Contract Automata
Towards an Algebra of Contracts

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

March 2016

What's so Interesting about Contracts?

- Normative notions — ideal vs actual behaviour.
- Enabling representation of concepts such as obligations, prohibitions permissions, and exceptional behaviour.
- Ought-to-do or ought-to-be?
- Plagued by paradoxes unless one is very careful and restricts the language.

Paradox of Blame

Paradox of Blame

The law says: *You are obliged to hand in Form A on Monday and Form B on Tuesday, unless officials stop you from doing so.*

But it's sunny outside: On Monday, John spent a day on the beach, thus not handing in Form A. On Tuesday at 00:00 he was arrested, and taken to court on Wednesday.

Prosecuting lawyer: *To satisfy his obligation the defendant had to hand in Form A on Monday, which he did not. Hence he should be found guilty.*

Defendant's lawyer: *But to satisfy the obligation the defendant had to hand in Form B on Tuesday, which he was stopped from doing by officials. He is hence innocent.*

Contracts vs. Specifications

- What are contracts, and are they different from properties or specifications?
  - Obligation to perform x related to the property *'action taken must include x'*
  - Prohibition from performing x related to the property *'action taken may not include x'*
  - But what are permission to perform x? Actions may include x? What are violations?
  - And what about liability?
  - Or exceptional (non-ideal) behaviour?
- Contracts are by definition an agreement between two or more *interacting* parties but most formal studies regulate a party at a time.

Contract clause #1

"John is permitted to withdraw cash."

- John may choose to perform an action *'withdraw cash'*,
- which the bank is bound to engage with.
- John may also choose not to perform the action.
- but if he does and the bank does not allow *the bank has violated the contract*.

Contract clause #2

"John is obliged to pay an annual fee."

- John should perform an action *'pay annual fee'*,
- If John chooses not to perform the action, he has violated the contract.
- But the bank is bound to engage with John's action to allow him to satisfy the contract.

Interacting Systems

- Interaction has a long history in computer science providing tools such as *communication* and *synchronisation* which allow the modelling of directed modalities in a two-party contract setting.
- We formalise two-party contracts modelling interaction using *synchronous composition with multiset-actions*.

Synchronous Composition over Multisets of Actions

- The synchronous composition of two automata $S_i = \langle Q_i,\ q0_i,\ \rightarrow_i \rangle$ synchronising over alphabet G, is written $S_1 \|_G S_2$, is defined to be $\langle Q_1 \times Q_2,\ (q0_1, q0_2), \rightarrow \rangle$.

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

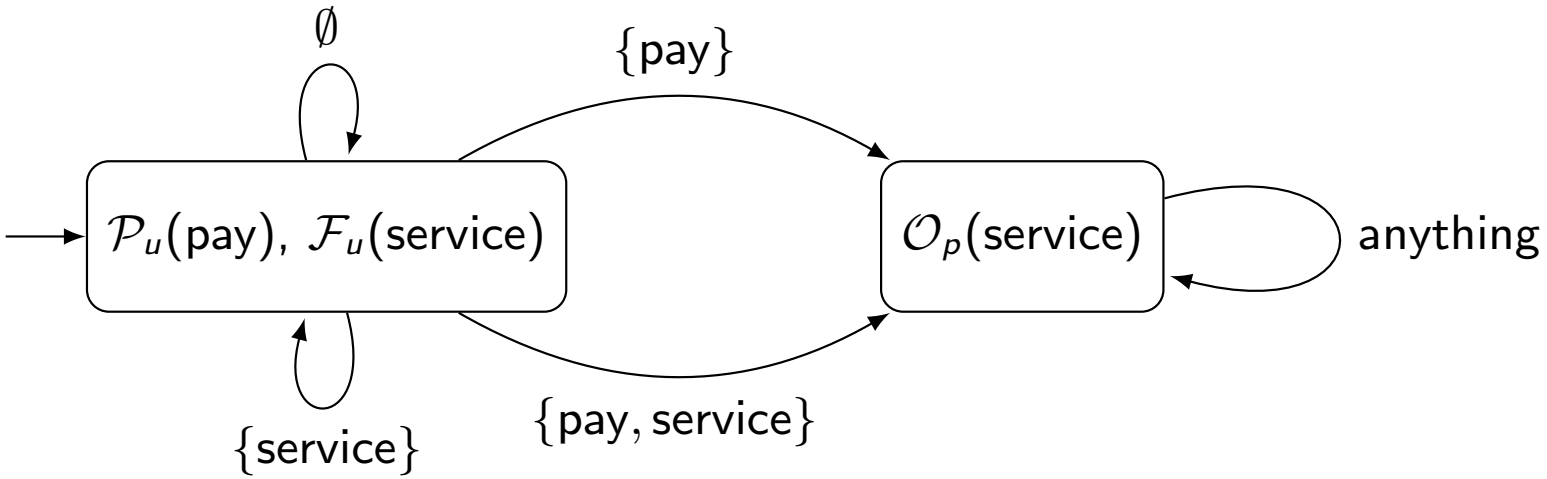
$$\frac{q_1 \xrightarrow{A}_1 q_1' \qquad q_2 \xrightarrow{B}_2 q_2'}{(q_1, q_2) \xrightarrow{A \cup B} (q_1', q_2')}\ A \cap G = B \cap G \neq \emptyset$$

