Permissions in Contracts, a Logical Insight[1]

Gordon J. PACE [a] Fernando SCHAPACHNIK [b]
[a] *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. Despite the fact that contracts are, by definition, an agreement between two or more parties, most formal studies limit themselves to contracts regulating only a single party or the parties independently of each other, without looking into how permissions, obligations or prohibitions of one party affect the other. This article deals with the analysis of what different types of permissions mean in the context of contracts. To give formal semantics we use an automata based formalism allowing to model for one party agreeing, delaying or plain refusing on performing certain actions that the other is attempting. This approach also yields a natural notion of contract strictness analysis for each party.

Keywords. Automated Legislative Drafting, Contract Verification, Permissions

1. Introduction

You sign a service agreement with your bank that states that you are *permitted* to withdraw your money at any time. It also stipulates that the bank is open only during working hours. Is there something wrong with this contract? You think there is and go to another bank that will let you withdraw your money at any time. But then, before each withdrawal it requires you to sign more and more papers. The first time you consider that the delaying in complying to the withdrawal request is justified, but as more and more requirements appear, you start to think that they are violating the contract as they are *obliged* to give you your money. In a more general way, does the permission for one party impose some kind of obligation to the other in a two-party contract? If so, what is the nature of this obligation? Should the other party allow for every intent of the exercise of the permission, or it suffices to eventually allow for one?

This article deals with the analysis of what different types of permissions mean in the context of contracts, but because deontic modalities are interrelated, we also analyse obligations and prohibitions. In an action based deontic logic, in which each party's actions are regulated by contract clauses, interaction between the parties is crucial. If an agreement with a bank states that a client is permitted to add new signatories to an account, the action of adding a new signatory is one which is shared by the client and the bank. Indeed, the clause giving permission to perform this action puts the onus on

[1] Partially supported by UBACyT 20020100200103.

the bank to accept such a request from the client. This implies a notion of synchronisation between the parties over certain actions, allowing the parties to agree, delay or plain refuse performing certain actions that the other is attempting. Despite the fact that contracts are, by definition, an agreement between two or more parties, most formal studies limit themselves to contracts regulating only one party and do not analyse how permissions, obligations or prohibitions for one party affect the other, something we study in this article.

2. Regulated Two-Party Systems

As argued earlier, to enable more direct reasoning about permission, we require a model in which the two parties agree on actions to perform. We use the notion of synchronous composition [1] to model such behaviour. Furthermore, to be able to deal with concurrent obligations (for instance, one party being obliged to perform one action and the other being obliged to perform another), we adopt a multi-action labels on transitions, since if we do not, it is impossible not to violate a contract in which both parties have different obligations at the same time.

Definition 2.1 *A multi-action automaton S is a tuple $\langle \Sigma,\ Q,\ q0,\ \rightarrow \rangle$, 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 $(q, A, q') \in \rightarrow$,* next$(q)$ *to be the set of target state and action set pairs of transitions outgoing from q (defined to be $\{(A, q') \mid q \xrightarrow{A} q'\}$) and* acts$(q)$ *to be the set of all action sets on the outgoing transitions from q (defined to be $\{A \mid \exists q' \cdot q \xrightarrow{A} q'\}$). We say that an automaton is total, if for every $q \in Q$ and $A \subseteq \Sigma$, there is a $q' \in Q$ such that $q \xrightarrow{A} q'$.*

The synchronous composition of two automata $S_i = \langle Q_i,\ q0_i,\ \rightarrow_i \rangle$ for $i \in \{1, 2\}$ (both with alphabet Σ) synchronising over alphabet G, written $S_1 \|_G S_2$, and 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].

We can now define contracts to be automata with each state tagged with the contract which will be in force at that point.

Definition 2.2 *A* contract clause *over alphabet Σ is structured as follows (where action $a \in \Sigma$, party $p \in \{1, 2\}$, set of actions $A \subseteq 2^{\Sigma}$, permission $\pi \in$ Permission):*

$$\textit{Clause} ::= \mathcal{O}_p(a) \mid \mathcal{F}_p(a) \mid \textit{Permission} \qquad \textit{Permission} ::= \mathcal{P}_p(a) \mid \delta^{A}_{\leq n}(\pi)$$

We will define party(π) *to be the party to whom permission π refers, while we will write $\overline{p}$ whenever we want to refer to the party other than p.*
A contract automaton *is a total and deterministic multi-action automaton $S = \langle Q,\ q0,\ \rightarrow \rangle$, together with a total function* contract $\in Q \rightarrow 2^{\textit{Clause}}$ *assigning a set of clauses to each state. We will use $\mathcal{CA}$ to refer to the class of all contract automata.*

We can now define a regulated two-party system in terms of multi-action automata.

