Synchronous Agents, Verification, and Blame — A Deontic View

Karam Kharraz[1], Shaun Azzopardi[2], Gerardo Schneider[2], and Martin Leucker[1]

[1] ISP, University of Lübeck, Germany {kharraz,leucker}@isp.uni-luebeck.de
[2] University of Gothenburg, Sweden {shauna,gersch}@chalmers.se

Abstract. A question we can ask of multi-agent systems is whether the agents' collective interaction satisfies particular goals or specifications, which can be either individual or collective. When a collaborative goal is not reached, or a specification is violated, a pertinent question is whether any agent is to blame. This paper considers a two-agent synchronous setting and a formal language to specify when agents' collaboration is required. We take a *deontic* approach and use *obligations*, *permissions*, and *prohibitions* to capture notions of non-interference between agents. We also handle *reparations*, allowing violations to be corrected or compensated. We give trace semantics to our logic, and use it to define blame assignment for violations. We give an automaton construction for the logic, which we use as the base for model checking and blame analysis. We also further provide quantitative semantics that is able to compare different interactions in terms of the required reparations.

1 Introduction

Interaction between agents can be adversarial, where each agent pursues its own set of individual goals, or cooperative where the agents collaborate to achieve a collective goal. Verification techniques can help us detect whether such goals may be achieved. Agents may also interfere or not cooperate, at which point the failure to achieve a goal could be attributed to some agent. In this paper, we develop a *deontic* logic allowing us to specify the anticipated interaction of two agents in the presence of such aspects.

A deontic logic [16,21] includes norms as first-class concepts, with *obligations*, *permissions*, and *prohibitions* as basic norms. These concepts are crucial in legal documents and contractual relationships, where the agents are the parties to a contract.[3] Norms are parameterised by actions/events or propositions and are used to specify what *ought to be*, or the parties *ought to do*.

In this paper, interaction or cooperation of the agents is modelled as the interplay of the individual actions performed by each agent, leading to the concept of cooperative actions. Cooperative actions could be synchronous, i.e., actions at each time point of each agent are meant to describe the possible cooperation,

[3] We use *party* and *agent* interchangeably throughout.

or asynchronous, meaning that actions for cooperation may happen a different time points.[4] We choose synchrony as an abstraction to simplify the concept of cooperation and non-interference between parties. We also study only the setting with two rather than many parties. As such, we are concerned with *two-party synchronous systems*, leaving extensions as future work.

We re-purpose and extend the syntax of a deontic language from literature [3,4] into a new deontic logic with denotational semantics appropriate for this two-party setting. Our semantics depends on two notions of *informative* satisfaction or violation, which talk about the exact point in time a contract is satisfied or violated. Other features of the logic include the ability to make contracts trigger on matching a regular language, requiring the satisfaction of a contract while one is still within the prefix language of a regular language, and a recursion operator to allow the definition of persistent contracts and repetition.

We extend the semantics with a notion of *blame assignment*, to identify which party is responsible for a certain violation. We further use this to define quantitative semantics that counts the number of violations caused by a certain party, which can be used to compare different traces or behaviour of a party.

We give an exponential automata construction for the logic, transforming a contract specification into an automaton capable of identifying satisfaction, and violation as specified in our semantics. We also provide a model checking algorithm, which is quadratic in the size of the contract automaton, hence exponential in the size of the contract. We re-use this construction for blame analysis, but leave analysis for the quantitative semantics for future work.

The paper organisation follows. Section 2 lays out preliminaries, Section 3 presents our logic, and Section 4 presents algorithms for model checking and blame analysis through automata constructions. Related work is considered in Section 5, and we conclude in Section 6.

2 Preliminaries

We write $\mathbb{N}_\infty$ for $\mathbb{N} \cup \{\infty\}$. Given a finite alphabet Σ, we write Σ_0, and Σ_1 for re-labellings of Σ with party identifiers 0 and 1, and $\Sigma_{0,1}$ for $\Sigma_0 \cup \Sigma_1$. We use $P[x/y]$ to refer to the syntactic replacement of x in P with y, where P can be an automaton (x and y are states), or a specification (x and y are syntactic objects in the language). We write $(*, s)$ to refer to all state pairs with s in the second position, and similarly for $(s, *)$.

Traces For $i \in \mathbb{N}$, $j \in \mathbb{N}_\infty$, and an infinite trace w over sets of actions from a finite alphabet Σ, we denote the trace between positions i and j by $w[i..j]$, including the values at both positions. If $j < i$ then $w[i..j]$ is the empty trace. When $j = \infty$ then $w[i..j]$ is the suffix of w from i. We write $w[i]$ for $w[i...i]$, and $w \cdot w'$ for concatenation of w and w', which is only defined for a finite word w.

Given two traces w, w' over 2^Σ, we define stepwise intersection: $(w \sqcap w')[i] \stackrel{\text{def}}{=} w[i] \cap w'[i]$, union $(w \sqcup w')[i] \stackrel{\text{def}}{=} w[i] \cup w'[i]$, and union with party labelling: $(w \sqcup^0_1 w')[i] \stackrel{\text{def}}{=} w[i] \cup^0_1 w'[i]$, where $E \cup^0_1 E' \stackrel{\text{def}}{=} \{a_0 \mid a \in E\} \cup \{a_1 \mid a \in E'\}$, i.e.

[4] Observe similarities with synchronous and asynchronous communication.

the left actions are labeled by 0 and the right actions by 1. This gives a trace in $\Sigma_{0,1}$. For instance, given $w = \langle\{a\},\{b\},\{c,d\}\rangle$ and $w' = \langle\{a\},\{e\},\{d,e\}\rangle$, we have that $w[2] \cap w'[2] = \{c,d\} \cap \{d,e\} = \{d\}$ and $w[2] \sqcup_1^0 w'[2] = \{c,d\} \sqcup_1^0 \{d,e\} = \{c_0, d_0, d_1, e_1\}$.

Given two traces w_0 and w_1, over 2^Σ, we write $\boldsymbol{w}_i^j$ for the pair $(w_0[i..j], w_1[i..j])$. $\boldsymbol{w}_i^j$ is said to be an *interaction*, and when $j \in \mathbb{N}$ a *finite interaction*. Sometimes we abuse notation and treat $\boldsymbol{w}_i^j$ as a trace in $\Sigma_{0,1}$, since it can be projected into such a trace through $\sqcup_1^0$.

Automata A tuple $A = \langle \Sigma, Q, q_0, Rej, \rightarrow \rangle$ is an *automaton*, where Σ is a finite alphabet, S is a finite set of states, $s_0 \in S$ is the initial state, $Rej \subseteq S$ is a set of rejecting states, and $\rightarrow \in S \times 2^\Sigma \rightarrow (2^S \setminus \emptyset)$ is the transition function ($\rightarrow \in S \times 2^\Sigma \rightarrow S$ when the automaton is deterministic). The language $L(A)$ of automaton A is the set of infinite traces with no prefix reaching a rejecting state. The rejecting language $RL(A)$ of automaton A is the set of infinite traces with a prefix reaching a rejecting state. We write $RL_s(A)$ for the rejecting language through a specific rejecting state $s \in Rej$.

The *synchronous product* of automata A and B over the same alphabet Σ, denoted by $A \| B$, is the automaton: $(\Sigma, S_A \times S_B, (s_{0_A}, s_{0_B}), (Rej_A \times S_B) \cup (S_A \times Rej_B), \rightarrow)$ where $\rightarrow$ is the minimal relation such that: for any $E \subseteq \Sigma$, if $s_1 \xrightarrow{E}_A s_1'$ and $s_2 \xrightarrow{E}_B s_2'$ then $(s_1, s_2) \xrightarrow{E} (s_1', s_2')$.

The *relaxed synchronous product* of automata A and B over the same alphabet Σ, denoted by $A \|^r B$ includes $A \| B$ but allows moving independently when there is no match: if $s_1 \xrightarrow{E}_A s_1'$ and $\nexists s_2' \cdot s_2 \xrightarrow{E}_B s_2'$, then $(s_1, s_2) \xrightarrow{E} (s_1', s_2)$; and symmetrically.

