Conditional Permissions in Contracts [1]

Gordon J. PACE [a] Fernando SCHAPACHNIK [b] Gerardo SCHNEIDER [c]
[a] *gordon.pace@um.edu.mt*
University of Malta, Malta
[b] *fschapachnik@dc.uba.ar*
Universidad de Buenos Aires, Buenos Aires, Argentina
[c] *gerardo@cse.gu.se*
University of Gothenburg, Sweden

Abstract. Defining and characterising conditional permissions has never been easy. Part of the problem, we believe, comes from the fact that there is not one but a whole family of possible deontic operators, all of them distinct and reasonable, that can be labelled as conditional permissions. In this article, rather than disputing the correct interpretation, we revisit a number of different interpretations the term has received in the literature, and propose appropriate formalisations for these interpretations within the context of contract automata.

Keywords. Deontic Logic, Automata, Normative Systems, Interactive Two-Party Systems, Contract Representation, Conditional Permission

1. Introduction

The notions of permission and the notion of conditionality must be amongst the hardest-to-formalise deontic concepts. Conditional permission combines the difficulties from both terms and has proved to be specially elusive. Part of the problem, we believe, comes from the fact that there is not one but many possible deontic operators, distinct yet all providing a reasonable interpretation of the term. The general consensus is that conditional permissive norms have a condition (let us call it φ) and a normative consequence of that condition, that can be either a state of affairs or an action or action set that becomes permitted.

The very notion of clauses being conditional is at heart of Makinson's and van der Torre's I/O logics [6]: "Technically, a normative code is seen as a set G of conditional norms, i.e., a set of such ordered pairs (a, x). For each such pair, the body a is thought of as an input, representing some condition or situation, and the head x is thought of as an output, representing what the norm tells us to be desirable, obligatory or whatever in that situation". In [5] the authors deal with conditional permissions and claim that under the principle of *nihil obstat* (the one ruling the so called *negative permissions*), conditional permission is interpreted as

[1] Partially supported by UBACyT 20020130200032BA and the Swedish Research Council under grant nr. 2012-5746 (project *REMU: Reliable Multilingual Digital Communication: Methods and Applications).*

follows: "[...] a code permits something with respect to a condition iff it does not forbid it under that same condition, i.e. iff the code does not require the contrary under that condition". So, if we start with an empty code and ask whether a person younger than 18 can drink and can work, then the answer is yes in both cases. If we add a single clause that permits drinking on the condition of being at least 18 years old then if follows that for someone with 18 years of age it is permitted both to drink (explicitly) and to work (because *nihil obstat*). However, someone younger that 18 will be in violation if she drinks but not if she works. This is because there is a implicit notion that the fact of putting a condition on a permission is at the same time prohibiting it when the condition does not hold (essentially $\neg\varphi \implies F(x)$). Yet, this is not always the interpretation given.

Consider the notion of *antithetic permission* as presented by Stolpe in [8] (see also [4,3]): "The second form of implied permission, may be called *antithetic permission*, since such a permission (the thesis) overrules any prohibition (the antithesis) that is incompatible with it. Unlike exemption, the primary function of antithetic permissions is not to limit or suspend an existing prohibition, but rather to prevent one from being passed or practised in the first place."

Besides examples from constitutional law, one might consider university statutes that permit students over 21 years old to vote for School President explicitly, thus disallowing schools (which might have the power to pass their own ruling) to forbid it. In this case the intended meaning is not to prohibit voting if the condition does not hold (a particular school might want to permit voting starting from 18 or even without considering age), but to make sure that *at least* it is permitted in the given case. The rule $\neg\varphi \implies F(x)$ does not apply for conditional antithetic permissions. In this case it can be considered, following the words of Alchourrón and Bulygin [1], that it might be that "there is a norm permitting p; p is strongly permitted, but $\neg p$ is not regulated (it belongs to the extranormative sphere)". As we will argue later, this can be the case even in normative systems that are not hierarchical.

