprise Computing Conference, 2005 Ninth IEEE International*, pages 46–57. IEEE, 2005.

[5] O. Marjanovic and Z. Milosevic. Towards formal modeling of e-contracts. In *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*, EDOC '01, pages 59–, Washington, DC, USA, 2001. IEEE Computer Society.

[6] G. Pace and F. Schapachnik. Contracts for interacting two-party systems. In *FLACOS 2012: Sixth Workshop on Formal Languages and Analysis of Contract-Oriented Software*, sep 2012.

[7] A. Pnueli, E. Asarin, O. Maler, and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control. Elsevier*. Citeseer, 1998.

[8] A. Salcianu and M. Rinard. Pointer and escape analysis for multithreaded programs. In *ACM SIGPLAN Notices*, volume 36, pages 12–23. ACM, 2001.

[9] W. M. Van Der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From public views to private views–correctness-by-design for services. In *Web Services and Formal Methods*, pages 139–153. Springer, 2008.