Moore Machines A Moore machine is a 5-tuple $M = (S, s_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Σ_I and Σ_O are respectively the finite set of input and output actions, $\delta : S \times 2^{\Sigma_I} \rightarrow 2^S$ is a transition function that maps each state and inputs to a next state, and $\lambda : S \rightarrow 2^{\Sigma_O}$ is an output function that maps each state to a set of outputs.

The *product* of a Moore machine M_1 over input alphabet Σ_I and output alphabet Σ_O, and Moore machine M_2 with flipped input and output alphabets is the automaton: $M_1 \otimes M_2 \stackrel{\text{def}}{=} (\Sigma_I \cup \Sigma_O, S_1 \times S_2, (s_{0_1}, s_{0_2}), \emptyset, \rightarrow)$ where $\rightarrow$ is the minimal relation such that: for any states $s_1 \in S_1$ and $s_2 \in S_2$, where $s_1 \xrightarrow{\lambda_2(s_2)} s_1'$ and $s_2 \xrightarrow{\lambda_1(s_1)} s_2'$ then $(s_1, s_2) \xrightarrow{\lambda_1(s_1) \cup \lambda_2(s_2)} (s_1', s_2')$.

Regular Expressions We use standard syntax for regular expressions. We treat as atomic boolean combinations of actions from $\Sigma_{0,1}$. The operators are standard: choice, $re + re$ (match either); sequence, $re; re$ (match the first then the second) and the Kleene plus, re^+ (match a non-zero finite amount of times in sequence). The language of a regular expression re is a set of finite traces: $L(re) \subseteq (2^{\Sigma_{0,1}})^*$. We abuse notation and write $\boldsymbol{w}_i^j \in L(re)$ for $w_0[i...j] \sqcup_1^0 w_1[i...j] \in L(re)$.

We restrict attention to the *tight language* of a regular expression, containing matching finite traces that have no matching strict prefix: $TL(re) \stackrel{\text{def}}{=} \{\boldsymbol{w}_i^j \in L(re) \mid \nexists k : k < j \wedge \boldsymbol{w}_i^k \in L(re)\}$. The *prefix closure* of the tight language is the

set of finite prefixes of the tight language up to a match: $cl(re) \stackrel{\text{def}}{=} \{\boldsymbol{w}_i^k \mid \exists j : \boldsymbol{w}_i^j \in TL(re) \wedge i \leq k < j\}$. We define the *complement of the prefix closure* as the set of finite traces that do not tightly match the regular expression but whose maximal strict prefix is in the closure of the expression: $\overline{cl}(re) \stackrel{\text{def}}{=} \{\boldsymbol{w}_i^j \mid (\boldsymbol{w}_i^{j-1} \in cl(re) \wedge \boldsymbol{w}_i^j \notin cl(re) \wedge \boldsymbol{w}_i^j \notin TL(re))\}$.

We denote by $A(re, s_0, s_\checkmark, s_\times)$ the deterministic finite automaton corresponding to regular expression re, with s_0, and $s_\times$ respectively as the initial and rejecting states and, $s_\checkmark$ as a sink state, s.t. $\forall \boldsymbol{w}_i^j \in TL(re) : s_0 \stackrel{\boldsymbol{w}_i^j}{\Longrightarrow} s_\checkmark$, $\forall \boldsymbol{w}_i^j \in cl(re) : \exists s : s_0 \stackrel{\boldsymbol{w}_i^j}{\Longrightarrow} s \wedge s \neq s_\checkmark \wedge s \neq s_\times$, and $\forall \boldsymbol{w}_i^j \in \overline{cl}(re) : s_0 \stackrel{\boldsymbol{w}_i^j}{\Longrightarrow} s_\times$.

3 A Deontic Logic for Collaboration

In this section, we present the syntax and semantics of $c\mathcal{DL}$, a deontic logic able to express the extent to which parties should cooperate and non-interfere.

Definition 3.1 $c\mathcal{DL}$ **Syntax.** *A $c\mathcal{DL}$ contract C is given by the following grammar, given an alphabet Σ, regular expressions re, a set of variables $\mathbb{X}$, and party labels p from $\{0, 1\}$:*

$$\begin{aligned} a \in\ & \Sigma_0 \cup \Sigma_1 \\ N :=\ & O_p(a) \mid F_p(a) \mid P_p(a) \mid \top \mid \bot \\ C :=\ & N \mid C \wedge C \mid C; C \mid C \blacktriangleright C \mid \\ & \langle re \rangle C \mid re \,\text{!`}\, C \mid \mathbb{X} \mid recX.C \end{aligned}$$

Our setting is that of two-party systems, with one party indexed with 0 and the other with 1. As the basic atoms of the language, we have *norms*. These norms are labeled by the party that is the main subject of the norm, and the action that is normed: $O_p(a)$ denotes an obligation for party p to achieve a; $F_p(a)$ denotes a prohibition for party p from achieving a, and $P_p(a)$ denotes a permission/right for party p to achieve a.

We call $c\mathcal{DL}$ specifications *contracts*. Contracts include norms, the atomic satisfied ($\top$), and the transgressed ($\bot$) contract. Contracts can be *conjuncts* ($\wedge$) and *sequentially composed* (;). A contract may repair the violation of another ($C \blacktriangleright C'$ means that C' is the *reparation* applied when C is violated).

Contracts can be triggered when a regular expression matches tightly ($\langle re \rangle C$). A regular expression can also guard $\text{!`}$ a contract $C$, such that an unrecoverable mismatch with it removes the need to continue complying with $C$ in ($re \,\text{!`}\, C$).

We allow *recursive* definitions of contracts ($rec\ X.C$), where $X \in \mathbb{X}$, with some restrictions. First, we do not allow a contract to have two recursive subcontracts using the same variable name. Secondly, we have some syntactic restrictions on the contract C appearing inside of the recursion: C's top-level operator is always a sequence, or a regular expression trigger contract, with X only appearing once and on the right-hand side of a sequence, i.e., the expression must be tail recursive. We also require an additional restriction for

recursion with the reparation operator: the reparation has to either not be the last operation before X or the whole recursion should be guarded with $re\,!\!\flat$, the reason behind it is to avoid the procrastination dilemma [14]. For example, $rec\ X.\langle re\rangle((C \blacktriangleright C'); X)$ and $re\,!\!\flat(rec\ X.C \blacktriangleright X)$ are valid, unlike $rec\ X.X$, $rec\ X.C; (C' \wedge X)$, $rec\ X.\langle re\rangle((C; X); C')$, and $rec\ X.C \blacktriangleright X$. Moreover, a recursion variable $X \in \mathbb{X}$ must always be bound when it appears in a contract.

In our setting, we want to be able to talk about collaborative actions (actions that require both parties to be achieved successfully) and non-interference between the parties (a party not being allowed to interfere with the other party carrying out a certain action). We model both of these using a notion of synchronicity. We will later represent parties as Moore machines; here we talk just about their traces.

We assume two traces over 2^{Σ}, one for each party: w_0 and w_1. A party's trace is a record of which actions were enabled (or attempted) by that party. The step-wise intersection of these traces, $w_0 \sqcap w_1$, is the trace of *successful* actions. Restricting attention to the successful actions misses information about attempts that were not successful. Instead, we give semantics over pairs of party traces, an *interaction*, rather than over $w_0 \sqcap w_1$, allowing us to localise interference. This setting allows us to model both collaboration and non-interference between the parties in the same way. If the parties are required to collaborate on an action, then they must both propose it (*obligation*). If instead, the parties should ensure an action is not successful, then at least one of them must not enable it (*prohibition*). If a party is required to not interfere with another party's action, then they must also enable it (*permission*). We refer to actions of one party variously as *proposed*, *attempted*, or *enabled* by that party. We consider an example specification in our language.

