Run-time Monitoring of Electronic Contracts*

Marcel Kyas, Cristian Prisacariu, and Gerardo Schneider**

Department of Informatics, University of Oslo,
P.O. Box 1080 Blindern, N-0316 Oslo, Norway.

Abstract. Electronic inter-organizational relationships are governed by contracts regulating their interaction. It is necessary to run-time monitor the contracts, as to guarantee their fulfillment as well as the enforcement of penalties in case of violations. The present work shows how to obtain a run-time monitor for contracts written in $\mathcal{CL}$, a formal specification language which allows to write conditional obligations, permissions and prohibitions over actions. We first give a trace semantics for $\mathcal{CL}$ which formalizes the notion of *a trace fulfills a contract*. We show how to obtain, for a given contract, an alternating Büchi automaton which accepts exactly the traces that fulfill the contract. This automaton is the basis for obtaining a finite state machine which acts as a run-time monitor for $\mathcal{CL}$ contracts.

1 Introduction

Internet inter-business collaborations, virtual organizations, and web services, usually communicate through service exchanges which respect an implicit or explicit *contract*. Such a contract must unambiguously determine correct interactions, and what are the exceptions allowed, or penalties imposed in case of incorrect behavior.

Legal contracts, as found in the usual judicial or commercial arena, may serve as basis for defining such machine-oriented *electronic contracts* (or e-contracts for short). Ideally, e-contracts should be shown to be contradiction-free both internally, and with respect to the governing policies under which the contract is enacted. Moreover, there must be a run-time system ensuring that the contract is respected. In other words, contracts should be amenable to formal analysis allowing both static and dynamic verification, and therefore written in a formal language. In this paper we are interested in the run-time monitoring of electronic contracts, and not in the static verification of their consistency or conformance with policies.

$\mathcal{CL}$, introduced in [17], is an action-based formal language tailored for writing e-contracts, and it has the following properties: (1) Avoids the most common philosophical paradoxes of deontic logic; (2) Has a formal semantics given in terms of Kripke structures; (3) It is possible to express in the language (conditional) obligations, permission and prohibition over concurrent actions; (4) It is possible to express *contrary-to-duty obligations* (CTD) and *contrary-to-prohibitions* (CTP). Both constructions specify the obligation/prohibition to be fulfilled and which is the *reparation*/*penalty* to be applied in case of violation.

* The appendix is for reviewing purpose only and should not be regarded as part of the paper.

** E-mail: kyas@ifi.uio.no and cristi@ifi.uio.no and gerardo@ifi.uio.no

$$
\begin{array}{rcl}
\mathcal{C} & := & \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\beta]\mathcal{C} \mid \top \mid \ \bot \\
\mathcal{C}_O & := & O_{\mathcal{C}}(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\
\mathcal{C}_P & := & P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\
\mathcal{C}_F & := & F_{\mathcal{C}}(\alpha) \mid \mathcal{C}_F \vee [\alpha]\mathcal{C}_F \\
\alpha & := & \mathbf{0} \mid \mathbf{1} \mid a \mid \alpha \& \alpha \mid \alpha \cdot \alpha \mid \alpha + \alpha \\
\beta & := & \mathbf{0} \mid \mathbf{1} \mid a \mid \beta \& \beta \mid \beta \cdot \beta \mid \beta + \beta \mid \beta^* \mid \mathcal{C}?
\end{array}
$$

Table 1. Syntax of the $\mathcal{CL}$ language to use for specifying contracts.

The use of e-contracts, and in particular of $\mathcal{CL}$, goes beyond the application domain of service-exchanges, comprising component-based development systems, fault-tolerant and embedded systems.

The main contribution of this paper is an automatic procedure for obtaining a run-time monitor for contracts, directly extracted from the $\mathcal{CL}$ specification. The road-map of the paper starts by recalling main results on $\mathcal{CL}$ in Section 2. We give a trace semantics for the expressions of $\mathcal{CL}$ in Section 3. This expresses the fact that a trace respects (does not violate) a contract clause (expression of $\mathcal{CL}$). In Section 4.1 we show how to construct for a contract an alternating Büchi automaton which recognizes exactly all the traces respecting the contract. The automaton is used in Section 4.2 for constructing the monitor as a Moore machine (for monitoring the contract).

Though in this paper we concentrate on theoretical aspects, we show the feasibility of our approach on the following small didactic example. It states one contract clause which we use throughout the paper to exemplify some of the main concepts we define.

Example 1. "If the *Client* exceeds the bandwidth limit then (s)he must pay $[price]$ immediately, or (s)he must delay the payment and notify the *Provider* by sending an e-mail. If in breach of the above (s)he must pay double."

2 $\mathcal{CL}$ – A Formal Language for Contracts

$\mathcal{CL}$ is an *action-based* language for writing contracts [17]. In this paper we are interested in monitoring the actions in a contract. Therefore, we give here a slightly different version of $\mathcal{CL}$ where we have dropped the assertions from the old $\mathcal{CL}$, keeping only the modalities over actions. Other differences are in the expressivity: we have incorporated the Kleene star operator over the actions in the dynamic box modality, and we have attached to the obligations the corresponding reparations (modelling the CTDs directly).

The syntax of $\mathcal{CL}$ is defined by the grammar in Table 1. The formal semantics in terms of traces is given later in Section 3 (see Table 2). In what follows we provide explanations of the $\mathcal{CL}$ syntax and define our notation and terminology.

We call an expression $\mathcal{C}$ a (general) *contract clause*. $\mathcal{C}_O$, $\mathcal{C}_P$, and $\mathcal{C}_F$ are called respectively *obligation*, *permission*, and *prohibition* clauses. We call $O_{\mathcal{C}}(\alpha)$, $P(\alpha)$, and $F_{\mathcal{C}}(\alpha)$ the *deontic modalities*, and they represent the obligation, permission, or prohibition of performing a given *action* α. Intuitively $O_{\mathcal{C}}(\alpha)$ states the obligation to execute α, and the *reparation* $\mathcal{C}$ in case the obligation is *violated*, i.e. whenever α is not performed.[1] The reparation may be any contract clause. Obligations without reparations

[1] The modality $O_{\mathcal{C}}(\alpha)$ (resp. $F_{\mathcal{C}}(\alpha)$) represents what is called CTD (resp. CTP) in the deontic logic community.

are written as $O_\perp(\alpha)$ where $\perp$ (and conversely $\top$) is the Boolean `false` (respectively `true`). We usually write $O(\alpha)$ instead of $O_\perp(\alpha)$. The prohibition modality $F_\mathcal{C}(\alpha)$ states the actual forbearing of the action $F(\alpha)$ together with the reparation $\mathcal{C}$ in case the prohibition is violated. Note that it is possible to express nested CTDs and CTPs.

We use the classical Boolean operators $\wedge$, $\vee$, and $\oplus$ (exclusive or). The dynamic logic modality $[\cdot]\mathcal{C}$ is parameterized by *actions* β. The expression $[\beta]\mathcal{C}$ states that after the action β is performed $\mathcal{C}$ must hold.

Throughout the paper we denote by $a, b, c \in \mathcal{A}_B$ the *basic actions*, by indexed $\alpha \in \mathcal{A}$ *compound actions*, and by indexed β the actions found in propositional dynamic logic [6] with intersection [7]. Actions α are used inside the deontic modalities, whereas the (more general) actions β are used inside the dynamic modality. An *action* is an expression containing one or more of the following binary constructors: *choice* "$+$", *sequence* "$\cdot$", *concurrency* "$\&$" and are constructed from the basic actions $a \in \mathcal{A}_B$ and $\mathbf{0}$ and $\mathbf{1}$ (called the *violating* action and respectively *skip* action). Indepth reading and results related to the α actions can be found in [16]. Here we give just the necessary intuitions behind actions. Actions β have the extra operators Kleene star * and *test* $?$.[2] To avoid using parentheses we give a precedence over the constructors: $\& > \cdot > +$.

The $[\cdot]$ modality allows having a *test* inside[3], and $[\mathcal{C}_1?]\mathcal{C}_2$ must be understood as $\mathcal{C}_1 \Rightarrow \mathcal{C}_2$. In $\mathcal{CL}$ we can write *conditional* obligations, permissions and prohibitions of two different kinds. As an example let us consider conditional obligations. The first kind is represented by $[\beta]O(\alpha)$, which may be read as "after performing β, one is obliged to do α". The second kind is modeled using the test operator $?$: $[\mathcal{C}?]O(\alpha)$, which is read as "If $\mathcal{C}$ holds then one is obliged to perform α". Similarly for permission and prohibition.