But What About the Contract?

- Contracts are also encoded as automata with states tagged with the contract clauses that will be in force at that point.

Contract

"Initially, the user (party u) is forbidden from using the *service* but permitted to *pay* after which the provider (party p) is obliged to provide the *service*."



Contract Automata

- A *contract clause* is one of the following:

$$\text{Clause} \quad ::= \quad \mathcal{O}_p(a) \mid \mathcal{O}_p(!a) \mid \mathcal{P}_p(a) \mid \mathcal{P}_p(!a)$$

- A contract automaton is a normal automaton with an additional function $Q \to 2^{\text{Clause}}$.
- The transition relation of contract automata is always total to ensure no deadlock, *even after a violation occurs*.

Contract Automata

- A *contract clause* is one of the following:

$$\text{Clause} \quad ::= \quad \mathcal{O}_p(a) \mid \mathcal{O}_p(!a) \mid \mathcal{P}_p(a) \mid \mathcal{P}_p(!a)$$

- A contract automaton is a normal automaton with an additional function $Q \to 2^{\text{Clause}}$.
- The transition relation of contract automata is always total to ensure no deadlock, *even after a violation occurs*.

Prohibition

Prohibition $\mathcal{F}_p(a)$ is just $\mathcal{O}_p(!a)$.

Contract Automata

- A *contract clause* is one of the following:

$$\text{Clause} \quad ::= \quad \mathcal{O}_p(a) \mid \mathcal{O}_p(!a) \mid \mathcal{P}_p(a) \mid \mathcal{P}_p(!a)$$

- A contract automaton is a normal automaton with an additional function $Q \to 2^{\text{Clause}}$.
- The transition relation of contract automata is always total to ensure no deadlock, *even after a violation occurs*.

Negative clauses

$$!\mathcal{O}_p(x) = \mathcal{P}_p(!x)$$
$$!\mathcal{P}_p(a) = \mathcal{O}_p(!a)$$

Regulated Two-Party Systems

- A *regulated two-party system* synchronising over the set of actions G consists of three parts: (i) the interacting systems S_1 and S_2 and (ii) the contract $\mathcal{A}$.
- By composing the contract automaton $\mathcal{A}$ with the parties' behaviour we can then identify what clauses are in force and when, hence allowing analysis for contract violation: $(S_1 \|_G S_2) \|_\Sigma \mathcal{A}$.

Contract Violation

- Given a regulated two-party system, we can now analyse the system automata with respect to the contract clauses and tag violations and the responsible party.
- Violations can occur on:
  - **Transitions:** e.g. a transition which contains an action which is prohibited at that point.
  - **States:** e.g. a state in which a party does not permit (allow) the other party to perform an action which should be permitted.