Example 3.1. *Consider two possibly distinct robots, 0 and 1, working on a factory floor, with their main goal being to cooperate in placing incoming packages on shelves. Each robot has sensors to identify when a new package is in the queue (detectProd), and they must lift the package together (lift), and place it on a shelf (putOnShelf). Between iterations of this process, the robots are individually allowed to go to their charging ports (charge0 or charge1). If a robot does not help in lifting, it is given another chance:*

$$
\begin{aligned}
\mathit{permitCharge} &\stackrel{\mathsf{def}}{=} P_0(\mathit{charge0}) \wedge P_1(\mathit{charge1}) \\
\mathit{lift}(p) &\stackrel{\mathsf{def}}{=} O_p(\mathit{lift}) \blacktriangleright O_p(\mathit{lift}) \\
\mathit{detect\&Lift}(p) &\stackrel{\mathsf{def}}{=} \langle \mathit{detectProd}_p \rangle \mathit{lift}(p) \\
\mathit{detect\&Place} &\stackrel{\mathsf{def}}{=} (\mathit{detect\&Lift}(0) \wedge \mathit{detect\&Lift}(1))\,; \\
&\qquad (O_0(\mathit{putOnShelf}) \wedge O_1(\mathit{putOnShelf})) \\
\mathit{collabRobot} &\stackrel{\mathsf{def}}{=} rec\ X.\mathit{permitCharge}; \mathit{detect\&Place}; X.
\end{aligned}
$$

$$
\begin{array}{lcl}
\boldsymbol{w}_i^j \models_s \top & \stackrel{\text{def}}{=} & i = j \\
\boldsymbol{w}_i^j \models_s \bot & \stackrel{\text{def}}{=} & \mathit{false} \\
\boldsymbol{w}_i^j \models_s O_p(a) & \stackrel{\text{def}}{=} & i = j \wedge a \in w_0[i] \text{ and } a \in w_1[i] \\
\boldsymbol{w}_i^j \models_s F_p(a) & \stackrel{\text{def}}{=} & i = j \wedge a \notin w_p[i] \text{ or } a \notin w_{1-p}[i] \\
\boldsymbol{w}_i^j \models_s P_p(a) & \stackrel{\text{def}}{=} & i = j \wedge a \in w_p[i] \text{ implies } a \in w_{1-p}[i]) \\
\boldsymbol{w}_i^j \models_v N & \stackrel{\text{def}}{=} & i = j \wedge \boldsymbol{w}_i^j \not\models_s N \\
 & & \\
\boldsymbol{w}_i^j \models_s \langle re \rangle C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \in \overline{cl}(re) \text{ or } (\exists k < j : \boldsymbol{w}_i^k \in TL(re) \text{ and } \boldsymbol{w}_{k+1}^j \models_s C) \\
\boldsymbol{w}_i^j \models_v \langle re \rangle C & \stackrel{\text{def}}{=} & \exists k < j : \boldsymbol{w}_i^k \in TL(re) \text{ and } \boldsymbol{w}_{k+1}^j \models_v C \\
\boldsymbol{w}_i^j \models_s re \natural C & \stackrel{\text{def}}{=} & (\boldsymbol{w}_i^j \in \overline{cl}(re) \cup TL(re) \text{ and } \nexists k < j : (\boldsymbol{w}_i^k \models_v C)) \\
 & & \text{ or } (\boldsymbol{w}_i^j \in cl(re) \text{ and } \boldsymbol{w}_i^j \models_s C) \\
\boldsymbol{w}_i^j \models_v re \natural C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \in cl(re) \text{ and } \boldsymbol{w}_i^j \models_v C \\
\boldsymbol{w}_i^j \models_s C \wedge C' & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^k \models_s C \text{ and } \boldsymbol{w}_i^l \models_s C' \text{ and } j = max(k,l) \\
\boldsymbol{w}_i^j \models_v C \wedge C' & \stackrel{\text{def}}{=} & (\boldsymbol{w}_i^j \models_v C \text{ or } \boldsymbol{w}_i^j \models_v C') \text{ and } \nexists k < j : \boldsymbol{w}_i^k \models_v C \wedge C' \\
\boldsymbol{w}_i^j \models_s C; C' & \stackrel{\text{def}}{=} & \exists k < j : \boldsymbol{w}_i^k \models_s C \text{ and } \boldsymbol{w}_{k+1}^j \models_s C' \\
\boldsymbol{w}_i^j \models_v C; C' & \stackrel{\text{def}}{=} & (\exists k < j : \boldsymbol{w}_i^k \models_s C \text{ and } \boldsymbol{w}_{k+1}^j \models_v C') \text{ or } \boldsymbol{w}_i^j \models_v C \\
\boldsymbol{w}_i^j \models_s C \blacktriangleright C' & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \models_s C \text{ or } (\exists k < j : \boldsymbol{w}_i^k \models_v C \text{ and } \boldsymbol{w}_{k+1}^j \models_s C') \\
\boldsymbol{w}_i^j \models_v C \blacktriangleright C' & \stackrel{\text{def}}{=} & \exists k < j : \boldsymbol{w}_i^k \models_v C \text{ and } \boldsymbol{w}_{k+1}^j \models_v C' \\
\boldsymbol{w}_i^j \models_s rec\ X.C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \models_s C[X \backslash rec\ X.C] \\
\boldsymbol{w}_i^j \models_v rec\ X.C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \models_v C[X \backslash rec\ X.C] \\
 & & \\
\boldsymbol{w}_i^j \models_? C & \stackrel{\text{def}}{=} & \nexists k \leq j : \boldsymbol{w}_i^k \models_s C \text{ or } \boldsymbol{w}_i^k \models_v C
\end{array}
$$

Fig. 1: Informative semantics rules over a finite interaction $\boldsymbol{w}_i^j$.

3.1 Informative Semantics

The semantics of our language is defined on an *interaction*, i.e. a pair of traces w_0 and w_1, restricting our view to a slice with a minimal position i and maximal one j. For the remainder of this paper, we will refer to this interaction with $\boldsymbol{w}_i^j$.

In Figure 1, we introduce the semantic relations for *informative* satisfaction ($\models_s$) and violation ($\models_v$). These capture the moment of satisfaction and violation of a contract in a finite interaction. We use this to later define when an infinite interaction models a contract. In Figure 1 we also capture with $\models_?$, when the interaction slice neither informatively satisfies nor violates the contract.

We give some intuition and mention interesting features of the semantics. Note how we only allow the status of atomic contracts to be informatively decided in one time-step (when $i = j$), given they only talk about one action. When it comes to the trigger contract, our goal is to confirm its fulfillment only when we no longer closely align with the specified trigger language. Alternatively, we consider it satisfied if we've matched it previously and subsequently maintained

compliance with the contract. Conversely, we would classify a violation if we achieved a close match but then deviated from the contract's terms. Regarding the regular expression guard, we have two scenarios for evaluating satisfaction. First, we ensure satisfaction when either we have precisely matched the language or have taken actions preventing any future matching of the guard, with no prior violations or the guarded contract. Second, we verify satisfaction when there's still a possibility of a precise match of the guard, and the guarded contract has already been satisfied. In contrast, a violation occurs when there remains a chance for a precise match in the future of the guard, and a violation of the sub-contract occurs.

The definitions for conjunction and sequence are relatively simple. Note that for conjunction we take the maximum index at which both contracts have been satisfied. Sequence and reparation are similar, except in reparation we only continue in the second contract if the first is violated, while we violate it if both contracts end up being violated. For recursion, we simply re-write variable X as needed to determine satisfaction or violation.

Example 3.2. *Note how the semantics ensure that, given traces* w_0 *and* w_1 *such that* $w_0[0] = w_1[1] = \{charge0, charge1\}$ *then* $\boldsymbol{w}_0^0 \models_s permitCharge$*, i.e. both robots try to charge and allow each other to charge. But if further* $w_0[1..3] = \langle\{detectProd\}, \{lift\}, \{lift\}\rangle$ *and* $w_1[1..3] = \langle\{\}, \{\}.\{\}\rangle$*, then* $\boldsymbol{w}_0^3 \models_v CollabRobot$*, since robot 0 attempted a lift but robot 1 declined helping in lifting.*

Then, we show that if a contract is informatively satisfied (violated), then any suffix or prefix of the interaction cannot also be informatively satisfied (violated):

Lemma 3.1 Unique satisfaction and violation. *If there exists* j *and* k *such that* $\boldsymbol{w}_i^j \models_s C$ *and* $\boldsymbol{w}_i^k \models_s C$*, then* $j = k$*. Similarly, if there exists* j *and* k *such that* $\boldsymbol{w}_i^j \models_v C$ *and* $\boldsymbol{w}_i^k \models_v C$ *then* $j = k$*.*

Proof sketch. For the atomic contracts, this is clear. By structural induction, the result follows for conjunction, sequence, and reparation. For the trigger operations, the definition of TL ensures the result. For recursion, note how given a finite interaction there is always a finite amount of times the recursion can be unfolded (with an upper bound of $j - i$) so that we can determine satisfaction or violation in finite time.