In the context of contracts there is a further complication when violations are considered, because some actions require active involvement of the other party. Contracts are general enough to be interesting and specific enough so many deontic aspects can be analysed in a constrained environment, yet they are not covered by many formal studies that regulate parties independently.

In contracts, the mere fact of permitting one to do a shared action, even without conditions involved, imposes an obligation on the other to synchronise with that action (for instance, a permission to buy that is also an obligation on the other to sell, much in the spirit of what is sometimes called a *right*). In this article we use contract automata [7], a formalism that allows to discuss interactions, while still tagging violations per party, to discuss different forms of conditional permissions. In the case of interacting systems, the notion of permission (unlike those of obligation and prohibition) refers to the possibility of potential behaviour in order to specify that *one of the parties does offer a viable way for the other party to perform a particular action*, in a manner analogous to that typically handled by branching time logics such as Computation Tree Logic (CTL). This is key to understanding one of the challenges in formally characterising conditional permissions which depend on a hybrid view of the system — the conditions typ-

ically refer to the present state of affairs, whereas permissions refer to potential states of affairs.

In order to avoid confusion with different interpretations given to terms by different authors, and because those are generally used in the context of general law and not specifically in the context of contracts, we will not refer to them as *positive*, *negative*, *antithetic*, *exceptional*, *static* or *dynamic* conditional permissions. Instead, this article will rely on formal semantics to present differences and similarities between the different interpretations — avoiding contentious discussions as to what is the generally accepted name to refer to each definition. The characterised forms of conditional permissions presented are i) those where the condition is *sufficient* for the permission to hold, but *not necessary* (type 1); ii) those where, in addition, the condition not holding transforms the permission into a prohibition: the condition is both *sufficient and necessary* (type 2); iii) those where the condition is *necessary*, but not *sufficient*, and additional conditions should hold (type 3); iv) those where the condition is *necessary*, but it is not something that has to hold but rather something that one of the parties has also to do (type 4).

In the next section we summarise key aspects of contract automata, and then discuss the different forms of conditional deontic clauses in Section 3. Section 4 concludes the article with some final remarks.

2. Contract Automata

To enable direct reasoning about contracts, one requires a model in which the two parties somehow interact to agree on which actions to perform. Here we present the key aspects of such a model (see [7] for full details):

- Each of the intervening parties is modelled as a multi-action automaton (Definition 2.1). Parties interact based on the notion of synchronous composition [2] and multi-action labels on transitions.
- Contracts that rule relationships among those parties are also multi-action automata tagged with deontic clauses, called *contract automata*.
- The synchronous composition of the parties and the contract is called a *regulated two-party system* (Definition 2.3). It provides the ability to predicate about deontic clauses enforced at each possible step of interaction between the two parties.

Definition 2.1 – Multi-Action Automaton
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* $\mathrm{acts}(q)$ *to be the set of all action sets on the outgoing transitions from* q *(defined to be* $\{A \mid \exists q' \cdot q \xrightarrow{A} q'\}$*).*

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* $\langle \Sigma,\ Q_1 \times Q_2,\ (q0_1, q0_2), \rightarrow \rangle$*, where* $\rightarrow$ *is the classical synchronous composition relation (e.g., [2]).*

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 an action. 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: $!\Sigma \stackrel{df}{=} \Sigma \cup \{!a \mid a \in \Sigma\}$.

Each clause in a contract automata refers to one of two parties, with the set of parties being Party $= \{1, 2\}$. We will use variables p, p_1 and p_2 to range over this type, and write $\overline{p}$ to refer to the party other than p (i.e., $\overline{1} = 2$ and $\overline{2} = 1$). 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). Note that being obliged not to perform an action is the same as being *forbidden* to perform the action ($\mathcal{F}_p(x) = \mathcal{O}_p(!x)$). We will use both expressions interchangeable in the rest of the paper.