The properties of the actions α can be summarized as: $(\mathcal{A}, +, \cdot, \mathbf{0}, \mathbf{1})$ is the algebraic structure of an idempotent semiring. At this point we give an informal intuition of the actions (elements) of $\mathcal{A}$: actions are considered to be performed by somebody (being that a person, a program, or an agent). We talk about "performing" actions and not of *processes executing actions* and operational semantics; we do not discuss such semantics in this paper. *Concurrent actions*, denoted by $\alpha_\&$, are actions of $\mathcal{A}_B^\& \subseteq \mathcal{A}$ generated from basic actions using only the $\&$ constructor (e.g. $a,\ a\&a, a\&b \in \mathcal{A}_B^\&$ and $a+b, a\&b+c, a\cdot b \notin \mathcal{A}_B^\&$). Note that $\mathcal{A}_B^\&$ is finite because $\mathcal{A}_B$ is defined as finite and $\&$ is defined idempotent over basic actions. The structure $(\mathcal{A}, +, \&, \mathbf{0}, \mathbf{1})$ is a commutative semiring. Therefore, we consider concurrent actions of $\mathcal{A}_B^\&$ as sets over basic actions of $\mathcal{A}_B$. We have now a natural way to compare concurrent actions using $\subseteq$ set inclusion. We say that an action, e.g. $a \,\&\, b \,\&\, c$ is *more demanding* than another action, e.g. $a \,\&\, b$ iff $\{a, b\} \subseteq \{a, b, c\}$.

In Example 1 the basic actions are $\mathcal{A}_B = \{e, p, n, d\}$ (standing for "extend bandwidth limit", "pay", "notify by email", and "delay"). An example of a concurrent action is $d\&n \in \mathcal{A}_B^\&$, or even $e \,\&\, p \,\&\, d \,\&\, n$.

[2] The investigation of the PDL actions β can be found in the literature related to dynamic and Kleene algebras; basically they define the regular languages of guarded strings and have the syntax of regular expressions with guards [9, 15].

[3] Note that *tests* cannot occur inside deontic modalities, nor can Kleene *.

Any action α can be equivalently written in a concise and clear way as the canonical form defined below.

Theorem 1 (canonical form). *For any α defined with the operators $+$, $\cdot$, $\&$, there exists an equivalent action denoted by $\underline{\alpha}$ which is called* the canonical form of α *and is defined as follows, where $\alpha^i_{\&} \in \mathcal{A}^{\&}_B$ and α^i is a compound action in canonical form.*

$$\underline{\alpha} = \underset{i\in I}{+}\ \alpha^i_{\&} \cdot \alpha^i$$

A natural and useful view of *action negation* is to say that the negation $\overline{\alpha}$ of action α is the action that *immediately take us outside* the trace of α [2]. With the canonical form it is easy to formally define $\overline{\alpha}$. The negation of an action says intuitively that the action which is negated is not executed. Action negation may be considered as a function $\overline{\ \ } : \mathcal{A} \to \mathcal{A}$ which takes an action α in canonical form and returns another compound action.

Definition 1 (action negation). *The action negation is denoted by $\overline{\alpha}$ and is defined as:*

$$\overline{\alpha} = \overline{\underset{i\in I}{+}\ \alpha^i_{\&} \cdot \alpha^i} = \underset{\gamma\in\overline{R}}{+}\ \gamma\ +\ \underset{i\in I}{+}\ \alpha^i_{\&} \cdot \overline{\alpha^i}$$

Where the set $\overline{R} = \{\gamma \,|\, \gamma \in \mathcal{A}^{\&}_B,\ \text{and}\ \forall i \in I, \alpha^i_{\&} \not\sqsubseteq \gamma\}$; (i.e. contains all concurrent actions γ with the property that there is no action $\alpha^i_{\&}$ included in γ).

From *Example 1* consider the negation $\overline{p + d\&n}$ of the action "pay or delay and notify by e-mail". Any action γ which does not "contain" neither p nor $d\&n$ (i.e. $p \not\sqsubseteq \gamma$ and $d\&n \not\sqsubseteq \gamma$) is part of the negation of $p+d\&n$; e.g. $p\&e \notin \overline{p + d\&n}$, but $d, e, d\&e \in \overline{p + d\&n}$.

Example 1 in $\mathcal{CL}$ syntax: The transition from the conventional contract given in the introduction to a formal representation is manual. The following is the $\mathcal{CL}$ expression that we obtain.

$$[e]O_{O_\perp(p\cdot p)}(p + d\&n)$$

In short the expression is read as: After executing the action "exceed bandwidth limit" there is the obligation of choosing between either "paying" or at the same time "delay the payment" and "notify by e-mail". The $\mathcal{CL}$ expression also states the reparation $O_\perp(p\cdot p)$ in case the obligation above is violated which is an obligation of doing twice in a row the action of paying. Note that this second obligation has no reparation attached, therefore if it is violated then the whole contract is violated. Note also that we translate "pay double" into the $\mathcal{CL}$ sequential composition of the same action p of paying.

3 Semantics on Respecting Traces

The present section is devoted to presenting a semantics for $\mathcal{CL}$ with the goal of monitoring electronic contracts. For this we are interested in identifying the *respecting* and *violating* traces of actions. We follow the many works in the literature which have a presentation based on traces e.g. [14]. We first give brief definitions used throughout the rest of the paper.

Definitions (traces): Consider a *trace* denoted $\sigma = a_0, a_1, \ldots$ as an ordered sequence of concurrent actions. Formally a trace is a map $\sigma : \mathbb{N} \rightarrow \mathcal{A}_B^{\&}$ from natural numbers (denoting positions) to concurrent actions from $\mathcal{A}_B^{\&}$. Take $m_\sigma \in \mathbb{N} \cup \infty$ to be the length of a trace.[4] A (infinite) trace which from some position m_σ onwards has only action $\mathbf{1}$ is considered *finite*. We use ε to denote the *empty trace*. We denote by $\sigma(i)$ the *element* of a trace at position i, by $\sigma(i..j)$ a finite *subtrace*, and by $\sigma(i..)$ the infinite subtrace starting at position i in σ. The *concatenation* of two traces σ' and σ'' is denoted $\sigma'\sigma''$ and is defined iff the trace σ' is finite; $\sigma'\sigma''(i) = \sigma'(i)$ if $i < m_{\sigma'}$ and $\sigma'\sigma''(i) = \sigma''(i - m_{\sigma'})$ for $i \geq m_{\sigma'}$ (e.g. $\sigma(0)$ is the first action of a trace, $\sigma = \sigma(0..i)\sigma'$ where $\sigma' = \sigma(i{+}1..)$).

Definitions (trees): A *tree* is a prefix-closed subset $T \subset \mathbb{N}^*$ s.t. if $xc \in T$ with $x \in T$ a tree node and $c \in \mathbb{N}$ then $xc' \in T,\ \forall c' < c$. We call xc the *successors* of x. A node with no successors is called *leaf*. The root of the tree is the empty string ε. A *Σ-labeled tree* is a pair $(T, \mathcal{V})$ where the *valuation function* $V(x) \in \Sigma$ assigns to each node an element of the alphabet Σ. A *path* τ in a tree T is a set $\tau \subseteq T$ s.t. $\varepsilon \in \tau$ and if $x \in \tau$ then either x is a leaf (in which case τ is called a *full path*) or $\exists c \in \mathbb{N}$ unique and $xc \in \tau$. We denote a path by $x_0, x_1, \ldots$. A path of a Σ-labeled tree $\tau = x_0, x_1, \ldots$ defines an (in)finite word $\alpha = \mathcal{V}(x_0), \mathcal{V}(x_1), \ldots$ over Σ. We denote by $|x|$ the *depth* of the node x in the tree.

We can interpret actions as trees. Most of the results and elaborations of [16] are done to prove the completeness of the interpretation of actions as (specially defined) trees. Consider a function $I_{\mathcal{CA}} : \mathcal{CA} \rightarrow (T, \mathcal{V})$ which interprets each action of $\mathcal{CA}$ as a $\mathcal{A}_B^{\&}$-labeled tree with $\mathcal{V}(\varepsilon) = \mathbf{1}$. For example the action $a + b$ is interpreted as the tree $I_{\mathcal{CA}}(a+b) = (\{\varepsilon, \varepsilon 0, \varepsilon 1\}, \mathcal{V})$ where $\mathcal{V}(\varepsilon 0) = a$ and $\mathcal{V}(\varepsilon 1) = b$. Intuitively, $+$ provides the branching in the tree, and $\cdot$ provides the parent-child relation on each branch. The node labels from $\mathcal{A}_B^{\&}$ encode the concurrency operator $\&$. A trace $\sigma \in T$ is said to be *contained* by the $\mathcal{A}_B^{\&}$-labeled tree $(T, \mathcal{V})$ iff $\exists \tau \subseteq T$ a path and $\sigma(0) = \mathcal{V}(\varepsilon c)$ and if $\sigma(i) = \mathcal{V}(x)$ then $\sigma(i+1) = \mathcal{V}(xc)$ with $x \in \tau$. Naturally, any trace σ which is contained in a tree $I_{\mathcal{CA}}(\alpha)$ of an action α is finite as the trees which interpret actions are of finite depth. We consider here the set of all traces which are full paths in the tree T and denote it by $\|T\| = \{\sigma \mid \sigma \text{ a full path in } T\}$.

A view of the complete set of traces that form the action negation can ease the understanding of some following concepts. Proposition 1 provides this.