If an interaction is not informative for satisfaction, it is not necessarily informative for violation, and vice-versa. But we can show that if there is a point of informative satisfaction then there is no point of informative violation.

Lemma 3.2 Disjoint satisfaction and violation. *Informative satisfaction and violation are disjoint: there are no* j, k *s.t.* $\boldsymbol{w}_i^j \models_s C$ *and* $\boldsymbol{w}_i^k \models_v C$*.*

Proof sketch. The proof follows easily by induction on the structure of C.

We can then give semantics to infinite interactions.

$$
\begin{array}{lcl}
\boldsymbol{w}_i^j \models_v^p \top & \stackrel{\text{def}}{=} & \mathit{false} \\
\boldsymbol{w}_i^j \models_v^p \bot & \stackrel{\text{def}}{=} & \mathit{false} \\
\boldsymbol{w}_i^j \models_v^p O_{1-p}(a) & \stackrel{\text{def}}{=} & i = j \wedge a \in w_{1-p}[i] \text{ and } a \notin w_p[i] \\
\boldsymbol{w}_i^j \models_v^p O_p(a) & \stackrel{\text{def}}{=} & i = j \wedge a \notin w_p[i] \\
\boldsymbol{w}_i^j \models_v^p F_{1-p}(a) & \stackrel{\text{def}}{=} & \mathit{false} \\
\boldsymbol{w}_i^j \models_v^p F_p(a) & \stackrel{\text{def}}{=} & i = j \wedge a \in w_p[i] \text{ and } a \in w_{1-p}[i] \\
\boldsymbol{w}_i^j \models_v^p P_{1-p}(a) & \stackrel{\text{def}}{=} & i = j \wedge a \in w_{1-p}[i] \text{ and } a \notin w_p[i] \\
\boldsymbol{w}_i^j \models_v^p P_p(a) & \stackrel{\text{def}}{=} & \mathit{false} \\
\boldsymbol{w}_i^j \models_v^p \langle re \rangle C & \stackrel{\text{def}}{=} & \exists k < j : \boldsymbol{w}_i^k \models_s TL(re) \text{ and } \boldsymbol{w}_{k+1}^j \models_v^p C \\
\boldsymbol{w}_i^j \models_v^p re \triangleright C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \in cl(re) \text{ and } \boldsymbol{w}_i^j \models_v^p C \\
\boldsymbol{w}_i^j \models_v^p C \wedge C' & \stackrel{\text{def}}{=} & (\boldsymbol{w}_i^j \models_v^p C \text{ or } \boldsymbol{w}_i^j \models_v^p C') \text{ and} \\
 & & (\nexists k < j : \boldsymbol{w}_i^k \models_v^{1-p} C \wedge C') \text{ and} \\
 & & \neg(\mathit{conflict}(C, C', \boldsymbol{w}_i^{j-1})) \\
\boldsymbol{w}_i^j \models_v^p C \blacktriangleright C' & \stackrel{\text{def}}{=} & \exists k : \boldsymbol{w}_i^k \models_v C \text{ and } \boldsymbol{w}_{k+1}^j \models_v^p C' \\
\boldsymbol{w}_i^j \models_v^p rec\ X.C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \models_v^p C[X \backslash rec\ X.C]
\end{array}
$$

Fig. 2: Blame semantics rules over a finite interaction $\boldsymbol{w}_i^j$

Definition 3.2 Models. *For an infinite interaction $\boldsymbol{w}_0^\infty$, and a cDL contract C, we say $\boldsymbol{w}_0^\infty$ models a contract C, denoted by $\boldsymbol{w}_0^\infty \vDash C$, when there is no prefix of the interaction that informatively violates C: $\boldsymbol{w}_0^\infty \vDash C \stackrel{\text{def}}{=} \nexists k \in \mathbb{N} \cdot \boldsymbol{w}_0^k \models_v C$.*

3.2 Blame Assignment

We are not interested only in whether a contract is satisfied or violated, but also on *causation* and *responsibility* [9,12,10]. Here we give a relation that identifies when a party is responsible for a violation at a certain point in an interaction. Blame assignment could be specified following multiple criteria, we assign blame when an agent neglects to perform an action it is obliged to do or that another agent is obliged to do (passive blame), or for attempting to do an action it is forbidden from doing (active blame). The blame is forward looking where we identify the earliest cause of violation. Furthermore, we are only interested in causation and not on more advanced features such as "moral responsibility" or "intentionality". The blame semantics is only defined as a violation by party p relation as in $\models_v^p$. This semantics is defined in Figure 2.

For blame assignment, the labeling of norms with parties is crucial. Here we give meaning to these labels in terms of who is the main subject of the norm in question. For example, consider that $O_0(a)$ can be violated in three ways: either (i) both parties do not attempt a, (ii) party 0 does not attempt a but party 1 does, or (iii) party 0 attempts a but party 1 does not. Our interpretation is that since party 0 is the main subject of the obligation, party 0 is blamed when it does not attempt a (cases (i) and (ii)), but party 1 is blamed when it does

not attempt a (case (iii)). The intuition is that by not attempting a, party 0 violated the contract, thus relieving party 1 of any obligation to cooperate or non-interfere (given party 0 knows there is no hope for the norm to be satisfied if they do not attempt a). We use similar interpretations for the other norms.

Another crucial observation is that violations of a contract are not necessarily caused by a party. For example, the violated contract $\bot$ cannot be satisfied. Moreover, norms can conflict, e.g., $O_p(a) \wedge F_p(a)$. Conflicts are not immediately obvious without some analysis, e.g., $\langle re \rangle O_p(a) \wedge \langle re' \rangle F_p(a)$ (where there is some interaction for which re and re' tightly match at the same time). We provide machinery to talk about conflicts, to avoid unsound blaming, by characterising two contracts to be conflicting when there is no way to satisfy them together.

Definition 3.3 Conflicts. *Two contracts C and C' are in conflict after a finite interaction $\boldsymbol{w}_i^j$ if at that point their conjunction has not been informatively satisfied or violated yet, but all possible further steps lead to its violation:* $\mathit{conflict}(C, C', \boldsymbol{w}_i^j) \stackrel{\text{def}}{=} \nexists \boldsymbol{w}' : \boldsymbol{w'}_i^j = \boldsymbol{w}_i^j \wedge \boldsymbol{w'}_i^{j+1} \not\models_v C \wedge C'$.

Another instance of a conflict can be observed between $C_1 = O_0(a); F_1(c)$ and $C_2 = O_0(b) \blacktriangleright O_0(c)$ at the second position. This can be demonstrated with a trace of length one, $\langle a_0; a_1 \rangle$, where the obligation to achieve c for party 0 and the prohibition to achieve c for party 1 have to be enforced simultaneously.

Example 3.3. *Recall the violating example in Example. 3.2, where robot 1 declines in helping lifing, twice. Clearly in that case $\boldsymbol{w}_0^3 \models_v^1$ collabRobot . However, if robot 0 did not attempt a lift in position 3 (i.e., to attempt to satisfy the reparation), the blame would be on the other agent.*

From the definition of blame it easily follows that a party is blamed for a violation only when there is a violation:

Proposition 3.1. *If a party p is blamed for the violation of C then C has been violated: $\exists p \cdot \boldsymbol{w}_i^j \models_v^p C$ implies $\boldsymbol{w}_i^j \models_v C$.*

Proof. Note how each case of $\models_v^p$ implies its counterpart in $\models_v$.

But the opposite is not true:

Proposition 3.2. *A contract may be violated but both parties be blameless: $\boldsymbol{w}_i^j \models_v C$ does not imply $\exists p \cdot \boldsymbol{w}_i^j \models_v^p C$.*

Proof. Consider their definitions on $\bot$, and given conjunction and the presence of conflicts.

Proposition 3.3 Satisfaction implies no blame. *Satisfaction of contract C means that no party will get blamed: $\boldsymbol{w}_i^j \models_s C$ implies $\nexists p \cdot \boldsymbol{w}_i^j \models_v^p C$*

Proof. Assume the contrary, i.e. that C is satisfied but party p is blamed. By Proposition 3.1 then there is a violation, but Lemma 3.2 implies we cannot both have a satisfaction or violation.