Obligations

- Obligation $\mathcal{O}_p(a)$ brings in constraints on both parties:
  1. party p must include the action in any outgoing transition in the composed system; and
  2. the other party $\overline{p}$ must provide a viable action set which allows p to perform *all* its obligation.
- The former marks transitions as violations, whereas the latter marks states.

$$viable_p(q_{\mathcal{A}}, A) \stackrel{df}{=} O_p(q_{\mathcal{A}}) \subseteq A \wedge F_p(q_{\mathcal{A}}) \cap A = \emptyset$$

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

Permissions

- If party p is permitted 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.
- Violations of a permission occur when no appropriate action is possible, and is thus a property of a state not a transition.
- We use a semantics that tags as a violation a state in which one party is permitted to perform an action, while the other provides no way of actually doing so.

$$(q_1, q_2)_{q_\mathcal{A}} \vdash_p \mathcal{P}_p(a) \stackrel{df}{=} \mathit{true}$$

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

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

Breach-incapability

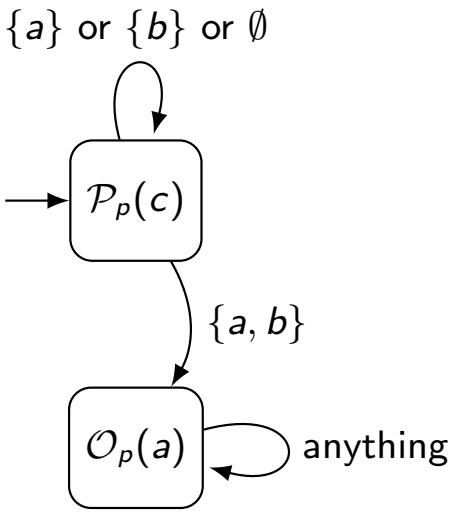
- A regulated system gives an automaton of all potential behaviours when composed.
- It is breach-incapable if no violating states and/or transitions are reachable from the initial state.
- This is stronger than being compliant for one specific run.

$$\text{correct}(p, R) \stackrel{df}{=} (\forall q \in \mathit{reachables}_S(R) \cdot \mathit{sat}_p(q)) \wedge (\forall t \in \mathit{reachable}_T(R) \cdot \mathit{sat}_p(t))$$

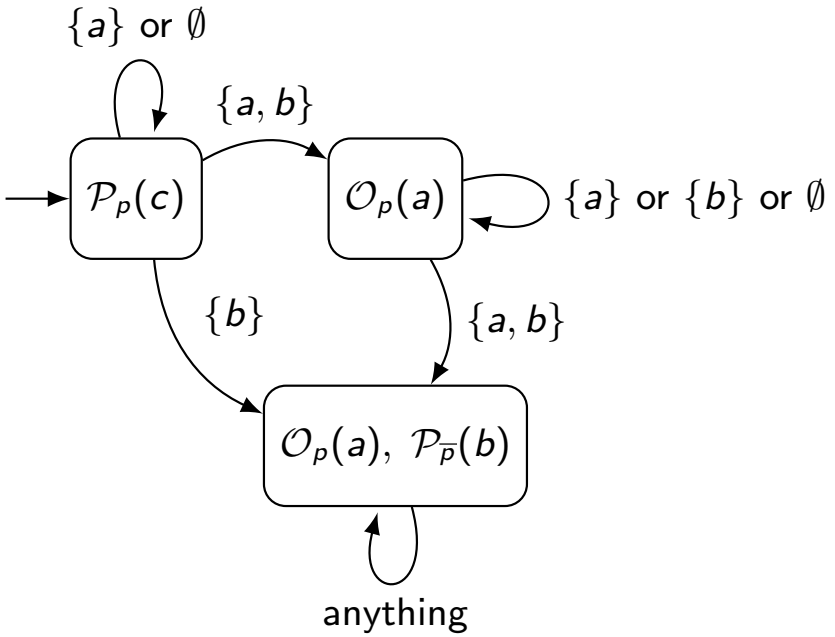
Ordering of contracts based on leniency

- A contract $\mathcal{A}$ is more lenient than another contract $\mathcal{A}'$ for a particular party p ($\mathcal{A} \sqsubseteq_p \mathcal{A}'$) if any system behaviour of p which may violate $\mathcal{A}$ may also violate $\mathcal{A}'$.
- This definition allows us to characterise the notion of contract equivalence for a particular party or even for all parties.