Proposition 1 (Characterizing action negation with traces). *The set $\|I_{\mathcal{CA}}(\overline{\alpha})\|$ of traces which are full paths of the tree interpreting the negation of $\alpha = +_{i \in I}\ \alpha_{\&}^i \cdot \alpha^i$ is equal to the following set of traces:*

$$\begin{aligned}\{\overline{\alpha}\} = &\{\sigma \mid\ \sigma = \sigma(0)\varepsilon\ \wedge\ \forall i \in I, \alpha_{\&}^i \not\subseteq \sigma(0)\}\ \cup \\ &\{\sigma \mid\ \sigma = \sigma(0)\sigma(1..)\ \wedge\ \exists i \in I,\ s.t.\ \alpha^i \neq \mathbf{1} \wedge \alpha_{\&}^i = \sigma(0) \wedge \sigma(1..) \in \{\overline{\alpha^i}\}\}.\end{aligned}$$

Definition 2 (Semantics of $\mathcal{CL}$). *We give in Table 2 a recursive definition of the* satisfaction relation $\models$ *over pairs $(\sigma, \mathcal{C})$ of traces and contracts; it is usually written $\sigma \models \mathcal{C}$ and it is read as "trace σ* respects *the contract (clause) $\mathcal{C}$". We write $\sigma \not\models \mathcal{C}$ instead of $(\sigma, \mathcal{C}) \notin \models$ and read it as "σ* violates *$\mathcal{C}$."*

[4] When σ is obvious from the context we use just m instead of m_σ.

$\sigma \models \mathcal{C}_1 \wedge \mathcal{C}_2$ if $\sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$.
$\sigma \models \mathcal{C}_1 \vee \mathcal{C}_2$ if $\sigma \models \mathcal{C}_1$ or $\sigma \models \mathcal{C}_2$.
$\sigma \models \mathcal{C}_1 \oplus \mathcal{C}_2$ if ($\sigma \models \mathcal{C}_1$ and $\sigma \not\models \mathcal{C}_2$) or ($\sigma \not\models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$).
$\sigma \models [\alpha_\&]\mathcal{C}$ if $\alpha_\& \subseteq \sigma(0)$ and $\sigma(1..) \models \mathcal{C}$, or $\alpha_\& \not\subseteq \sigma(0)$.
$\sigma \models [\beta \cdot \beta']\mathcal{C}$ if $\sigma \models [\beta][\beta']\mathcal{C}$.
$\sigma \models [\beta + \beta']\mathcal{C}$ if $\sigma \models [\beta]\mathcal{C}$ and $\sigma \models [\beta']\mathcal{C}$.
$\sigma \models [\beta^*]\mathcal{C}$ if $\sigma \models \mathcal{C}$ and $\sigma \models [\beta][\beta^*]\mathcal{C}$.
$\sigma \models [\mathcal{C}_1?]\mathcal{C}_2$ if $\sigma \not\models \mathcal{C}_1$, or if $\sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$.
$\sigma \models O_\mathcal{C}(\alpha_\&)$ if $\alpha_\& \subseteq \sigma(0)$, or if $\sigma(1..) \models \mathcal{C}$.
$\sigma \models O_\mathcal{C}(\alpha \cdot \alpha')$ if $\sigma \models O_\mathcal{C}(\alpha)$ and $\sigma \models [\alpha]O_\mathcal{C}(\alpha')$.
$\sigma \models O_\mathcal{C}(\alpha + \alpha')$ if $\sigma \models O_\bot(\alpha)$ or $\sigma \models O_\bot(\alpha')$ or $\sigma \models [\overline{\alpha + \alpha'}]\mathcal{C}$.
$\sigma \models F_\mathcal{C}(\alpha_\&)$ if $\alpha_\& \not\subseteq \sigma(0)$, or if $\alpha_\& \subseteq \sigma(0)$ and $\sigma(1..) \models \mathcal{C}$.
$\sigma \models F_\mathcal{C}(\alpha \cdot \alpha')$ if $\sigma \models F_\bot(\alpha)$ or $\sigma \models [\alpha]F_\mathcal{C}(\alpha')$.
$\sigma \models F_\mathcal{C}(\alpha + \alpha')$ if $\sigma \models F_\mathcal{C}(\alpha)$ and $\sigma \models F_\mathcal{C}(\alpha')$.
$\sigma \models [\overline{\alpha_\&}]\mathcal{C}$ if $\alpha_\& \not\subseteq \sigma(0)$ and $\sigma(1..) \models \mathcal{C}$, or if $\alpha_\& \subseteq \sigma(0)$.
$\sigma \models [\overline{\alpha \cdot \alpha'}]\mathcal{C}$ if $\sigma \models [\overline{\alpha}]\mathcal{C}$ and $\sigma \models [\alpha][\overline{\alpha'}]\mathcal{C}$.
$\sigma \models [\overline{\alpha + \alpha'}]\mathcal{C}$ if $\sigma \models [\overline{\alpha}]\mathcal{C}$ or $\sigma \models [\overline{\alpha'}]\mathcal{C}$.

Table 2. Trace semantics of $\mathcal{CL}$.

Since some of the cases from the definition of $\models$ are typical for modal (dynamic) logics, we only comment on those that are particular to our logic. A trace σ respects an obligation $O_\mathcal{C}(\alpha_\&)$ if either of the two complementary conditions (see Proposition 2) is satisfied. The first condition deals with the obligation itself: the trace σ respects the obligation $O(\alpha_\&)$ if the first action of the trace includes $\alpha_\&$. Otherwise, in case the obligation is violated,[5] the only way to fulfill the contract is by respecting the reparation $\mathcal{C}$; i.e. $\sigma(1..) \models \mathcal{C}$. Respecting an obligation of a choice action $O_\mathcal{C}(\alpha_1 + \alpha_2)$ means that it must be executed one of the actions α_1 or α_2 completely; i.e. obligation needs to consider only one of the choices. If none of these is entirely executed then a violation occurs (thus the negation of the action is needed) so the reparation $\mathcal{C}$ must be respected. An important requirement when modelling electronic contracts is that the obligation of a sequence of actions $O_\mathcal{C}(\alpha \cdot \alpha')$ must be equal to the obligation of the first action $O_\mathcal{C}(\alpha)$ and after the first obligation is respected the second obligation must hold $[\alpha]O_\mathcal{C}(\alpha')$. Note that if $O_\mathcal{C}(\alpha)$ is violated then it is required that the second obligation is discarded, and the reparation $\mathcal{C}$ must hold. Violating $O_\mathcal{C}(\alpha)$ means that α is not executed and thus, by the semantic definition, $[\alpha]O_\mathcal{C}(\alpha')$ holds regardless of $O_\mathcal{C}(\alpha')$.

In the Appendix, the proof of Proposition 2 provides for a more clear understanding of why the two conditions in the semantical definition of $O_\mathcal{C}(\alpha_\&)$ are complementary and therefore why the second condition does not need the explicit specification of the

[5] Violation of an obligatory action is encoded by the action negation.

negation of the concurrent action. The same result is also useful in giving the completeness of the semantical definitions of $[\alpha_{\&}]\mathcal{C}$, $[\overline{\alpha_{\&}}]\mathcal{C}$, or $F_{\mathcal{C}}(\alpha_{\&})$. That is, we get as an immediate corollary that the conditions on traces cover all the possible traces. For convenience we define an *enclosing* relation over traces $\sigma \supseteq \sigma'$ iff $\forall i \in \mathbb{N}$, $\sigma'(i) \subseteq \sigma(i)$. Note that this definition requires that $m_\sigma \geq m_{\sigma'}$.

Proposition 2. *For an arbitrary action α, any infinite trace σ is either starting with a trace bigger w.r.t. $\supseteq$ then a complete path of $I_{\mathcal{CA}}(\alpha)$ or it starts with a trace bigger than a complete path of $I_{\mathcal{CA}}(\overline{\alpha})$.*

From [8] we know how to encode LTL only with the dynamic $[\cdot]$ modality and the Kleene *; e.g. "*always* obliged to do α" is encoded as $[(+_{\gamma \in \mathcal{A}_B^{\&}} \gamma)^*]O(\alpha)$. The action $+_{\gamma \in \mathcal{A}_B^{\&}} \gamma$ is read as "choice between *any* concurrent action" and we denote it by $\mathbf{any}$.

Example 1 as traces: Consider the expression on page 4 which encodes in $\mathcal{CL}$ the contract clause of Example 1 from the introduction. We give here few examples of traces of actions which *respect* the contract clause:

- e, p ("exceed bandwidth limit" and then "pay") which respects the contract because it respects the top level obligation;
- e, d, p, p ("exceed bandwidth limit" and then "delay payment" after which "pay" twice in a row) which even if it violates the top level obligation because it does not notify by e-mail at the same time when "delaying payment", it still respects the reparation by paying twice;
- p, p, p ("pay" three times in a row) because every trace which does not start with the action e respects the contract.

Examples of traces which *violate* the clause are:

- e, e, e (constantly "exceeding the bandwidth limit") which violates both the first obligation and the second one by not paying;
- e, d, d (after "exceeding the bandwidth limit" it constantly "delays the payment") which again violates both obligations.