Observation 3.1. *For any contract $C^{\not\perp}$ defined on cDL free of $\perp$ and free of conflicts, the violation of a contract $C^{\not\perp}$ leads to blame.*

Observation 3.2 Double blame. *Double blame in cDL for both parties p and $1-p$ is possible. Consider $C = O_p(a) \wedge O_p(b)$. Violation of the left-hand side by p and the violation of the right-hand side by $1-p$ can happen at the same time.*

3.3 Quantitative Semantics

While it is possible to assign blame to one party for violating a contract, other qualitative metrics can provide additional information about the violation. These metrics can determine the number of violations caused by each party, as well as the level of satisfaction with the contract. To assess responsibility for contract violations, we introduce the notion of a mistake score, ρ, for each party, enabling us to calculate a *responsibility degree*. It is important to note that our language permits reparations, whereby violations can be corrected in the next time step. However, interactions that are satisfied with reparations are not considered ideal. We present quantitative semantics to compare satisfying interactions based on the number of repaired violations a party incurs. We define relations that track the number of repaired violations attributed to each party with a mistake score, ρ, written $\models^p_s$ for informative satisfaction and $\models^p_v$ for informative violation of the contract. We can also keep track of the number of violations when the trace is not informative through $\models^p_?$. Figure 3 provides a definition of this semantics. Note this definition intersects the previous semantic definitions, and due to space constraints, we do not re-expand that further. The addition is that we are disambiguating some cases to identify when to add to the score to identify a violation caused by p. For example, see the definition of $\models^p_v$ for a norm.

Example 3.4. *Consider again Example 3.1, and consider the finite interaction $(\langle\{charge0\},\{detectProd\},\{\},\{lift\}\rangle$ and $\langle\{charge0\},\{\},\{lift\},\{lift\}\rangle$. Note how this will lead to robot 0 being given a score of one since on the third step there is a violation that is repaired subsequently.*

Lemma 3.3 Soundness and completeness. *The quantitative semantics is sound and complete with regard to the informative semantics: $\boldsymbol{w}_i^j \models_\gamma C \iff \exists \rho_1, \rho_2 : \boldsymbol{w}_i^j, \rho_1 \models^p_v C$ and $\boldsymbol{w}_i^j, \rho_2 \models^{1-p}_v$ with $\gamma \in \{s, v, ?\}$.*

Proof sketch. By induction on the quantitative semantics and informative semantics.

Lemma 3.4 Fairness of the Quantitative semantics. *The quantitative semantic is fair, meaning that if the score of a player p is ρ then p is to be blamed for non-fulfilling ρ norms of the contract: $\boldsymbol{w}_i^j, \rho \models^p_\gamma C \implies \exists N_1 \dots N_\rho \in$ subcontracts$(C) : \boldsymbol{w}_i^j \models^p_\gamma N_i$ with $\gamma \in \{s, v, ?\}$, where subcontracts(C) is a multiset containing the subcontracts of C, up to how often they appear.*

Proof sketch. We prove, this by structural induction, noting that the score only increases when p is blamed for the violation of a norm, while the inductive case easily follows from the inductive hypothesis.

$$
\begin{array}{lcl}
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_s^p N & \stackrel{\text{def}}{=} & \rho = 0 \wedge \boldsymbol{w}_i^j \models_s N \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_v^p N & \stackrel{\text{def}}{=} & \begin{cases} \rho = 1 & \text{if } \boldsymbol{w}_i^i \models_v^p N \\ \rho = 0 & \text{if } N = \bot \vee \boldsymbol{w}_i^i \models_v^{1-p} N \end{cases} \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p N & \stackrel{\text{def}}{=} & \mathit{false} \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_s^p \langle re \rangle C & \stackrel{\text{def}}{=} & \begin{cases} \rho = 0 & \text{if } \boldsymbol{w}_i^j \in \overline{cl}(re) \\ \boldsymbol{w}_{k+1}^j, \rho \models\!\!\!\models_s^p C & \text{if } \exists k : \boldsymbol{w}_i^k \in TL(re) \end{cases} \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_v^p \langle re \rangle C & \stackrel{\text{def}}{=} & \exists k : \boldsymbol{w}_i^k \in TL(re) \text{ and } \boldsymbol{w}_{k+1}^j, \rho \models\!\!\!\models_v^p C \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p \langle re \rangle C & \stackrel{\text{def}}{=} & \begin{cases} \rho = 0 & \text{if } \quad \nexists k \leq j : \boldsymbol{w}_i^k \in TL(re) \\ \boldsymbol{w}_{k+1}^j, \rho \models\!\!\!\models_?^p C & \text{else } \exists k \leq j : \boldsymbol{w}_i^k \in TL(re) \end{cases} \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_s^p re \rangle\!\!\rangle C & \stackrel{\text{def}}{=} & \begin{cases} \boldsymbol{w}_i^{j-1}, \rho \models\!\!\!\models_?^p C \text{ if } \boldsymbol{w}_i^j \in \overline{cl}(re) \cup TL(re) \\ \qquad\qquad \textit{and } \nexists \rho', k < j : \boldsymbol{w}_i^k, \rho' \models\!\!\!\models_v^p C \\ \boldsymbol{w}_i^j, \rho \models\!\!\!\models_s^p C \quad \text{if } \boldsymbol{w}_i^j \in cl(re) \end{cases} \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_v^p re \rangle\!\!\rangle C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \in cl(re) \text{ and } \boldsymbol{w}_i^j, \rho \models\!\!\!\models_v^p C \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p re \rangle\!\!\rangle C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j \in cl(re) \text{ and } \boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p C \\
\boldsymbol{w}_i^j, (\rho_1 + \rho_2) \models\!\!\!\models_s^p C \wedge C' & \stackrel{\text{def}}{=} & \exists k, l : \boldsymbol{w}_i^k, \rho_1 \models\!\!\!\models_s^p C \text{ and } \boldsymbol{w}_i^l, \rho_2 \models\!\!\!\models_s^p C' \\
 & & \text{and } j = max(k, l) \\
\boldsymbol{w}_i^j, (\rho_1 + \rho_2) \models\!\!\!\models_v^p C \wedge C' & \stackrel{\text{def}}{=} & ((\boldsymbol{w}_i^j, \rho_2 \models\!\!\!\models_v^p C' \text{ or } \boldsymbol{w}_i^j, \rho_2 \models\!\!\!\models_s^p C') \\
 & & \quad \text{and } \boldsymbol{w}_i^j, \rho_1 \models\!\!\!\models_v^p C) \\
 & & \text{or } ((\boldsymbol{w}_i^j, \rho_1 \models\!\!\!\models_v^p C' \text{ or } \boldsymbol{w}_i^j, \rho_1 \models\!\!\!\models_s^p C') \\
 & & \quad \text{and } \boldsymbol{w}_i^j, \rho_2 \models\!\!\!\models_v^p C') \\
 & & \text{or } (\boldsymbol{w}_i^j, \rho_1 \models\!\!\!\models_v^p C \text{ and } \boldsymbol{w}_i^j, \rho_2 \models\!\!\!\models_v^p C' \\
 & & \quad \text{and } \neg \mathit{conflict}(C, C', \boldsymbol{w}_i^{j-1})) \\
 & & \text{or } (\boldsymbol{w}_i^{j-1}, \rho_1 \models\!\!\!\models_?^p C \text{ and } \boldsymbol{w}_i^{j-1}, \rho_2 \models\!\!\!\models_?^p C' \\
 & & \quad \text{and } \mathit{conflict}(C, C', \boldsymbol{w}_i^{j-1})) \\
\boldsymbol{w}_i^j, (\rho_1 + \rho_2) \models\!\!\!\models_?^p C \wedge C' & \stackrel{\text{def}}{=} & (\boldsymbol{w}_i^j, \rho_1 \models\!\!\!\models_s^p C \text{ and } \boldsymbol{w}_i^j, \rho_2 \models\!\!\!\models_?^p C') \\
 & & \text{or } (\boldsymbol{w}_i^j, \rho_1 \models\!\!\!\models_?^p C \text{ and } \boldsymbol{w}_i^j, \rho_2 \models\!\!\!\models_s^p C') \\
\boldsymbol{w}_i^j, (\rho_1 + \rho_2) \models\!\!\!\models_s^p C; C' & \stackrel{\text{def}}{=} & \exists k < j : \boldsymbol{w}_i^k, \rho_1 \models\!\!\!\models_s^p C \text{ and } \boldsymbol{w}_{k+1}^j, \rho_2 \models\!\!\!\models_s^p C' \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_v^p C; C' & \stackrel{\text{def}}{=} & \begin{cases} \rho = \rho_1 + \rho_2 \text{ if } \quad \exists k < j : \boldsymbol{w}_i^k, \rho_1 \models\!\!\!\models_s^p C \\ \qquad\qquad\qquad\quad \text{and } \boldsymbol{w}_{k+1}^j, \rho_2 \models\!\!\!\models_v^p C' \\ \boldsymbol{w}_i^j, \rho \models\!\!\!\models_v^p \quad \text{else} \end{cases} \\
\boldsymbol{w}_i^j, (\rho_1 + \rho_2) \models\!\!\!\models_?^p C \ ; \ C' & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p C \\
 & & \quad \text{or } (\boldsymbol{w}_i^k, \rho_1 \models\!\!\!\models_s^p C \text{ and } \boldsymbol{w}_{k+1}^j, \rho_2 \models\!\!\!\models_?^p C') \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_s^p C \blacktriangleright C' & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j, \rho \models\!\!\!\models_s^p C \\
 & & \text{or } (\exists k < j : \boldsymbol{w}_i^k, \rho_1 \models\!\!\!\models_v^p C \\
 & & \quad \text{and } \boldsymbol{w}_{k+1}^j, \rho_2 \models\!\!\!\models_s^p C \wedge \rho = \rho_1 + \rho_2) \\
\boldsymbol{w}_i^j, (\rho_1 + \rho_2) \models\!\!\!\models_v^p C \blacktriangleright C' & \stackrel{\text{def}}{=} & (\exists k < j : \boldsymbol{w}_i^k, \rho_1 \models_v^p C \text{ and } \boldsymbol{w}_{k+1}^j, \rho_2 \models\!\!\!\models_v^p C') \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p C \blacktriangleright C' & \stackrel{\text{def}}{=} & \begin{cases} \rho = \rho_1 + \rho_2 \text{ if } \quad \boldsymbol{w}_i^k, \rho_1 \models\!\!\!\models_v^p C \\ \qquad\qquad\qquad\quad \text{and } \boldsymbol{w}_{k+1}^j, \rho_2 \models\!\!\!\models_?^p C' \\ \boldsymbol{w}_i^j, \rho \models\!\!\!\models_?^p C \text{ else} \end{cases} \\
\boldsymbol{w}_i^j, \rho \models\!\!\!\models_\gamma^p rec\ X.C & \stackrel{\text{def}}{=} & \boldsymbol{w}_i^j, \rho \models\!\!\!\models_\gamma^p C[X \setminus rec\ X.C] \qquad \text{for } \gamma \in \{s, v, ?\}
\end{array}
$$