$$\mathcal{A} \sqsubseteq_p \mathcal{A}' \stackrel{df}{=} \forall S_1, S_2 \cdot \text{correct}(p, \langle S_1, S_2 \rangle_{\mathcal{A}'}) \implies \text{correct}(p, \langle S_1, S_2 \rangle_{\mathcal{A}})$$



$\sqsubseteq_p$

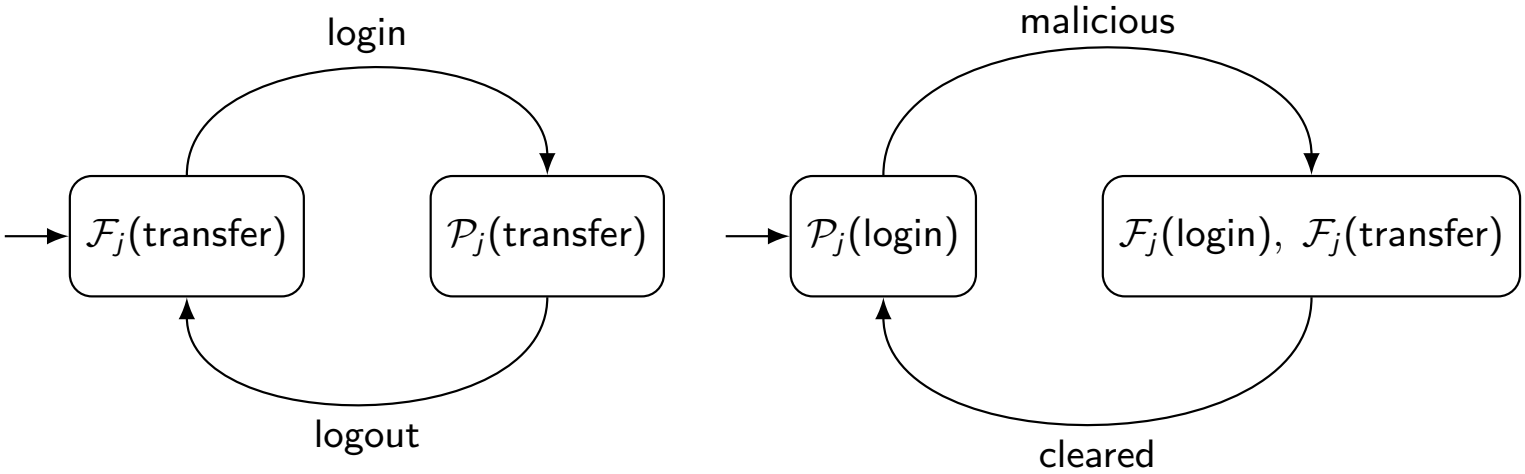


Conflicts in Contracts

Axiom 1: Opposite permissions conflict: $\vdash \mathcal{P}_p(x) \maltese !\mathcal{P}_p(x)$.
Axiom 2: Obligation to perform mutually exclusive actions is a conflict: $a \bowtie b \vdash \mathcal{O}_p(a) \maltese \mathcal{O}_p(b)$.
Axiom 3: Conflicts are closed under symmetry: $C \maltese C' \vdash C' \maltese C$.
Axiom 4: Conflicts are closed under increased strictness: $C \maltese C' \wedge C' \sqsubseteq C'' \vdash C \maltese C''$.