4 Monitoring $\mathcal{CL}$ Specifications of Contracts

4.1 Satisfiability checking for $\mathcal{CL}$ using alternating automata

Automata theoretic approach to satisfiability of temporal logics was introduced in [22] and has been extensively used and developed since (see [10] for a more recent overview of the field and a particularly detailed presentation of alternating tree automata and the automata approach to branching time logics). We recall first basic theory of automata on infinite objects. We follow the presentation of Vardi [20, 21] and try to use the same terminology and notation. Given an alphabet Σ, a *word over* Σ is a sequence $a_0, a_1 \ldots$ of symbols from Σ. The set of *infinite words* is denoted by Σ^ω.

We denote by $\mathcal{B}^+(X)$ the set of positive Boolean formulas θ (i.e. containing only $\wedge$ and $\vee$, and not the $\neg$) over the set X together with the formulas $\mathbf{true}$ and $\mathbf{false}$. For example $\theta = (s_1 \vee s_2) \wedge (s_3 \vee s_4)$ where $s_i \in X$. A subset $Y \subseteq X$ is said to *satisfy* a

formula θ iff the truth assignment which assigns *true* only to the elements of Y assigns *true* also to θ. In the example, the set $\{s_1, s_3\}$ satisfies θ; but this set is not unique.

Alternating automata [4] combine existential choice of nondeterministic finite automata (i.e. disjunction) with the universal choice (i.e. conjunction) of $\forall$-automata [11] (where a run of the automaton says that from a state the automaton must move to *all* the next states given by the transition function, making a copy of itself for each next state). For example the transition $\rho(s, a) = \{s_1, s_2, s_3\}$ of a NFA (which takes a state s and a symbol a and returns a set of states to which the automaton can move by reading the symbol a in the state s) can be equivalently viewed as the Boolean formula $\theta = s_1 \vee s_2 \vee s_3$. For a $\forall$-automaton the same transition is encoded by the $\theta = s_1 \wedge s_2 \wedge s_3$.

An *alternating Büchi automaton* [12] is a tuple $A = (S, \Sigma, s_0, \rho, F)$, where S is a finite nonempty set of *states*, Σ is a finite nonempty *alphabet*, $s_0 \in S$ is the *initial* state, and $F \subseteq S$ is the set of *accepting* states. The automaton can move from one state when it reads a symbol from Σ according to the transition function $\rho : S \times \Sigma \to \mathcal{B}^+(S)$. For example $\rho(s_0, a) = (s_1 \vee s_2) \wedge (s_3 \vee s_4)$ means that the automaton moves from s_0 when reading a to state s_1 or s_2 and at the same time to state s_3 or s_4. Intuitively the automaton chooses for each transition $\rho(s, a) = \theta$ one set $S' \in S$ which satisfies θ and spawns a copy of itself for each state $s_i \in S'$ which should test the acceptance of the remaining word from that state s_i.

Because the alternating automaton moves to all the states of a (nondeterministically chosen) satisfying set of θ, a run of the automaton is a *tree* of states. Formally, a run of the alternating automaton on an input word $\alpha = a_0, a_1, \ldots$ is an S-labeled tree $(T, \mathcal{V})$ (i.e. the nodes of the tree are labeled by state names of the automaton) such that $\mathcal{V}(\varepsilon) = s_0$ and the following hold:

> for a node x with $|x| = i$ s.t. $\mathcal{V}(x) = s$ and $\rho(s, a_i) = \theta$ then x has k children $\{x_1, \ldots, x_k\}$ which is the number of states in the chosen satisfying set of states of θ, say $\{s_1, \ldots, s_k\}$, and the children are labeled by the states in the satisfying set; i.e. $\{\mathcal{V}(x_1) = s_1, \ldots, \mathcal{V}(x_k) = s_k\}$.

For example, if $\rho(s_0, a) = (s_1 \vee s_2) \wedge (s_3 \vee s_4)$ then the nodes of the run tree at the first level have one label among s_1 or s_2 and one label among s_3 or s_4. When $\rho(\mathcal{V}(x), a) = \mathbf{true}$, then x need not have any children; i.e. the branch reaching x is finite and ends in x. A run tree of an alternating Büchi automaton is *accepting* if every infinite branch of the tree includes infinitely many nodes labeled by accepting states of F. Note that the run tree may also have finite branches in the cases when the transition function returns $\mathbf{true}$.

Complementation of alternating automata is straight forward. It involves constructing a *dual* of a Boolean formula θ of $\mathcal{B}^+(S)$. The dual $\overline{\theta}$ is obtained from θ by switching $\vee$ and $\wedge$, and by switching $\mathbf{true}$ and $\mathbf{false}$. For an automaton $A = (S, \Sigma, s_0, \rho, F)$ we define the negated automaton $\overline{A} = (S, \Sigma, s_0, \overline{\rho}, F)$ where $\overline{\rho}(s, a) = \overline{\rho(s, a)}$. The language accepted by $\overline{A}$ is the complement of the language accepted by A [4].

Fischer-Ladner closure for $\mathcal{CL}$: For constructing the alternating automaton for a $\mathcal{CL}$ expression we need the Fischer-Ladner closure [6] for our $\mathcal{CL}$ logic. We follow the presentation in [8] and use similar terminology. We define a function $FL : \mathcal{CL} \to 2^{\mathcal{CL}}$ which for each expression $\mathcal{C}$ of the logic $\mathcal{CL}$ returns the set of its subex-

$$
\begin{aligned}
FL(\top) &\triangleq \{\top\} \qquad FL(\bot) \triangleq \{\bot\} \qquad FL(P(\alpha)) \triangleq \{P(\alpha)\} \\
FL(\mathcal{C}_1 \wedge \mathcal{C}_2) &\triangleq \{\mathcal{C}_1 \wedge \mathcal{C}_2\} \cup FL(\mathcal{C}_1) \cup FL(\mathcal{C}_2) \\
FL(\mathcal{C}_1 \vee \mathcal{C}_2) &\triangleq \{\mathcal{C}_1 \vee \mathcal{C}_2\} \cup FL(\mathcal{C}_1) \cup FL(\mathcal{C}_2) \\
FL(\mathcal{C}_1 \oplus \mathcal{C}_2) &\triangleq \{\mathcal{C}_1 \oplus \mathcal{C}_2\} \cup FL(\mathcal{C}_1) \cup FL(\mathcal{C}_2) \\
FL([\beta]\mathcal{C}) &\triangleq FL^{\Box}([\beta]\mathcal{C}) \cup FL(\mathcal{C}) \\
FL^{\Box}([\beta_{\&}]\mathcal{C}) &\triangleq \{[\beta_{\&}]\mathcal{C}\} \\
FL^{\Box}([\beta \cdot \beta']\mathcal{C}) &\triangleq \{[\beta \cdot \beta']\mathcal{C}\} \cup FL^{\Box}([\beta][\beta']\mathcal{C}) \cup FL^{\Box}([\beta']\mathcal{C}) \\
FL^{\Box}([\beta + \beta']\mathcal{C}) &\triangleq \{[\beta + \beta']\mathcal{C}\} \cup FL^{\Box}([\beta]\mathcal{C}) \cup FL^{\Box}([\beta']\mathcal{C}) \\
FL^{\Box}([\beta^*]\mathcal{C}) &\triangleq \{[\beta^*]\mathcal{C}\} \cup FL^{\Box}([\beta][\beta^*]\mathcal{C}) \\
FL^{\Box}([\mathcal{C}_1?]\mathcal{C}_2) &\triangleq \{[\mathcal{C}_1?]\mathcal{C}_2\} \cup FL(\mathcal{C}_1) \\
FL(O_{\mathcal{C}}(\alpha_{\&})) &\triangleq \{O_{\mathcal{C}}(\alpha_{\&})\} \cup FL(\mathcal{C}) \\
FL(O_{\mathcal{C}}(\alpha \cdot \alpha')) &\triangleq \{O_{\mathcal{C}}(\alpha \cdot \alpha')\} \cup FL(O_{\mathcal{C}}(\alpha)) \cup FL([\alpha]O_{\mathcal{C}}(\alpha')) \\
FL(O_{\mathcal{C}}(\alpha + \alpha')) &\triangleq \{O_{\mathcal{C}}(\alpha + \alpha')\} \cup FL(O_{\bot}(\alpha)) \cup FL(O_{\bot}(\alpha')) \cup FL(\mathcal{C}) \\
FL(F_{\mathcal{C}}(\alpha_{\&}) &\triangleq \{F_{\mathcal{C}}(\alpha_{\&}\} \cup FL(\mathcal{C}) \\
FL(F_{\mathcal{C}}(\alpha \cdot \alpha')) &\triangleq \{F_{\mathcal{C}}(\alpha \cdot \alpha')\} \cup FL(F_{\bot}(\alpha)) \cup FL(F_{\mathcal{C}}(\alpha')) \\
FL(F_{\mathcal{C}}(\alpha + \alpha')) &\triangleq \{F_{\mathcal{C}}(\alpha + \alpha')\} \cup FL(F_{\mathcal{C}}(\alpha)) \cup FL(F_{\mathcal{C}}(\alpha'))
\end{aligned}
$$

Table 3. Computing the Fisher-Ladner Closure

pressions. FL was introduced for process logics in order to deal with the compound actions. For avoiding circularity in the definition of FL an auxiliary function is needed $FL^{\Box} : \{[\beta]\mathcal{C} \mid \beta \text{ an action}\} \rightarrow 2^{\mathcal{CL}}$ (see [8]). The functions FL and $FL^{\Box}$ are defined inductively in Table 3.