Fig. 3: Quantitative semantics rules over a finite interaction $\boldsymbol{w}_i^j$.

4 Analysis

In this section, we define an automata-theoretic approach to analyzing $c\mathcal{DL}$ contracts, through a construction to a safety automaton. We use this for model checking and blame analysis, but leave the application for quantitative analysis for future work.

4.1 Contracts to Automata

We give a construction from $c\mathcal{DL}$ contracts to automata that recognize interactions that are informative for satisfaction or violation. For brevity, we keep the definition of the automata symbolic, with transitions tagged by propositions over party actions, representing a set of concrete transitions. The automaton is over the alphabet $\Sigma_{0,1}$ since it requires information about the parties.

Definition 4.1. *The deterministic* automaton of contract C *is:*

$$aut(C) \stackrel{\text{def}}{=} \langle \Sigma_{0,1}, S, s_0, \{s_B\}, \rightarrow \rangle.$$

We define $\rightarrow$ *through the below function* $\tau(C, s_0, s_G, s_B, \{\})$ *that computes a set of transitions, given a contract, an initial state* (s_0)*, a state denoting satisfaction* (s_G)*, a state denoting violation* (s_B)*, and a partial function* V *from recursion variables* ($\mathbb{X}$) *to states, characterised by (with* s *as a fresh state):*

$$\begin{aligned}
\tau(\top, s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \{s_0 \xrightarrow{true} s_G\} \\
\tau(\bot, s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \{s_0 \xrightarrow{true} s_B\} \\
\tau(O_p(a), s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \{s_0 \xrightarrow{a_p \wedge a_{1-p}} s_G, s_0 \xrightarrow{\neg(a_p \wedge a_{1-p})} s_B\} \\
\tau(F_p(a), s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \{s_0 \xrightarrow{\neg(a_p \wedge a_{1-p})} s_G, s_0 \xrightarrow{a_p \wedge a_{1-p}} s_B\} \\
\tau(P_p(a), s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \{s_0 \xrightarrow{a_p \implies a_{1-p}} s_G, s_0 \xrightarrow{\neg(a_p \implies a_{1-p})} s_B\} \\
\tau(\langle re \rangle C, s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} A(re, s_0, s, s_G) \cup \tau(C, s, s_G, s_B, V) \\
\tau(re \,\natural\, C, s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} (A(re, s_0, s_G, s_G) \| \tau(C, s_0, s_G, s_B, V)) \\
&\qquad [(s_G, *)/s_G][(*, s_B)/s_B][(*, s_G)/s_G] \\
\tau(C \wedge C', s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} (\tau(C, s_0, s_G, s_B, V) \|^r \tau(C', s_0, s_G, s_B, V)) \\
&\qquad [(s_G, s_G)/s_G][(s_B, *)/s_B][(*, s_B)/s_B] \\
\tau(C; C', s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \tau(C, s_0, s, s_B, V) \cup \tau(C', s, s_G, s_B, V) \\
\tau(C \blacktriangleright C', s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \tau(C, s_0, s_G, s, V) \cup \tau(C', s, s_G, s_B, V) \\
\tau(X, s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \{s_G \xrightarrow{\epsilon} V(X)\} \\
\tau(recX.C, s_0, s_G, s_B, V) &\stackrel{\text{def}}{=} \tau(C, s_0, s_G, s_B, V[X \mapsto s_0])
\end{aligned}$$

We define $\rightarrow'$ *as* $\tau(C, s_0, s_G, s_B, \{\})$ *without all transitions outgoing from* s_G *and* S_B*, and define* $\rightarrow \stackrel{\text{def}}{=} \rightarrow' \cup \{s_B \xrightarrow{true} s_B\} \cup \{s_G \xrightarrow{true} s_G\}$*, where* S *is the set of states used in* $\rightarrow$*. We assume the* ϵ*-transitions are removed using standard methods.*

We give some intuition for the construction. The transitions for the atomic contracts follow quite clearly from their semantics. For the trigger contracts, we use a fresh state s to connect the automaton for the regular expression, with that of the contract, ensuring the latter is only entered when the former tightly matches. For the guard contract, we instead synchronously compose ($\|$) both automata (i.e., intersect their languages), getting a set of transitions. Here we also relabel tuples of states to single states. Recall we use $(*, s)$ to match any pair, where the second term is s, and similarly for $(s, *)$. Through the sequence of re-labellings, we ensure: first that reaching s_G in the acceptance of the first means; (2) reaching s_B in the second means violation; and (3) if the previous two situations are not the case, reaching s_G in the second means acceptance.

For conjunction, instead of using the synchronous product, we use the relaxed variant ($\|^r$), since the contracts may require traces of different lengths for satisfaction. This relaxed product allows the 'longer' contract to continue after the status of the other is determined. For sequence, we use the fresh state s to move between the automata, once the first contract has been satisfied. For reparation this is similar, except we move between the contracts at the moment the first is violated. For recursion, we simply loop back to the initial state of the recursed contract with an ϵ-transition once the corresponding recursion variable is encountered.

Note how analyzing states without viable transitions, after applying τ, can be used for *conflict analysis* of $c\mathcal{DL}$ contracts. For example, when there is a conflict, e.g., $O_p(a) \wedge F_p(a)$, there will be a state with all outgoing transitions to s_B.