Definition 2.3 *A* regulated two-party system *synchronising over the set of actions G is a tuple $R = \langle S_1, S_2 \rangle^{\mathcal{A}}_{G}$, where $S_i = (\Sigma_i, Q_i, q0_i, \rightarrow_i)$ is a multi-action automaton specifying the behaviour of party i, and $\mathcal{A}$ is a contract automaton over alphabet $\Sigma_1 \cup \Sigma_2$.*

The behaviour of a regulated two-party system R, written $[\![R]\!]$, is defined to be the automaton $(S_1 \|_G S_2) \|_\Sigma \mathcal{A}$. To make states in such systems more readable, we will write $((q_1, q_2), q_\mathcal{A})$ as $(q_1, q_2)_{q_\mathcal{A}}$.

Note that the totality of the contract automaton guarantees that the system behaviour is not modified, but tags the states with the relevant contracts at each point in time.

3. Contract Satisfaction

Given a two-party system (S_1, S_2), and a contract automaton $\mathcal{A}$, we now need to define whether or not either party is violating the contract in a given execution when a particular state is reached or a transition is taken. As we will see, a dual-view of violation, identifying *both* bad states and transitions, is necessary in a deontic context. We will look at the different deontic operators and define the set of violations induced for each of them.

We will use functions $F_p(q_\mathcal{A})$ and $O_p(q_\mathcal{A})$ to refer to the set of actions respectively prohibited and obliged for party p. They are defined in terms of the contract clauses in the current state. For example, $F_p(q_\mathcal{A}) = \{a \mid \mathcal{F}_p(a) \in \mathit{contract}(q_\mathcal{A})\}$.

Since we would like to be able to place blame in the case of a violation, we parametrise contract satisfaction or violation by party.

It is also worth noting that while prohibition, for instance, is violated as a transition which includes the prohibited action is taken, 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 $\mathit{sat}_p(X)$ will denote that reaching state X or traversing transition X, does not constitute a violation for party p. X ranges over states and transitions in the composed system. We start by defining separate satisfaction predicates for the deontic operators.

Permission. 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. Permission to perform local actions can never be violated. In the case of a single permission, this can be expressed as follows:

$$(q_1, q_2)_{q_\mathcal{A}} \vdash_p \mathcal{P}_{\overline{p}}(a) \stackrel{df}{=} a \in G \implies \exists A \in \mathit{acts}(q_p) \cdot a \in A \wedge A \cap F_{\overline{p}}(q_\mathcal{A}) = \emptyset$$

While actual prohibition violations occur when an action takes place, violations of a permission occur when no appropriate action is possible. In this paper we give 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. Different semantics are also possible. For any other parameters, the permission is otherwise satisfied.

Delayed Permission. We use the notion of delayed permission, written $\delta^A_{\leq n}(\pi)$ to denote that party p has permission π, although up to n actions from action set A may occur before this permission is granted. This can be defined recursively over n in the following manner:

$$(q_1, q_2)_{q_\mathcal{A}} \vdash_p \delta^A_{\leq 0}(\pi) \stackrel{df}{=} (q_1, q_2)_{q_\mathcal{A}} \vdash_p \pi$$

$$(q_1, q_2)_{q_\mathcal{A}} \vdash_p \delta^A_{\leq n+1}(\pi) \stackrel{df}{=} (q_1, q_2)_{q_\mathcal{A}} \vdash_p \pi \vee$$
$$\forall (A', (q_1', q_2')_{q'_\mathcal{A}}) \in \mathit{next}((q_1, q_2)_{q_\mathcal{A}}) \cdot A' \subseteq A \wedge (q_1', q_2')_{q'_\mathcal{A}} \vdash_p \delta^{n+1}_{\leq A}(\pi)$$

To combine all permissions in a state, we simply take the conjunction of all conditions:

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

All transitions are taken as satisfying the permission satisfaction function.

Prohibition. Party p causes no violation of a single prohibition whenever it engages in a set of actions which does not include the forbidden action a. Generalising over all prohibitions in a state is equivalent to this holding for every prohibited action:

$$sat_p^F((q_1,q_2)_{q_\mathcal{A}} \xrightarrow{A} (q_1',q_2')_{q'_\mathcal{A}}) \stackrel{df}{=} A \cap F_p(q_\mathcal{A}) = \emptyset$$

The semantics we adopted tags *actual* violations and not *potential* or *intended* ones, the type of violation that would occur if a party is willing to perform the forbidden action but the other does not synchronise, and thus the action does not happen in a given run. Slight modifications to the semantics can be given to also flag these potential violations.