Theorem 2 (automaton construction). *Given a $\mathcal{CL}$ expression $\mathcal{C}$, one can build an alternating Büchi automaton $A^{\mathcal{N}}(\mathcal{C})$ which will accept all and only the traces σ respecting the contract expression.*

Proof: Take an expression $\mathcal{C}$ of $\mathcal{CL}$, we construct the alternating Büchi automaton $A^{\mathcal{N}}(\mathcal{C}) = (S, \Sigma, s_0, \rho, F)$ as follows. The alphabet $\Sigma = \mathcal{A}_B^{\&}$ consists of the finite set of concurrent actions; that is basic actions $a, b, \ldots$ and actions composed only by using the concurrent composition operator, like $a\&b$. Therefore the automaton accepts traces as defined in Section 3. The set of states $S = FL(\mathcal{C}) \cup \overline{FL(\mathcal{C})}$ contains the subexpressions of the start expression $\mathcal{C}$ and their negations. Note that in $\mathcal{CL}$ the negation $\neg\mathcal{C}$ is $\mathcal{C} \Rightarrow \bot$ which is $[\mathcal{C}?]\ \bot$, thus $\forall \mathcal{C} \in FL(\mathcal{C})$ then $[\mathcal{C}?]\ \bot \in \overline{FL(\mathcal{C})}$. The initial state s_0 is the expression $\mathcal{C}$ itself. The set of final states F contains all the expressions of the type $[\beta^*]\mathcal{C}$.

The transition function $\rho : S \times \mathcal{A}_B^{\&} \rightarrow \mathcal{B}^+(S)$ is defined in Table 4 and is based on the dualizing construction we have seen before, only that the dual of a state $\overline{\mathcal{C}}$ is the state $[\mathcal{C}?]\ \bot$ containing the negation of the expression. It is easy to see that if a run tree

$$
\begin{aligned}
\rho(\bot,\gamma) &\triangleq \mathbf{false} \qquad \rho(\top,\gamma) \triangleq \mathbf{true} \qquad \rho(P(\alpha),\gamma) \triangleq \mathbf{true}\\
\rho(\mathcal{C}_1 \wedge \mathcal{C}_2,\gamma) &\triangleq \rho(\mathcal{C}_1,\gamma) \wedge \rho(\mathcal{C}_2,\gamma) \qquad \rho(\mathcal{C}_1 \vee \mathcal{C}_2,\gamma) \triangleq \rho(\mathcal{C}_1,\gamma) \vee \rho(\mathcal{C}_2,\gamma)\\
\rho(\mathcal{C}_1 \oplus \mathcal{C}_2,\gamma) &\triangleq (\rho(\mathcal{C}_1,\gamma) \wedge \overline{\rho(\mathcal{C}_2,\gamma)}) \vee (\overline{\rho(\mathcal{C}_1,\gamma)} \wedge \rho(\mathcal{C}_2,\gamma))\\
\rho(O_{\mathcal{C}}(\alpha_{\&}),\gamma) &\triangleq \text{ if } \alpha_{\&} \subseteq \gamma \text{ then } \mathbf{true} \text{ else } \mathcal{C}\\
\rho(O_{\mathcal{C}}(\alpha \cdot \alpha'),\gamma) &\triangleq \rho(O_{\mathcal{C}}(\alpha),\gamma) \wedge \rho([\alpha]O_{\mathcal{C}}(\alpha'),\gamma)\\
\rho(O_{\mathcal{C}}(\alpha + \alpha'),\gamma) &\triangleq \rho(O_{\bot}(\alpha),\gamma) \vee \rho(O_{\bot}(\alpha'),\gamma) \vee \mathcal{C}\\
\rho(F_{\mathcal{C}}(\alpha_{\&}),\gamma) &\triangleq \text{ if } \alpha_{\&} \not\subseteq \gamma \text{ then } \mathbf{true} \text{ else } \mathcal{C}\\
\rho(F_{\mathcal{C}}(\alpha \cdot \alpha'),\gamma) &\triangleq \rho(F_{\bot}(\alpha),\gamma) \vee F_{\mathcal{C}}(\alpha')\\
\rho(F_{\mathcal{C}}(\alpha + \alpha'),\gamma) &\triangleq \rho(F_{\mathcal{C}}(\alpha),\gamma) \wedge \rho(F_{\mathcal{C}}(\alpha'),\gamma)\\
\rho([\alpha_{\&}]\mathcal{C},\gamma) &\triangleq \text{ if } \alpha_{\&} \subseteq \gamma \text{ then } \mathcal{C} \text{ else } \mathbf{true}\\
\rho([\beta \cdot \beta']\mathcal{C},\gamma) &\triangleq \rho([\beta][\beta']\mathcal{C},\gamma)\\
\rho([\beta + \beta']\mathcal{C},\gamma) &\triangleq \rho([\beta]\mathcal{C},\gamma) \wedge \rho([\beta']\mathcal{C},\gamma)\\
\rho([\beta^*]\mathcal{C},\gamma) &\triangleq \rho(\mathcal{C},\gamma) \wedge \rho([\beta][\beta^*]\mathcal{C},\gamma)\\
\rho([\mathcal{C}_1?]\mathcal{C}_2,\gamma) &\triangleq \overline{\rho(\mathcal{C}_1,\gamma)} \vee (\rho(\mathcal{C}_1,\gamma) \wedge \rho(\mathcal{C}_2,\gamma))
\end{aligned}
$$

Table 4. Transition Function of Alternating Büchi Automaton

has an infinite path then this path goes infinitely often through a state of the form $[\beta^*]\mathcal{C}$, thus explaining the F set. By looking at the definition of ρ we see that the expression $[\beta^*]\mathcal{C}$ is the only expression which requires repeated evaluation of itself at a later point in the run. This causes the infinite unwinding in the run tree.

The rest of the proof shows the correctness of the automaton construction.

Soundness: given an accepting run tree $(T, \mathcal{V})$ of $A^{\mathcal{N}}(\mathcal{C})$ over a trace σ we prove that $\forall x \in T$ a node of the run tree with depth $|x| = i,\ i \geq 0$, labeled by $\mathcal{V}(x) = \mathcal{C}_x$ a state of the automaton represented by a subexpression $\mathcal{C}_x \in FL(\mathcal{C})$, it is the case that $\sigma(i..) \models \mathcal{C}_x$. Thus we have as a special case that also $\sigma(0..) \models \mathcal{V}(\varepsilon) = \mathcal{C}$, which means that if the automaton $A^{\mathcal{N}}(\mathcal{C})$ accepts a trace σ then the trace respects the initial contract $\mathcal{C}$. We use induction on the structure of the expression $\mathcal{C}_x$.

Completeness: given a trace σ s.t. $\sigma \models \mathcal{C}$ we prove that the constructed automaton $A^{\mathcal{N}}(\mathcal{C})$ accepts σ (i.e. there exists an accepting run tree $(T, \mathcal{V})$ over the trace σ). □

Example 1 as alternating automata: We shall now briefly show how for the $\mathcal{CL}$ expression $\mathcal{C} = [e]O_{O_{\bot}(p \cdot p)}(p + d\&n)$ of page 4 we construct an alternating automaton which accepts all the traces (like the ones we have seen on page 7) that satisfy $\mathcal{C}$ and none others. The Fischer-Ladner closure of $\mathcal{C}$ generates the following set of subexpressions:

$$FL(\mathcal{C}) = \{\mathcal{C}, O_{O_{\bot}(p \cdot p)}(p + d\&n), O_{\bot}(p), \bot, O_{\bot}(d\&n), O_{\bot}(p \cdot p), [p]O_{\bot}(p)\}$$

The set $\mathcal{A}_B^{\&}$ of concurrent actions is the set $\{e, p, n, d\}^{\&}$ of basic actions closed under the constructor $\&$. The alternating automaton is:

$$A^{\mathcal{N}}(\mathcal{C}) = (FL(\mathcal{C}) \cup \overline{FL(\mathcal{C})}, \{e, p, n, d\}^{\&}, \mathcal{C}, \rho, \emptyset)$$

Note that there is no expression of the form $[\beta^*]\mathcal{C}$ in FL because we have no recursion in our original contract clause from Example 1, therefore the set of final states is empty. This means that the automaton is accepting all run trees which end in a state where the transition function returns **true** on the input symbol.[6]

The transition function ρ is defined in table below where $\mathcal{C}_1 = O_{O_\perp(p\cdot p)}(p + d\&n)$:

$\rho(state, action)$	e	p	d	$e\&d$	$e\&p$	$d\&n$	$e\&d\&n$
$\mathcal{C}$	$\mathcal{C}_1$	**true**	**true**	$\mathcal{C}_1$	$\mathcal{C}_1$	**true**	$\mathcal{C}_1$
$\mathcal{C}_1$	$O_\perp(p \cdot p)$	**true**	$O_\perp(p \cdot p)$	$O_\perp(p \cdot p)$	**true**	**true**	**true**
$O_\perp(p)$	$\perp$	**true**	$\perp$	$\perp$	**true**	$\perp$	$\perp$
$O_\perp(d\&n)$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	**true**	**true**
$O_\perp(p \cdot p)$	$\perp$	$O_\perp(p)$	$\perp$	$\perp$	$O_\perp(p)$	$\perp$	$\perp$
$[p]O_\perp(p)$	**true**	$O_\perp(p)$	**true**	**true**	$O_\perp(p)$	**true**	**true**