Definition 2.2 – Contract Automaton
A contract clause *over alphabet Σ is structured as follows (where action $x \in !\Sigma$, party $p \in$ Party): Clause $::= \mathcal{O}_p(x) \mid \mathcal{P}_p(x)$.*

A contract automaton *is a total and deterministic multi-action automaton $S = \langle \Sigma,\ Q,\ q0,\ \rightarrow \rangle$, together with a total function* contract $\in Q \rightarrow 2^{Clause}$ *assigning a set of clauses to each state.*

As we are dealing here only with two-party contracts, we define next *regulated two-party systems* as being an automaton composed by the behaviour of the two parties together with a contract automaton among them.

Definition 2.3 – Regulated Two-Party System
A regulated two-party system *synchronising over the set of actions G is a tuple $R = \langle S_1, S_2 \rangle_G^{\mathcal{A}}$, where $S_p = (\Sigma_p, Q_p, q0_p, \rightarrow_p)$ is a multi-action automaton specifying the behaviour of party $p \in$ Party 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 $(S_1 \|_G S_2) \|_\Sigma \mathcal{A}$. We will write $((q_1, q_2), q_{\mathcal{A}})$ as $(q_1, q_2)_{q_{\mathcal{A}}}$.*

A regulated two-party system is well-formed *if $S_1 \|_G S_2$ never deadlocks: $\forall (q_1, q_2) \cdot \text{acts}((q_1, q_2)) \neq \emptyset$.*

In the rest of the paper we will assume that all systems are well-formed. One way of guaranteeing this may be by having all system states provide a transition with the empty action set. Also note that the totality of the contract automaton guarantees that the system behaviour is not constrained, but simply acts to tag the states with the relevant contracts at each point in time. 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 2.4 – Viable Action Sets
Given a contract automaton $\mathcal{A} = \langle \Sigma,\ Q,\ q0,\ \rightarrow \rangle$, and a state $q_{\mathcal{A}} \in Q$, functions

$O_p(q_\mathcal{A})$ and $F_p(q_\mathcal{A})$ give the set of actions respectively obliged to be performed and forbidden to be performed by party p:

$O_p(q_\mathcal{A}) \stackrel{df}{=} \{a \mid \mathcal{O}_p(a) \in \text{contract}(q_\mathcal{A})\}$ $\quad$ $F_p(q_\mathcal{A}) \stackrel{df}{=} \{a \mid \mathcal{O}_p(!a) \in \text{contract}(q_\mathcal{A})\}$

Action set A is said to be viable for party p in a contract automaton state $q_\mathcal{A} \in Q$, written $\text{viable}_p(q_\mathcal{A}, A)$*, if (i) all her obliged actions are included in A but; (ii) no actions which the party is forbidden to perform are included in A:*

$$\text{viable}_p(q_\mathcal{A}, A) \stackrel{df}{=} O_p(q_\mathcal{A}) \subseteq A \wedge F_p(q_\mathcal{A}) \cap A = \emptyset.$$

For space reasons in what follows we define permissions and obligations in an informal manner; we refer the reader to [7] for a full formalisation.

Definition 2.5 – 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. 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.*

Definition 2.6 – 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.*

As an example let us consider the case when p is permitted to withdraw money from the bank, permitted not to deposit, obliged to pay the fee, and obliged not to steal ($\mathcal{P}_p(w)$, $\mathcal{P}_p(!d)$, $\mathcal{O}_p(f)$, $\mathcal{O}_p(!s)$). Then $\overline{p}$ should provide at least one transition that contains both a w and an f but does not contain d nor s.

2.1. Trace Semantics

In order to present richer conditional operators (see Section 3) we need to be able to analyse the deontic status of specific traces, as defined below.