Theorem 4.1 Correctness. *An infinite interaction is a model of* C*, iff it never reaches a rejecting state in* $aut(C)$*:*
$\forall \boldsymbol{w}_0^\infty \cdot \boldsymbol{w}_0^\infty \models C \iff w_0 \sqcup_1^0 w_1 \in L(aut(C))$.

Proof sketch. For the atomic contracts, the correspondence should be clear. By structural induction on the rest: triggering, sequence, and reparation should also be clear from the definition. For conjunction, the relaxed synchronous product makes sure the contract not yet satisfied continues being executed, as required, while the replacements ensure large nestings of conjunctions do not lead to large tuples of accepting or rejecting states. For $\natural$, using the synchronous product ensures the path ends when either is satisfied/violated, as required.

Corrolary 4.1. *An infinite interaction is not a model of* C*, iff it reaches a rejecting state in* $aut(C)$*:* $\forall (w_0, w_1) \not\models C \iff \exists j \in \mathbb{N} \cdot s_0 \xrightarrow{(w_0 \sqcup_1^0 w_1)[0 \ldots j]} s_B$.

Proof sketch. Follows from Theorem. 4.1 and completeness (up to rejection) of $aut(C)$.

Complexity From the translation note that without regular expressions the number of states and transitions is linear in the number of sub-clauses and operators in the contract, but is exponential in the presence of regular expressions.[5]

4.2 Model Checking

We represent the behaviour of each party as a Moore machine (M_0, and M_1). For party 0, the input alphabet is Σ_1 and the output alphabet is Σ_0, and vice-versa for party 1. We characterise their composed behaviour by using the product of the two dual Moore machines: $M_0 \otimes M_1$, getting an automaton over $\Sigma_0 \cup \Sigma_1$.

We can then compose this automaton that represents the interactive behaviour of the parties with the contract's automaton, $(M_0 \otimes M_1) \| aut(C)$. Then, if no rejecting state is reachable in this automaton, the composed party's behaviour respects the contract.

Theorem 4.2 Model Checking Soundness and Completeness. $\emptyset = RL((M_0 \otimes M_1) \| aut(C))$ *iff* $\nexists \boldsymbol{w}_0^\infty : w_0 \sqcup_1^0 w_1 \in L(M_0 \otimes M_1) \wedge \boldsymbol{w}_0^\infty \models_v C$.

Proof. Consider that $\|$ computes the intersection of the languages, while Theorem. 4.1 states that $L(aut(C))$ contains exactly the traces satisfying C (modulo a simple technical procedure to move between labelled traces and pairs of traces). Then it follows easily that $RL((M_0 \otimes M_1) \| aut(C))$ is empty only when there is no trace in $(M_0 \otimes M_1)$ that leads to a rejecting state in $aut(C)$. The same logic can be taken in the other direction. □

4.3 Blame Assignment

For the blame assignment, we can modify the automaton construction by adding two other violating states: s_B^0 and s_B^1, and adjust the transitions for the basic norms accordingly.

Definition 4.2. *The deterministic* blame automaton of contract C *is:*

$$blAut(C) \stackrel{\text{def}}{=} \langle \Sigma_{0,1}, S, s_0, \{s_B, s_B^0, s_B^1, (s_B^0, s_B^1)\}, \rightarrow \rangle$$

We define $\rightarrow$ *through the function* $\tau(C, s_0, s_G, s_B^0, s_B^1, V)$ *that computes a set of transitions, as in Definition 4.1 but now assigning blame by transitioning to the appropriate state. We focus on a subset of the rules, given limited space, where there are substantial changes*[6]*:*

$$\tau(O_p(a), s_0, s_G, s_B^0, s_B^1, V) \stackrel{\text{def}}{=} \{s_0 \xrightarrow{a_p \wedge a_{1-p}} s_G, s_0 \xrightarrow{\neg a_p} s_B^p, s_0 \xrightarrow{a_p \wedge \neg a_{1-p}} s_B^{1-p}\}$$

[5] For example, a contract $recX.\top; (O_0(a) \wedge P_1(b)); X$ has size 8 (note normed actions are not counted).

[6] The missing rules essentially mirror the previous construction with the added states, and the different domains.

$$\tau(F_p(a), s_0, s_G, s_B^0, s_B^1, V) \stackrel{\text{def}}{=} \{s_0 \xrightarrow{\neg(a_p \wedge a_{1-p})} s_G, s_0 \xrightarrow{a_p \wedge a_{1-p}} s_B^p\}$$

$$\tau(P_p(a), s_0, s_G, s_B^0, s_B^1, V) \stackrel{\text{def}}{=} \{s_0 \xrightarrow{a_p \implies a_{1-p}} s_G, s_0 \xrightarrow{a_p \wedge \neg a_{1-p}} s_B^{1-p}\}$$

$$\tau(C \blacktriangleright C', s_0, s_G, s_B^0, s_B^1, V) \stackrel{\text{def}}{=} \tau(C, s_0, s_G, s^0, s^1, V) \cup \tau(C', s^0, s_G, s_B^0, V) \cup \tau(C', s^1, s_G, s_B^1, V)$$

Given $\rightarrow' = \tau(C, s_0, s_G, s_B, \{\})$, $\rightarrow$ is defined as $\rightarrow'$ with the following transformations, in order: (1) any tuple of states containing both s_B^0 and s_B^1 is relabelled as (s_B^0, s_B^1); (2) any tuple of states containing s_B^0 (s_B^1) is relabelled as s_B^0 (s_B^1); (3) any state for which all outgoing transitions go to a bad state are redirected to s_B; (4) any tuple of states containing s_G is relabelled as s_G; and (5) all bad states and s_G become sink states. S is the set of states used in $\rightarrow$. We assume the ϵ-transitions are removed using standard methods.

Note how this automata simply refines the bad states of the original automata construction, by assigning blame for the violation of norms through a transition to an appropriate new state. While the post-processing (see (3)), allows violations caused by conflicts to go instead to state s_B, where no party is blamed.

Then we prove correspondence with the blame semantics:

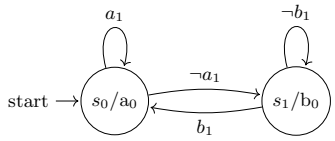
Theorem 4.3 Blame Analysis Soundness and Completeness. *Where RL_p, for $p \in \{0, 1\}$, is the rejecting language of the automaton through states that pass through s_B^p or the tuple state (s_B^0, s_B^1):*

$\emptyset = RL_p((M_0 \otimes M_1) \| blAut(C))$ iff $\nexists w_0, w_1 \in (2^\Sigma)^* : w_0 \sqcup_1^0 w_1 \in L(M_0 \otimes M_1) \wedge (w_0, w_1) \models_v^p C$.

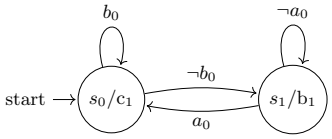
Proof. This follows from a slight modification of Corollary. 4.1 (since here we just refine the bad states of $aut(C)$) with the replacement of s_B by party-tagged bad states, and from a similar argument to Theorem. 4.2.

This automaton can be used for model checking as before, but it can also answer queries about who is to blame.

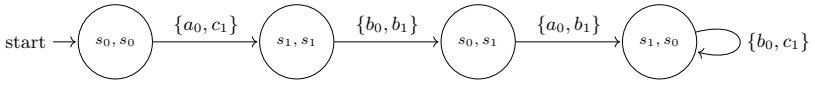
Example 4.1. *We illustrate in Figure 4 an example of two Moore machines representing the behaviour of two parties (Figures 4a and 4b). Note these are deterministic, therefore their composition (Figure 4) is just a trace. Note the same theory applies even when the Moore machines are non-deterministic. In Figures 4d and 4e we show the automaton and blame automaton for the contract $recX.(O_1(c) \blacktriangleright O_0(b); X)$. Our model checking procedure (without blame) will compose Figure 4 and Figure 4d, and identify that the trace reaches the bad state. Consider that the reparation consisting of an obligation to perform an action b was not satisfied. Similarly (not shown here) blame automaton would blame party 1 for the violation.*



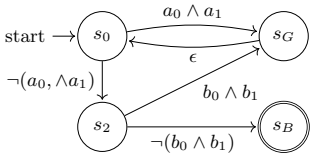
(a) M_0: Moore machine for agent 0.