Computing the values in the table above is easy; e.g.:

$$\begin{aligned} \rho(\mathcal{C}_1, e) &= \rho(O_\perp(p), e) \vee \rho(O_\perp(d\&n), e) \vee O_\perp(p \cdot p) \\ &= \perp \vee \perp \vee O_\perp(p \cdot p) \end{aligned}$$

Because from the state $\perp$ nothing can be accepted (as it generates only **false**) we have written in the table only $O_\perp(p \cdot p)$. There are 2^4 labels in the alphabet of $A^{\mathcal{N}}(\mathcal{C})$ and we have exemplified only some of the more interesting ones. Moreover, none of the states from $\overline{FL}$ (i.e. $[\mathcal{C}_1?]\ \perp$, the complemented expressions) are reachable nor do they contribute to the computation of any transition to a reachable state (like e.g. $O_\perp(d\&n)$ contributes to the computation of $\rho(\mathcal{C}_1, e)$), so we have not included them in the table. The line for state $\perp$ is omitted as it generates only **false**.

In Figure 1 we picture all the reachable states of the automaton $A^{\mathcal{N}}(\mathcal{C})$. For brevity, we have not represented all the transitions; e.g. from state $\mathcal{C}$ to state $\mathcal{C}_1$ there should be a transition for each label which includes e (like $e\&d$, $e\&p$, or $e\&d\&n$).

For example, the automaton accepts the trace $e, e\&d, p, p$, because starting from the state $\mathcal{C}$, the state $\mathcal{C}_1$ is reached after e, then the state labelled $O_\perp(p \cdot p)$ after $e \,\&\, d$, then $O_\perp(p)$ after p. From this state, the transition function results in **true** on the symbol p, and the automaton accepts the trace.

Conversly, the automaton rejects the trace $e, e \,\&\, d, p, e \,\&\, d$, since, from the state labelled $O_\perp(p)$ the state labelled $\perp$ is reached while reading $e \,\&\, d$. From then on the transition function will never result in **true** *and* the Büchi acceptance set is empty, which means that any infinite extension of the run tree will also not accept.

[6] Note that for this particular example we do not see the power of alternating automata. More, the alternating Büchi automata behaves like a NFA.

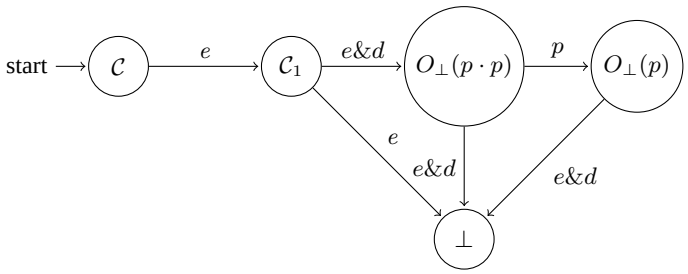

Fig. 1. Sketch of the alternating automaton for the expression $\mathcal{C} = [e]O_{O_\perp(p \cdot p)}(p + d\&n)$.

4.2 Constructing the monitor

We use the method of [1] and we consequently use a 3-valued semantics approach to run-time monitoring. The monitor will generate a sequence of observations, denoted $[\sigma \models \mathcal{C}]$, for a finite trace σ by:

$$[\sigma \models \mathcal{C}] = \begin{cases} \mathbf{tt} & \text{if } \forall \sigma' \in \Sigma^\omega : \sigma\sigma' \models \mathcal{C} \\ \mathbf{ff} & \text{if } \forall \sigma' \in \Sigma^\omega : \sigma\sigma' \not\models \mathcal{C} \\ ? & \text{otherwise} \end{cases}$$

The method of [1] uses the $NBA(\mathcal{C})$ together with the automaton for the negated expression $NBA(\neg\mathcal{C})$, and returns a Moore machine associated with an expression $\mathcal{C}$ which for each state it outputs a symbol of $\{\mathbf{tt}, \mathbf{ff}, ?\}$ if what it has seen until that state respectively *respects* the expression $\mathcal{C}$, *violates* $\mathcal{C}$, or it *don't know yet*.

We can obtain a nondeterministic Büchi automaton $NBA(\mathcal{C})$ from our alternating automaton $A^{\mathcal{N}}(\mathcal{C})$ s.t. both automata accept the same trace language. The method is standard [20] and it constructs an automaton exponentially larger than the input automaton $A^{\mathcal{N}}(\mathcal{C})$, therefore $NBA(\mathcal{C})$ is exponential in the size of the expression.

The method of [1] is the following: take the $NBA(\mathcal{C})$ for which we know that $[\sigma \models \mathcal{C}] \neq \mathbf{ff}$ if there exists a state reachable by reading σ and from where the language accepted by $NBA(\mathcal{C})$ is not empty. Similarly for $[\sigma \models \mathcal{C}] \neq \mathbf{tt}$ when taking the complement of $NBA(\mathcal{C})$ (or equivalently we can take the $NBA(\neg\mathcal{C})$ of the negated formula which is $[\mathcal{C}?]\ \perp$). Construct a function $F : S \to \{\top, \perp\}$ which for each state s of the $NBA(\mathcal{C})$ returns $\top$ iff $\mathcal{L}(NBA(\mathcal{C}), s) \neq \emptyset$ (i.e. the language accepted by $NBA(\mathcal{C})$ from state s is not empty), and $\perp$ otherwise. Using F one can construct a nondeterministic finite automaton $NFA(\mathcal{C})$ accepting finite traces s.t. $\sigma \in \mathcal{L}(NFA(\mathcal{C})$ iff $[\sigma \models \mathcal{C}] \neq \perp$. This is the same NBA only that the set of final states contains all the states mapped by F to $\top$. Similarly construct a $NFA(\neg\mathcal{C})$ from $NBA(\neg\mathcal{C})$. One uses classical techniques to determinize the two NFAs. Using the two obtained DFAs one constructs the monitor as a finite state machine which at each state outputs $\{\mathbf{tt}, \mathbf{ff}, ?\}$ if the input read until that state respectively satisfies the contract clause $\mathcal{C}$, violates it, or it cannot be decided. The monitor is the product of the two $DFA(\mathcal{C})$ and $DFA(\neg\mathcal{C})$.

The method of [1] omits one important requirement that we need for monitoring electronic contracts. We need that the monitor can read (and move to a new state) each

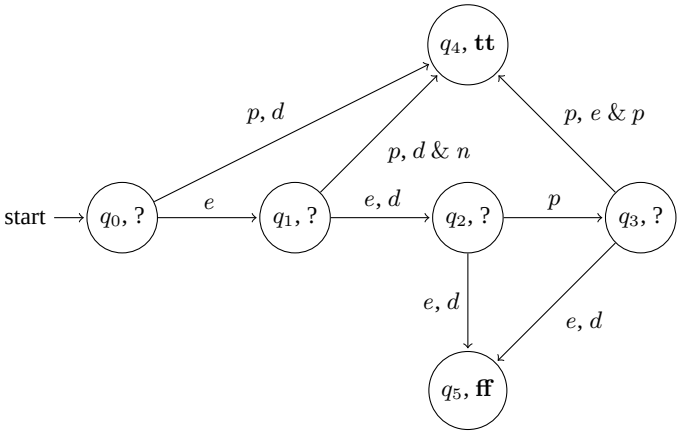

Fig. 2. Sketch of a Run-Time Monitor for the expression $\mathcal{C} = [e]O_{O_\perp(p \cdot p)}(p + d\&n)$.

possible action from the input alphabet. When doing the product of the two DFAs, if one of them does not have a transition for one of the symbols then this is lost for the monitor too. The solution is simple; we add to each DFA a dummy state which is not accepting and which collects all the missing transitions (with the missing labels).

The whole method is proven correct; i.e. $[\sigma \models \mathcal{C}] = \lambda(\rho(s_0, \sigma))$ the semantics of $\mathcal{C}$ on the finite trace σ ($[\sigma \models \mathcal{C}]$) is exactly the output of the Moore machine ($\lambda : S \to \{\mathbf{tt}, \mathbf{ff}, ?\}$ is the output function) from the state reached by reading σ from the starting state s_0. The monitor generated is proven to have size double-exponential in the size of the expression; one exponent coming from the Büchi automaton and the other from determinizing the NBAs [1]. It is known that minimization techniques give good results for such problems and on-the-fly generation of the state space and of the transition relation make the automata manageable. For $\mathcal{CL}$ we get the same size of the final monitor even if we go in the beginning through alternating automata.

Example 1 as run-time monitor: Consider the alternating automaton $A^{\mathcal{N}}(\mathcal{C})$ constructed before. The resulting monitor for this automaton is sketched in Fig. 2. Consider the traces showed on page 7 for the expression $\mathcal{C}$: for a respecting trace e, p the monitor outputs $?, \mathbf{tt}$; for the rejecting trace e, e, e the monitor outputs $?, ?, ?, \mathbf{ff}$.