Definition 2.7 – Regulated Contract Trace
Given a regulated two-party system $R = \langle S_1, S_2 \rangle^\mathcal{A}_G$, a regulated contract trace *is any run of $[\![R]\!]$: $(q_1^0, q_2^0)_{q_\mathcal{A}^0} \xrightarrow{A^0} (q_1^1, q_2^1)_{q_\mathcal{A}^1} \xrightarrow{A^1} \ldots$*

Definition 2.8 – Trace-Based Modalities
Permission *Our model characterises permission as the provision of viable options for the other party regardless of whether the other party takes them or not. Because of that, we do not tag permission violations in traces.*

Obligation *Each party satisfies its obligation in a transition if the action set is viable for her and for the other party. I.e., she included all her mandatory actions, along with the ones of the other party, and did not include any forbidden ones.*

$$\text{sat}_p^{\mathcal{O}}((q_1^i, q_2^i)_{q_{\mathcal{A}}^i} \xrightarrow{A^i} (q_1^{i+1}, q_2^{i+1})_{q_{\mathcal{A}}^{i+1}}) \stackrel{df}{=} \text{viable}_p(q_{\mathcal{A}}^i, A^i) \wedge \text{viable}_{\overline{p}}(q_{\mathcal{A}}^i, A^i).$$

Violations of prohibitions can be identified using $\mathcal{F}_p(x) = \mathcal{O}_p(!x)$.

2.2. Enriched Trace Semantics

Conditions are predicates over an (as yet) implicit state of the system[2]. To go beyond basic modalities and support conditional ones we thus need to enrich the traces with this state information.

Definition 2.9 – Enriched Contract Trace
Given a regulated contract trace $(q_1^0, q_2^0)_{q_{\mathcal{A}}^0} \xrightarrow{A^0} (q_1^1, q_2^1)_{q_{\mathcal{A}}^1} \xrightarrow{A^1} \ldots$ *we call* enriched contract trace *a trace of the form* $(q_1^0, q_2^0)^{\theta^0}_{q_{\mathcal{A}}^0} \xrightarrow{A^0} (q_1^1, q_2^1)^{\theta^1}_{q_{\mathcal{A}}^1} \xrightarrow{A^1} \ldots$ *where each* θ^i *is a state-predicate in an unspecified logic.*

Note that we do not pose any requirements on the underlying logic for θ, allowing it to be determined depending on the intended domain of application, clearly affecting the complexity of the analysis (including decidability issues).

3. Conditional Permissions

In this section we introduce four types of conditional permissions. The first two of them try to characterise the two different intended meaning of conditional permissions raised by the example of the university statute discussed in the introduction: when the condition is sufficient but not necessary, and when it is both sufficient and necessary. The third and four types include cases when additional actions or facts are needed in order for the conditional permission to be satisfied.

Type 1 Conditional Permission
We will define as *type 1 conditional permission* the one where the condition is *sufficient* for the permission to hold, but it is *not necessary*. In the given example being emancipated could also be a trigger for the permission. To formalise it we will consider enriched contract traces and parties $p_1, p_2 \in \text{Party}$.

Definition 3.1 – Type 1 Conditional Permission
Party p *satisfies party* $\overline{p}$*'s* type 1 conditional permission $\mathcal{P}^1_{\overline{p}}(\varphi \triangleright a)$ *over an enriched transition* $(q_1, q_2)^{\theta}_{q_{\mathcal{A}}} \xrightarrow{A} \ldots$ *iff* θ *satisfying the condition* φ *implies that there is an action set which is viable for* $\overline{p}$ *and contains action* a[3].

[2] We can associate these predicates to the states of the contract automaton, but it makes more sense to keep the state over which conditions depend to be distinct from the state of the contract.