John and his Bank

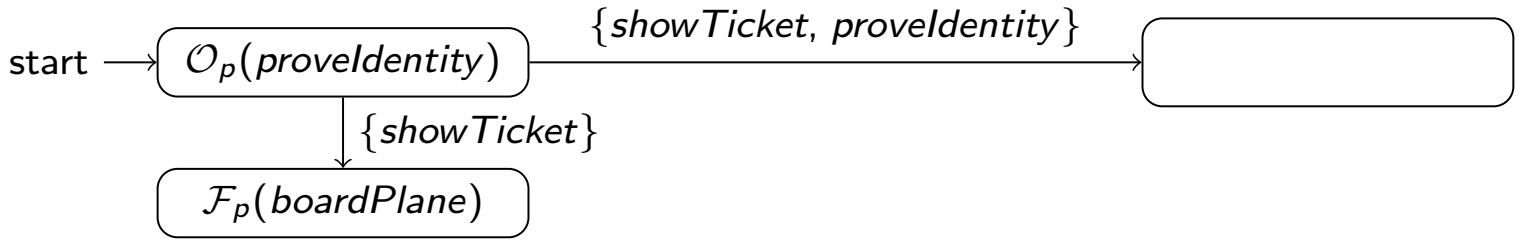
The contract says that (i) whenever he is logged into his Internet banking account, he is to be permitted to make money transfers; and (ii) if a malicious attempt to log in to his account is identified, logging in and making transfers will be prohibited until the situation is cleared.



Note what happens after $\{\text{login}, \text{malicious}\}$ in the composition of these contracts.

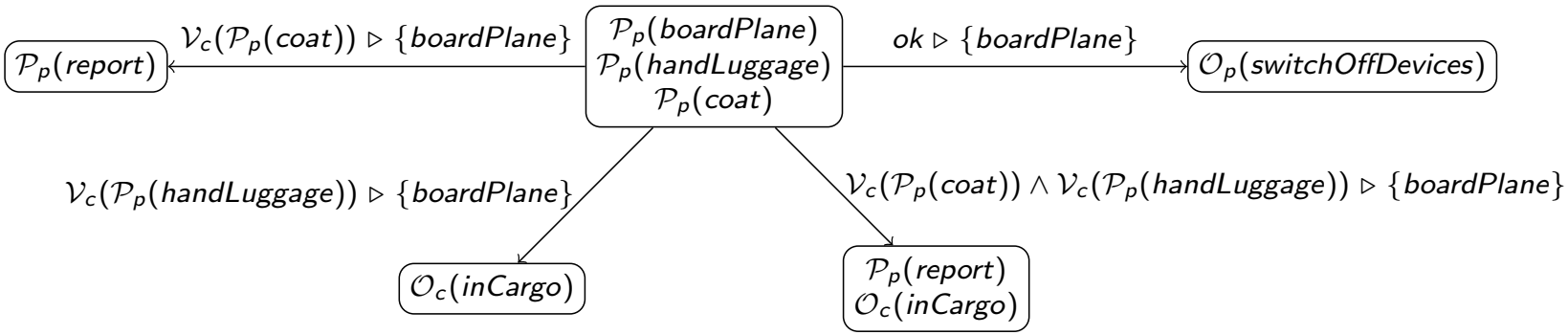
Adding Reparations

- Some reparations can be 'handled' by contract automata.
- e.g. *"The passenger is obliged to show a means of identification when presenting the ticket, and would otherwise be prohibited from boarding."*



- But with such an encoding, we have no notion of which transitions are violating ones.
- And even worse, we cannot express a reparation of a permission in this manner.

Reparations



- By tagging each transition with which clauses would have to be violated to take that transition, we can model reparations.
- In addition, we can talk about *hypothetical reparations*.

Discussion

- By keeping the contract as a separate automaton:
  - We share the same formalism and theory as for systems;
  - We are able to reason about contracts independently of the system — e.g. compose contracts using synchronous composition
- Although we can reduce a regulated system to a single automaton, by keeping the original systems and contract as automaton we keep the system behaviours separate and intact. Permission can only be deduced with this unadulterated behaviour.

Conclusions

- The main contributions of this work are:
  - A formalisation of the meaning of directed deontic operators in a two-party setting;
  - The use of standard techniques from computer science, namely communication and synchronisation, to analyse contracts regulating two parties.
- Practically all the work done on directed obligations and permissions introduces new modalities such as intention, causality, etc.
- We are now looking into how much further such an automaton based approach supports contract analysis, and extending it to deal with multi-party ($n > 2$) systems.