5 Conclusion

The work reported here may be viewed from different angles. On one hand we use *alternating automata* which has recently gained popularity [10, 21] in the temporal logics community. We apply these to a rather unconventional logic $\mathcal{CL}$ [17], a process logic (PDL [6]) extended with deontic logic modalities [23]. On another hand we presented the formal language $\mathcal{CL}$ with a trace semantics, and showed how we specify electronic contracts using it. Though $\mathcal{CL}$ has been originally designed as a language for specifying electronic contracts in the context of service-oriented architectures, it has been argued that its use may be extremely useful in component-based development systems [13].

Due to the contrary-to-duties and contrary-to-prohibitions, $\mathcal{CL}$ is also suitable to specify properties, and reason about, fault-tolerant systems, similar to what is presented in [3].

From a practical point of view we presented here a first fully automated method of extracting a run-time monitor for a contract formally specified using the $\mathcal{CL}$ logic.

Note that our main objective is not to enforce a contract, but only to monitor it, that is to observe that the contract is indeed satisfied. In our opinion monitoring contracts is more reasonable than enforcing them, since in our contract such agreements are supposed to be written between different parties who have already agreed on their content. In other words, a contract must also contain what are the actions to be performed in case of violation of certain clauses.

The trace semantics presented in this paper is intended for monitoring purposes, and not to explain the language $\mathcal{CL}$. Thus, from the trace semantics point of view $[\alpha_{\&}]\mathcal{C}$ is equivalent to $F_{\mathcal{C}}(\alpha_{\&})$, we need such a distinction since this is not the case in $\mathcal{CL}$ (see $\mathcal{CL}$ branching semantics [18]). Another particular issue concerns the $\oplus$ operator which needs some additional explanation. Our trace semantics interprets it as a an exclusive or, but when combined with obligations we might get some counter-intuitive specifications. For instance, according to the trace semantics there is no model for $O(a) \oplus O(a)$ though one may expect to get $O(a)$ instead under an interpretation of $\oplus$ as a *choice* operator. Moreover, the trace semantics will exclude traces starting with $a \,\&\, b$ and those starting with a, a and b, b for the $\mathcal{CL}$ expression $\sigma \models O_{O(b)}(a) \oplus O_{O(a)}(b)$.

Related work: For run-time verification our use of alternating automata on infinite traces of actions is a rather new approach. This is combined with the method of [1] that uses a three value (i.e. *true*, *false*, *inconclusive*) semantics view for run-time monitoring of LTL specifications. We know of the following two works that use alternating automata for run-time monitoring: in [5] LTL on infinite traces is used for specifications and alternating Büchi automata are constructed for LTL to recognize *finite* traces. The paper presents several algorithms which work on alternating automata to check for word inclusion. In [19] LTL has semantics on *finite* traces and nondeterministic alternating finite automata are used to recognize these traces. A determinization algorithm for alternating automata is given which can be extended to our alternating Büchi automata.

We have taken the approach of giving semantics to $\mathcal{CL}$ on *infinite* traces of actions which is more close to [5] but we want a deterministic finite state machine which at each state checks the finite input trace and outputs an answer telling if the contract has been violated. For this reason we fount the method of [1] most appealing. On the other hand a close look at the semantics of $\mathcal{CL}$ from Section 3 reveals the nice feature of this semantics which behaves the same for finite traces as for infinite traces. This coupled with the definition of alternating automata from Section 4.1 which accepts both infinite and finite traces gives the opportunity to investigate the use of alternating finite automata from [19] on the finite trace semantics. This may generate a monitor which is only single-exponential in size.

References

1. A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *FSTTCS'06*, volume 4337 of *LNCS*, pages 260–272. Springer, 2006.
2. J. Broersen, R. Wieringa, and J.-J. C. Meyer. A fixed-point characterization of a deontic logic of regular action. *Fundam. Inf.*, 48(2-3):107–128, 2001.
3. P. F. Castro and T. S. E. Maibaum. A complete and compact propositional deontic logic. In *ICTAC*, volume 4711 of *LNCS*, pages 109–123. Springer, 2007.
4. A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
5. B. Finkbeiner and H. Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24(2):101–127, 2004.
6. M. J. Fischer and R. E. Ladner. Propositional modal logic of programs. In *9th ACM Symposium on Theory of Computing (STOC'77)*, pages 286–294. ACM, 1977.
7. S. Göller, M. Lohrey, and C. Lutz. PDL with Intersection and Converse Is 2 EXP-Complete. In H. Seidl, editor, *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2007.
8. D. Harel, J. Tiuryn, and D. Kozen. *Dynamic Logic*. MIT Press, 2000.
9. D. Kozen. Automata on guarded strings and applications. In *WoLLIC'01*, volume 24 of *Matematica Contemporanea*. Sociedade Brasileira de Matemática, 2003.
10. O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of ACM*, 47(2):312–360, 2000.
11. Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *POPL'87*, pages 1–12, 1987.
12. D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *LICS*, pages 422–427. IEEE, 1988.
13. O. Owe, G. Schneider, and M. Steffen. Components, objects, and contracts. In *6th Workshop on Specification And Verification of Component-Based Systems (SAVCBS'07)*, ACM Digital Library, pages 91–94, September 2007.
14. V. R. Pratt. Process logic. In *POPL'79*, pages 93–100. ACM, 1979.
15. V. R. Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In *Algebraic Logic and Universal Algebra in Computer Science*, volume 425 of *LNCS*, pages 77–110. Springer-Verlag, 1990.
16. C. Prisacariu and G. Schneider. An Algebraic Structure for the Action-Based Contract Language CL - Theoretical Results. Technical Report 361, Univ. Oslo, 2007.
17. C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.
18. C. Prisacariu and G. Schneider. CL: A Logic for Reasoning about Legal Contracts — Semantics. Technical Report 371, Univ. Oslo, 2008.
19. V. Stolz and E. Bodden. Temporal Assertions Using AspectJ. In *RV'05*, volume 144 of *ENTCS*, pages 109–124. Elsevier, 2006.
20. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, volume 1043 of *LNCS*, pages 238–266. Springer, 1995.
21. M. Y. Vardi. Alternating Automata: Unifying Truth and Validity Checking for Temporal Logics. In *CADE*, volume 1249 of *LNCS*, pages 191–206. Springer, 1997.
22. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE, 1986.
23. G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.

A Proofs

We know [16] that we have an interpretation of actions of $\mathcal{A}$ as trees by an interpretation function $I_{\mathcal{CA}}$. We also know [16] that there is a one-to-one (i.e. a completeness result) relation between the actions and their tree interpretations. Therefore we can consider actions of $\mathcal{A}$ as the associated set of traces of the tree returned by $I_{\mathcal{CA}}$. Recall that we denote this set of traces as $\|I_{\mathcal{CA}}(\alpha)\|$.

We have given the negation of actions as a derived operator $\overline{\ } : \mathcal{A} \to \mathcal{A}$. The negation operator takes an action in canonical form (note that we know [16] that for each action there exists an equivalent canonic form) and returns another action also in canonical form. It may be intuitive that the set of traces associated to an action α is $\|I_{\mathcal{CA}}(\alpha)\|$, but it is not so intuitive what is the set of traces for a negation of an action $\overline{\alpha}$. The following result shows us explicitly the set of traces which correspond to the action negation.

The result below helps in understanding better the semantic definition of $[\overline{\alpha_{\&}}]\mathcal{C}$. It also helps in the proof of the next result in Proposition 4.

Proposition 3 (Characterization of action negation with traces).

The set of traces which are full paths of the tree interpreting the negation of $\alpha = +_{i\in I}\ \alpha^i_{\&} \cdot \alpha^i$ *denoted by* $\|I_{\mathcal{CA}}(\overline{\alpha})\|$ *is equal to the set defined below.*

$$\begin{aligned}\{\overline{\alpha}\} =&\{\sigma \mid\ \sigma = \sigma(0)\varepsilon\ \wedge\ \forall i \in I, \alpha^i_{\&} \not\sqsubseteq \sigma(0)\}\cup\\ &\{\sigma \mid\ \sigma = \sigma(0)\sigma(1..)\ \wedge\ \exists i \in I,\ s.t.\ \alpha^i \neq \mathbf{1} \wedge \alpha^i_{\&} = \sigma(0) \wedge \sigma(1..) \in \{\overline{\alpha^i}\}\}\end{aligned}$$

Proof: The definition of the set of traces is inductive. This is because the definition of the canonical form of actions is inductive and therefore also the action negation. The tree $I_{\mathcal{CA}}(\overline{\alpha})$ is the same as $I_{\mathcal{CA}}(\overline{+_{i\in I}\ \alpha^i_{\&} \cdot \alpha^i})$ which is $I_{\mathcal{CA}}(+_{\gamma\in\overline{R}}\gamma\ +\ +_{i\in I}\ \alpha^i_{\&} \cdot \overline{\alpha^i})$. The first part of the action, i.e. $+_{\gamma\in\overline{R}}\gamma$ gives the full paths of the tree of length 1 (i.e. on the first level). It is simple to observe that these paths are captured by the first set of traces $\{\sigma \mid \sigma = \sigma(0)\varepsilon\ \wedge\ \forall i \in I, \alpha^i_{\&} \not\sqsubseteq \sigma(0)\}$. These are traces with one element $\sigma(0)$ (i.e. ending in the empty trace ε) and they respect the same condition like in the definition of $\overline{R}$. Note that when I is a singleton and we have only one $\alpha_{\&}$ then the condition $\alpha_{\&} \not\sqsubseteq \sigma(0)$ becomes $\sigma(0) = \sigma'(0) \cup \sigma''(0) \wedge \sigma'(0) \subset \alpha_{\&} \wedge \sigma''(0) \subseteq \mathcal{A}^{\&}_B \backslash \alpha_{\&}$. We read this condition as: the first element of σ (which we recall is a set of basic actions) has some actions, i.e. $\sigma'(0)$, among those, but not all, of the basic actions of $\alpha_{\&}$ and the other basic actions, i.e. $\sigma''(0)$ are different than those of $\alpha_{\&}$; i.e. are among $\mathcal{A}^{\&}_B \backslash \alpha_{\&}$.