[3] Recall that only shared actions (actions in G) can be restricted by the other party.

$$(q_1^i, q_2^i)^{\theta^i}_{q^i_{\mathcal{A}}} \xrightarrow{A^i} (q_1^{i+1}, q_2^{i+1})^{\theta^{i+1}}_{q^{i+1}_{\mathcal{A}}} \vdash_{p_1} \mathcal{P}^1_{p_2}(\varphi \triangleright a) \stackrel{df}{=}$$

$$\begin{cases} \theta^i \vdash \varphi \wedge a \in G \implies & \\ \quad \exists A' \in \text{acts}(q^i_{p_1}), A'' \in (\Sigma - G) \cdot & \\ \quad\quad a \in A' \wedge \text{viable}_{p_2}(q^i_{\mathcal{A}}, A' \cup A'') & \text{if } p_1 \neq p_2 \\ true & \text{if } p_1 = p_2 \end{cases}$$

Note that this conditional permission can be satisfied even if the actually taken action set (A^i) and the action set that contains the permitted actions (A') are not the same. In the university statute example, if the student is at least 21, the permission is satisfied even if voting is not present in A^i, but was present in any other of the transitions that the School offered at state (q_1, q_2). This is an interesting property shared also by the other types: the satisfaction of the permission has to do with *potentiality*, and as such, depends on the structure of the system and not on the particular branch taken by the trace. Yet, it is the enriched state of the trace that triggers the signalling of the violation. As an analogy, consider the prohibition of being drunk while driving. The subject is drunk or not independently of the driving, but only when he gets behind the wheel that property becomes of legal interest.

Type 2 Conditional Permission

Another interpretation of the university statute example can be given in which, whenever the condition does not hold, transforms the permission into a prohibition: that being 21 years old is both *sufficient and necessary*. This will be the case for *type 2 conditional permission*: the holding of the condition brings about the holding of the permission, and it is also required that the condition holds for the permission to be in effect. That is, the condition becomes *necessary and sufficient* condition for the permission to be effective. Another example might be "Passengers can unbuckle seat belts when the red light is off". Only when the light goes off passengers can opt to unbuckle, being that this is the only requirement for the permission to hold.

Note that in this case, a conditional permission where the condition does not hold leads to an additional prohibition which means that we need to enrich our notion of active prohibitions and viability to depend also on the conditions (and hence state θ):

$$F^+_p(q_{\mathcal{A}}, \theta) \stackrel{df}{=} F_p(q_{\mathcal{A}}) \cup \{a \mid \mathcal{P}^2_p(\varphi \triangleright a) \in \mathit{contract}(q_{\mathcal{A}}) \wedge \theta \nvdash \varphi\}$$
$$\mathit{viable}^+_p(q_{\mathcal{A}}, A, \theta) \stackrel{df}{=} O_p(q_{\mathcal{A}}) \subseteq A \wedge F^+_p(q_{\mathcal{A}}, \theta) \cap A = \emptyset.$$

Definition 3.2 – Type 2 Conditional Permission

Party p satisfies party $\overline{p}$*'s* type 2 conditional permission $\mathcal{P}^2_{\overline{p}}(\varphi \triangleright a)$ *over an enriched transition* $(q_1, q_2)^{\theta}_{q_{\mathcal{A}}} \xrightarrow{A} \ldots$ *if there is an action set A that is viable for* $\overline{p}$ *and contains action a if and only if* θ *satisfies the condition* φ *(in the particular case of* θ *not satisfying* φ*, it means that a should not be in* A^i*).*

$$(q_1^i, q_2^i)^{\theta^i}_{q^i_{\mathcal{A}}} \xrightarrow{A^i} (q_1^{i+1}, q_2^{i+1})^{\theta^{i+1}}_{q^{i+1}_{\mathcal{A}}} \vdash_{p_1} \mathcal{P}^2_{p_2}(\varphi \triangleright a) \stackrel{df}{=}$$

$$\begin{cases} true & \text{if } p_1 = p_2 \vee a \notin G \\ \exists A' \in \text{acts}(q^i_{p_1}), A'' \in (\Sigma - G) \cdot & \text{if } p_1 \neq p_2 \wedge a \in G \wedge \theta^i \vdash \varphi \\ \quad\quad a \in A' \wedge \text{viable}^+_{p_2}(q^i_{\mathcal{A}}, A' \cup A'', \theta^i) & \\ a \notin A^i & \text{if } p_1 \neq p_2 \wedge a \in G \wedge \theta^i \nvdash \varphi \end{cases}$$

Note that in the case of action a being a shared action and θ^i not satisfying φ we are blaming party p_1 if action a is present in A^i. If action a is shared then both parties had it in their actions sets prior to synchronisation, so p_1 is to blame for offering a when it should not have done so.