Obligation. Obligation brings 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; and (ii) the other party $\overline{p}$ must provide a viable action set which allows p to perform *all* its obligations:

$$sat_p^O((q_1,q_2)_{q_\mathcal{A}} \xrightarrow{A} (q_1',q_2')_{q'_\mathcal{A}}) \stackrel{df}{=} O_p(q_\mathcal{A}) \subseteq A$$
$$sat_p^O((q_1,q_2)_{q_\mathcal{A}}) \stackrel{df}{=} \exists A \in acts(q_{\overline{p}}) \cdot O_p(q_\mathcal{A}) \subseteq A \wedge A \cap F_p(q_\mathcal{A}) = \emptyset$$

General contract satisfaction. It is defined as $sat_p(X) \stackrel{df}{=} sat_p^P(X) \wedge sat_p^O(X) \wedge sat_p^F(X)$

Definition 3.1 *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.*

Note that being breach-incapable is stronger than just being compliant for one specific run — *breachIncapable*$_p(R)$ means that there is no possible trace in which p breaches the contract. We can now define strictness relationships over contracts.

Definition 3.2 *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,* breachIncapable$_p(\langle S_1, S_2 \rangle_G^{\mathcal{A}'}) \implies$ breachIncapable$_p(\langle S_1, S_2 \rangle_G^{\mathcal{A}})$. *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 $\mathcal{A}' \sqsubseteq_p \mathcal{A}$. 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 $\mathcal{A} = \mathcal{A}'$.*

Although contracts are expressed as automata, we would like to be able to compare individual clauses. To do this, we will need to relate contract automata which are equivalent except for a particular clause replaced by another.

Definition 3.3 *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 for contract clauses. We say that clause C' is stricter than clause C, written $C \sqsubseteq C'$, if for any contract automata $\mathcal{A}$ and $\mathcal{A}'$ such that $(\mathcal{A}, \mathcal{A}') \in [C \to C']$, it follows that $\mathcal{A} \sqsubseteq \mathcal{A}'$.

Usually one would equate prohibition from performing an action with lack of permission to do so. However, in our context, this law holds only in one direction.

Theorem 3.1 *If we define the deontic modality $\overline{\mathcal{P}}$ such that the semantics of $\overline{\mathcal{P}}_p(a)$ are the exact negation of those of $\mathcal{P}_p(a)$, then $\mathcal{F}_p(a) \sqsubseteq \overline{\mathcal{P}}_p(a)$, but not $\overline{\mathcal{P}}_p(a) \sqsubseteq \mathcal{F}_p(a)$.*

The implication is that if an agent at runtime, violates a prohibition she is also violating lack of permission, but conversely, it may be possible to satisfy a prohibition by not doing the prohibited action while at the same time satisfying a permission to the same action: the agent is allowed to perform the action, but chooses not to do so. Using a similar analysis, we can compare the strictness of obligations and permissions:

Theorem 3.2 *Obligation is stricter than permission: $\mathcal{P}_p(a) \sqsubseteq \mathcal{O}_p(a)$. For synchronised actions, obligation for one party is stricter than permission for the other: $\mathcal{P}_p(a) \sqsubseteq \mathcal{O}_{\overline{p}}(a)$. The inequality does not hold in the opposite direction.*

It is interesting to note, that without blame-identification, in a synchronous world, one could show equivalence between $\mathcal{O}_p(a)$ and $\mathcal{O}_{\overline{p}}(a)$, since a lack of a on a transition would cause a violation of both obligations. However, since our partial order $\sqsubseteq_p$ is parametrised by the party, one can show that the two obligations are in fact different.

4. Conclusions

[2] deals with obligation violations in contracts using the domain specific BCL language, including the use of directed obligations, but does not analyse the reciprocity of deontic clauses in a contract. [4] aims at formalisations of contracts for e-commerce, presents some notions of deadlines and repetitions, but focuses only on analysing temporal consistency. A related line of research was started by [3]. It considers obligations of one individual towards another, termed *directed obligations*. *Directed permissions* have also been studied, but were considered to be conflicting because of lack of a clear counterparty, following both the *claimant theory* or the *benefit theory*. The synchronous nature of some actions in our approach is what makes the concept meaningful.

We have presented a model of deontic modalities in contracts focusing on (i) the role played by the interaction between the parties in the contract satisfiability; and (ii) different forms of permission. Contracts in which parties are bound to grant permission to each other turn out to be semantically rich and many questions arise as to the details of how they behave. One of these is the notion of contract strictness for a particular party. We chose to derive the ordering as a consequence of the semantics and satisfiability of the clauses, rather than by axiomatically defining an ordering for each modality.

References

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

[2] 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.

[3] H. Herrestad and C. Krogh. Deontic logic relativised to bearers and counterparties. *Anniversary Anthology in Computers and Law*, pages 453–522, 1995.

[4] Olivera Marjanovic and Zoran 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.