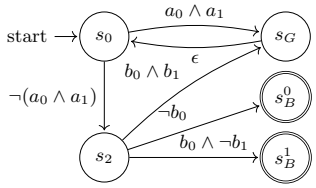
(b) M_1: Moore machine for agent 1.



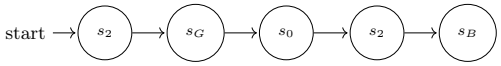
(c) The composition $M_0 \otimes M_1$.



(d) $C = aut(recX.(O_1(a) \blacktriangleright O_0(b)); X)$



(e) $blAut(recX.(O_1(a) \blacktriangleright O_0(b)); X)$



(f) $(M_0 \otimes M_1) \| C$

Fig. 4: Example of the model checking approach.

5 Related Work

Multi-agent systems. Several logics can express properties about multi-agent systems. For example, ATL can express the existence of a strategy for one or more agents to enforce a certain specification [2], while strategy logic makes strategies first-class objects [7]. Checking for the existence of strategies is in 2EXPTIME. Our logic is not concerned with the existence of strategies, but with analyzing the party strategies to ensure they respect a contract. So, our approach is more comparable to LTL than to game-based logic, limited to (co-)safety properties and with a notion of norms that allows us to talk about blame natively.

Concerning blame, [11] considers the notion of *blameworthiness*. They use structural equations to represent agents, but the approach is not temporal, and each agent performs only one action. Work in this area (e.g., [11,13,9]) tends to be in a different setting than ours.

They consider the cost of actions and agents' beliefs about the probability of their actions not achieving the expected outcome. Instead, we assume all the parties have knowledge of the contract, and we take an automata-theoretic

approach. Moreover, our blame derives from the norms, whereas other work depends on a notion of causality [8].

The work [1] extends *STIT logic* with notions of responsibility, allowing reasoning about blameworthiness and praiseworthiness. This, and other similar work (e.g., [15]) is more related to our work and even has a richer notion of blame. However, we give an automata-based model checking procedure.

Deontic logics Deontic logics have been used in a multi-agent setting before. For example, [6] define deontic notions in terms of ATL, allowing reasoning like *an obligation holds for an agent iff they have a strategy to carry it out.* These approaches (e.g., [6,17,20]) focus on obligations and neglect both reparations and our view of permissions as rights. Some approaches (e.g., [17,19]) however do perform model checking for a deontic logic in a multi-agent system setting. The work most similar to ours is that of *contract automata* [5], wherein a contract is represented as a Kripke structure (with states tagged by norms), two parties as automata, and permissions with a similar rights-based view. However, it takes a purely operational approach, and lacks a notion of blame.

Our language is an extension and combination of the deontic languages presented in [3,4,18], combining action attempts, a right-based view of permission, a two-party setting, and regular expressions as conditions.

Besides maintaining all these, we give denotational trace semantics, and provide blame and model checking algorithms.

6 Conclusions

In this paper we have introduced a deontic logic for reasoning about a two-party synchronous setting. This logic allows one to define constraints on when parties should support or non-interfere with the carrying out of a certain action or protocol. Using a pair of party traces, we can talk about attempts and success to perform collaborative actions. We consider automata constructions describing both the set of all satisfying and violating sequences. Given the behavior of the agents in the form of suitable automata, we have also provided algorithms for model checking and for blame assignment. To differentiate between satisfying a formula in the expected manner or by fullfilling the exceptional case, we introduce a quantitative semantics. This allows ordering satisfying traces depending on how often they use these exceptions.

This work may be extended in many directions. First, we could consider asynchronous interaction, distinguishing between sending and receiving. The syntax and semantics can also be extended easily to handle multi-party agents rather than just a two-party setting. Different quantitative semantics could be given, for example considering the *costs of actions* to reason when it is better to pay a fine rather than to behave as expected. We plan to study how to synthesise strategies for the different parties, for instance to ensure the optimal behaviour of agents.

References

1. Abarca, A.I.R., Broersen, J.M.: A stit logic of responsibility. In: Faliszewski, P., Mascardi, V., Pelachaud, C., Taylor, M.E. (eds.) 21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022. pp. 1717–1719. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS) (2022), https://www.ifaamas.org/Proceedings/aamas2022/pdfs/p1717.pdf
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM (JACM) **49**(5), 672–713 (2002)
3. Azzopardi, S., Gatt, A., Pace, G.J.: Reasoning about partial contracts. In: JURIX'16. Frontiers in Artificial Intelligence and Applications, vol. 294, pp. 23–32. IOS Press (2016). https://doi.org/10.3233/978-1-61499-726-9-23
4. Azzopardi, S., Pace, G.J., Schapachnik, F.: On observing contracts: Deontic contracts meet smart contracts. In: JURIX'18. Frontiers in Artificial Intelligence and Applications, vol. 313, pp. 21–30. IOS Press (2018). https://doi.org/10.3233/978-1-61499-935-5-21
5. Azzopardi, S., Pace, G.J., Schapachnik, F., Schneider, G.: Contract automata. Artif. Intell. Law **24**(3), 203–243 (sep 2016), https://doi.org/10.1007/s10506-016-9185-2
6. Broersen, J.: Strategic deontic temporal logic as a reduction to atl, with an application to chisholm's scenario. In: DEON'06. p. 53–68. Springer (2006). https://doi.org/10.1007/11786849_7
7. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007 – Concurrency Theory. pp. 59–73. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
8. Chockler, H.: Causality and responsibility for formal verification and beyond. In: CREST@ETAPS'16. EPTCS, vol. 224, pp. 1–8 (2016). https://doi.org/10.4204/EPTCS.224.1
9. Chockler, H., Halpern, J.Y.: Responsibility and blame: A structural-model approach. In: Gottlob, G., Walsh, T. (eds.) IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003. pp. 147–153. Morgan Kaufmann (2003), http://ijcai.org/Proceedings/03/Papers/021.pdf
10. Chockler, H., Halpern, J.Y.: Responsibility and blame: A structural-model approach. Journal of Artificial Intelligence Research **22**, 93–115 (2004)
11. Friedenberg, M., Halpern, J.Y.: Blameworthiness in multi-agent settings. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 525–532 (2019). https://doi.org/10.1609/aaai.v33i01.3301525
12. Halpern, J.Y.: Cause, responsibility and blame: a structural-model approach. Law, probability and risk **14**(2), 91–118 (2015)
13. Halpern, J.Y., Kleiman-Weiner, M.: Towards formal definitions of blameworthiness, intention, and moral responsibility. In: AAAI/IAAI/EAAI'18. AAAI Press (2018)
14. Jackson, F.: On the semantics and logic of obligation. Mind **94**(374), 177–195 (1985)
15. Lorini, E., Longin, D., Mayor, E.: A logical analysis of responsibility attribution: emotions, individuals and collectives. Journal of Logic and Computation **24**(6), 1313–1339 (12 2013), https://doi.org/10.1093/logcom/ext072
16. McNamara, P.: Deontic logic. In: Gabbay, D.M., Woods, J. (eds.) Logic and the Modalities in the Twentieth Century, Handbook of the History of Logic, vol. 7, pp. 197–288. Elsevier (2006), https://doi.org/10.1016/S1874-5857(06)80029-4

17. Penczek, W., Lomuscio, A.: Verifying epistemic properties of multi-agent systems via bounded model checking. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. p. 209–216. AAMAS '03, Association for Computing Machinery, New York, NY, USA (2003), https://doi.org/10.1145/860575.860609
18. Prisacariu, C., Schneider, G.: Cl: An action-based logic for reasoning about contracts. In: WoLLIC'09. LNCS, vol. 5514, pp. 335–349. Springer (2009). https://doi.org/10.1007/978-3-642-02261-6_27
19. Raimondi, F., Lomuscio, A.: Automatic verification of deontic properties of multi-agent systems. In: Lomuscio, A., Nute, D. (eds.) Deontic Logic in Computer Science. pp. 228–242. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
20. Shea-Blymyer, C., Abbas, H.: A deontic logic analysis of autonomous systems' safety. In: HSCC'20. pp. 26:1–26:11. ACM (2020), https://doi.org/10.1145/3365365.3382203
21. Von Wright, G.H.: Deontic logic. Mind **60**(237), 1–15 (1951)