As before, a party always satisfies its own permissions. This decision is rather arbitrary, because it could perfectly be decided to blame p_1 for overindulging herself by having a in the action set even when the condition did not allow for it. We decide to choose a semantics where this responsibility lays into the other party.

Note that Definition 3.1 would also require the use of $viable^+(\cdot)$ if the two types would be used together.

Type 3 Conditional Permission
Consider now the case of "A ticket can be purchased provided that there are seats available" and "Tickets need to be bought three days in advance". Each requirement (seats being available and purchase date being at least three days ahead) is necessary for the permission to be effective, yet none of them alone is sufficient. Thus, if one would state each of them individually, the given condition would be *necessary*, yet not *sufficient*. This is the case for *type 3 conditional permission*, $\mathcal{P}^3_p(\varphi \triangleright a)$. This is similar to type 1 but a little bit more complex, because in order to exercise the action many conditions have to hold, and they could had be expressed in different deontic clauses.

Take the air ticket example, where bt is the buying ticket action, φ_1 stands for the condition that tickets are available and φ_2 for the condition that the departure date is at least three days in advance. If a given contract has both $\mathcal{P}^3_p(\varphi_1 \triangleright bt)$ and $\mathcal{P}^3_p(\varphi_2 \triangleright bt)$, both of them have to be considered. Contract automata assume full deontic information, i.e., given an enriched state $(q_1, q_2)^\theta_{q_\mathcal{A}}$, all relevant deontic clauses are in $q_\mathcal{A}$. Thus, if one would want to support type 3 conditional permissions it is sufficient to consider the conjunction of all the conditions imposed to an action in the given state and then fall back to type 1. Note that other formalisms that operate under an open world assumption need a more sophisticated definition of type 3 conditional permission.

Type 4 Conditional Permission
The first three types can be defined by giving their satisfaction predicates over extended contract traces. However, it could also be the case that the conditions are actually actions that the party has to do, as in the case of "You have to pay the ticket to watch the movie" or "You have to take your little brother with you if you want to go to play". The case is similar to type 1 in that the condition is necessary, but differs in that the condition is not something that has to hold but rather something that party has also to do. I.e, in order to do a you have to do other things as well.

This scenario can sometimes be encoded as a type 1 conditional permission by sequentialising actions meant to be done concurrently. The example can be turned into "Tickets have to be *already paid* to watch the movie". Contract automata allows the expression of type 4 conditional permission without having to resort to

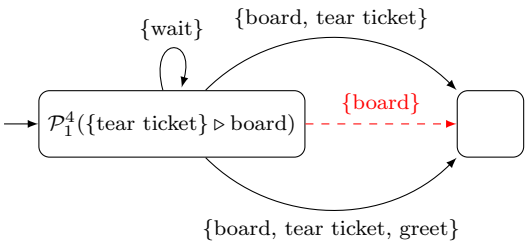

Figure 1. Type 4 Conditional Permission Example

such transformations, by taking advantage of transitions being labelled by sets of actions meant to be executed concurrently. The advantage of doing so being that violations of such type of conditional permission can be detected without having to consider specific traces.

As an example, consider Figure 1. It can be analysed with or without the dashed transition. Without it, both parties respect the permission: party 1 can choose to wait, which is a local action not involved into the boarding, or it can board with or without greeting. In both cases the tearing of the ticket is involved. If the dashed transition were present in the synchronised system both parties would be violating the conditional permission because both of them would be having a transition that allowed the boarding without the tearing. If neither of the solid transitions were present because of party 2's unwillingness to offer them, then party 2 would be in violation.

To fully present *type 4 conditional permission* we first need to define what it means for an action set to be required.

Definition 3.3 – Required Action Sets
Party p satisfies the requirement of having to do action set C in order to do a if all of its outgoing transitions that contain action a also contain the actions in C that are either local or shared (party p cannot assume responsibility for actions in C that are exclusive of the other party):

$$\text{requires}_p(q, C, a) \stackrel{df}{=} \forall A \in \text{acts}(q),\ a \in A\ \cdot\ ((C \cap (\Sigma_p \cup G)) \subseteq A).$$

Definition 3.4 – Type 4 Conditional Permission
Party p is permitted to do action *a* given condition $C \subseteq 2^{!Sigma}$, *written* $\mathcal{P}_p^4(C \triangleright a)$, *if there is a transition which makes it possible to do* $\mathcal{P}_p(a)$ *and also contains C, and there's no way of doing a without doing C as well. In other words, doing C is* imposed *in order to do a.*

Party p complies with $\mathcal{P}_p^4(C \triangleright a)$ *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.*

$$(q_1, q_2)_{q_\mathcal{A}} \vdash_{p_1} \mathcal{P}_{p_2}^4(C \triangleright a) \stackrel{df}{=}$$
$$\begin{cases} \text{requires}_{p_1}(q_{p_1}, C, a) & \text{if } p_1 = p_2 \\ \text{requires}_{p_1}(q_{p_1}, C, a)\ \wedge \exists A \in \text{acts}(q_{p_1}),\ A' \subseteq (\Sigma - G)\ \cdot & \\ \quad \text{viable}_{p_2}(q_\mathcal{A}, A \cup A') \wedge C \cup \{a\} \subseteq (A \cup A') & \text{if } p_1 \neq p_2 \end{cases}$$

In the above definition, the set C can contain either local or synchronised actions. That allows to express requirements to the same individual (e.g., "You need to show your boarding pass while entering the plane"), or even more complex interactions requiring the other party involvement (e.g., "You need to have your ticket torn upon entering the plane").

Note that we could have given an alternative definition, 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.

4. Conclusions and Future Work

In this article we have formally characterised four different forms of conditional permissions found in the literature, namely i) those where the condition is *sufficient* for the permission to hold, but it is *not necessary* (type 1); ii) those where, in addition, the condition not holding transforms the permission into a prohibition: the condition is both *sufficient and necessary* (type 2); iii) those where the condition is *necessary*, but not *sufficient*, and additional conditions should hold (type 3); iv) those where the condition is necessary, but it is not something that has to hold but rather something that one of the parties has also to do (type 4).

All of the presented permissions are different and have proper use cases, thus suggesting that "conditional permission" describes not one but a family of operators. In future work we plan to address conditional obligations to explore whether variations described in the literature can be accommodated under one formal presentation or they also describe a set of modalities.

References

[1] C.E. Alchourrón and E. Bulygin. Permission and permissive norms. *Theorie der Normen, Duncker & Humblot, Berlin*, 1984.

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

[3] Guido Boella and Leendert van der Torre. Permissions and obligations in hierarchical normative systems. In *ICAIL'03*, pages 109–118. ACM, 2003.

[4] E. Bulygin. Permissive norms and normative systems. In A. Martino and F. Socci Natali, editors, *Automated Analysis of Legal Texts*, pages 211–218. Publishing Company, 1986.

[5] D. Makinson and L. van der Torre. Permission from an input/output perspective. *Journal of Philosophical Logic*, 32(4):391–416, 2003.

[6] David Makinson and Leendert W. N. van der Torre. What is Input/Output Logic? Input/Output Logic, Constraints, Permissions. In *Normative Multi-agent Systems*, volume 07122 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2007.

[7] Gordon J. Pace and Fernando Schapachnik. Contracts for Interacting Two-Party Systems. In *FLACOS'12*, volume 94 of *ENTCS*, 2012.

[8] Audun Stolpe. A theory of permission based on the notion of derogation. *J. Applied Logic*, 8(1):97–113, 2010.