The other full paths of length greater than 1 of the tree are given by the second part of the action, i.e. $+_{i\in I}\ \alpha^i_{\&} \cdot \overline{\alpha^i}$. All these paths are captured by the second set of traces which are of length at least 2. All branches of $+_{i\in I}\ \alpha^i_{\&} \cdot \overline{\alpha^i}$ where $\alpha^i = \mathbf{1}$ dissapear. This is because when negating $\overline{\alpha^i} = \overline{\mathbf{1}} = \mathbf{0}$ that branch ends in $\mathbf{0}$ which propagates upwards by $\alpha \cdot \mathbf{0} = \mathbf{0}$ and dissapears eventually by $\alpha + \mathbf{0} = \alpha$. Therefore we are looking only at traces s.t. $\alpha^i \neq \mathbf{1}$. From these we take the traces which start with the action $\alpha^i_{\&}$ (i.e. $\alpha^i_{\&} = \sigma(0)$) and are followed by a trace (i.e. $\sigma(1..)$) which is part of the traces of the negation of the smaller compound action α^i. □

Note: We can use an equivalent characterization of the traces of length one where instead of testing for (non)inclusion of sets we can test for nonemptiness of set subtraction; i.e. replace $\alpha^i_{\&} \not\sqsubseteq \sigma(0)$ with $\alpha^i_{\&} \setminus \sigma(0) \neq \emptyset$.

The next result of Proposition 4 is useful in giving us the completeness of the semantical definition for $[\alpha_{\&}]\mathcal{C}$. That is, we get as an immediate corollary that the conditions on traces from the semantic definition of $[\alpha_{\&}]\mathcal{C}$ cover all the possible traces. The same result is used for the semantical definitions of $[\overline{\alpha_{\&}}]\mathcal{C}$, $O_{\mathcal{C}}(\alpha_{\&})$, or $F_{\mathcal{C}}(\alpha_{\&})$. For convenience we define an *enclosing* relation over traces $\sigma \supseteq \sigma'$ iff $\forall i \in \mathbb{N}$, $\sigma'(i) \subseteq \sigma(i)$. Note that this definition requires that $m_\sigma \geq m_{\sigma'}$.

Proposition 4. *Any infinite trace σ is either starting with a trace bigger w.r.t. $\supseteq$ then a complete path of $I_{\mathcal{CA}}(\alpha)$ or it starts with trace bigger than a complete path of $I_{\mathcal{CA}}(\overline{\alpha})$.*

Proof: The proof is by *reductio ad absurdum*. If the trace $\sigma \supseteq \sigma^{I_{\mathcal{CA}}(\alpha)}\sigma'$ starts with a full path of the tree $I_{\mathcal{CA}}(\alpha)$ the proof is finished. Suppose it is not the case that $\sigma \supseteq \sigma^{I_{\mathcal{CA}}(\alpha)}\sigma'$. This means that $\exists i \leq h(I_{\mathcal{CA}}(\alpha))$ s.t. $\sigma(0..i-1) \supseteq \sigma^{I_{\mathcal{CA}}(\alpha)}(0..i-1)$ and for all possible actions $\sigma^{I_{\mathcal{CA}}(\alpha)}(i)$ of extending the trace $\sigma^{I_{\mathcal{CA}}(\alpha)}(0..i-1)$ in the tree $I_{\mathcal{CA}}(\alpha)$ it is the case that $\sigma(i) \not\supseteq \sigma^{I_{\mathcal{CA}}(\alpha)}(i)$. Consider the characterization of the negation of Proposition 3. It is easy to see that the trace $\sigma^{I_{\mathcal{CA}}(\alpha)}(0..i-1)\sigma(i)$ is a full path of the tree $I_{\mathcal{CA}}(\overline{\alpha})$ interpreting the negation of α because the first part is a trace of the action α and the last step of the trace respects the condition in the first set of traces of Proposition 3. More explicitly, in Proposition 3 $\forall i \in I, \alpha^i_{\&}$ means that *for all branches*... each action on the branch must not be less than the current element of the trace. This is the same as the argument needed above.

Considering that $\sigma^{I_{\mathcal{CA}}(\alpha)}(0..i-1)\sigma(i)$ is a full path of $I_{\mathcal{CA}}(\overline{\alpha})$ and that $\sigma(0..i-1) \supseteq \sigma^{I_{\mathcal{CA}}(\alpha)}(0..i-1)$ we finish the proof as the trace σ is starting with the trace $\sigma(0..i) \supseteq \sigma^{I_{\mathcal{CA}}(\overline{\alpha})}$ which is greater than a full path of the tree interpreting the negation of α. □

The semantics of Section 3 is defined s.t. it captures some intuitive properties one finds in legal contracts; we list them in Proposition 5. We say that a formula $\mathcal{C}$ is *valid* and denote it by $\models \mathcal{C}$ iff $\forall \sigma,\ \sigma \models \mathcal{C}$. A formula is not valid, denoted $\not\models \mathcal{C}$ iff $\exists \sigma$ s.t. $\sigma \not\models \mathcal{C}$.

Proposition 5 (properties on traces).

$O_{\mathcal{C}}(a) \wedge O_{\mathcal{C}}(b) \Leftrightarrow O_{\mathcal{C}}(a\&b)$	*(1)*	$F(a) \Rightarrow F(a\&b)$	*(2)*
$\not\models O(a+b) \Rightarrow O(a\&b)$	*(3)*	$F(a+b) \Leftrightarrow F(a) \wedge F(b)$	*(4)*
$\not\models O(a+b) \Rightarrow O(a)$	*(5)*	$F(a \cdot b) \Leftrightarrow F(a) \vee [a]F(b)$	*(6)*
		$\not\models F(a\&b) \Rightarrow F(a)$	*(7)*
$[\alpha_{\&}]\mathcal{C} \Rightarrow [\alpha_{\&}\&\alpha'_{\&}]\mathcal{C}$	*(8)*	$[\beta]\mathcal{C}_1 \wedge [\beta']\mathcal{C}_2 \Rightarrow [\beta\&\beta']\mathcal{C}_1 \wedge \mathcal{C}_2$	*(9)*

Proof: The proofs of these properties is routine. The method is the classical one for validity of implication where we need to look at all and only the models which satisfy the formula on the left of the implication and make sure that they satisfy also the formula on the right.

For property (1): We must prove two implications. We deal first with the $\Rightarrow$ one. Take a trajectory σ s.t. it satisfies the formula on the left, i.e. $\sigma \models O_{\mathcal{C}}(a)$ and $\sigma \models O_{\mathcal{C}}(b)$. We are in the simple case when we consider basic actions a and b. We look at the semantics of obligation. If it is the case that $\sigma(1..) \models \mathcal{C}$ than it is clear that $\sigma \models O(a\&b)$. Otherwise we have the case when both $\sigma(0) \supseteq a$ and $\sigma(0) \supseteq b$. It

implies that $\sigma(0) \supseteq a\&b$ which means that $\sigma \models O(a\&b)$. The second implication $\Leftarrow$ is simpler and uses the same judgement.

Properties (2), (4), or (6) are similar.

For property (3) we need to give a counterexample. Clearly a trace starting with $\sigma(0) = a$ satisfies $O(a+b)$ but does not satisfy $O(a\&b)$. We find similar counterexamples for properties like (5) or (7).

For properties (8) and (9) we again need to show that for all σ which respect the formula on the left of the arrow they also respect the formula on the right. □

$$
\begin{array}{lcl}
FL(\top) & \triangleq & \{\top\} \\
FL(\bot) & \triangleq & \{\bot\} \\
FL(\mathcal{C}_1 \wedge \mathcal{C}_2) & \triangleq & \{\mathcal{C}_1 \wedge \mathcal{C}_2\} \cup FL(\mathcal{C}_1) \cup FL(\mathcal{C}_2) \\
FL(\mathcal{C}_1 \vee \mathcal{C}_2) & \triangleq & \{\mathcal{C}_1 \vee \mathcal{C}_2\} \cup FL(\mathcal{C}_1) \cup FL(\mathcal{C}_2) \\
FL(\mathcal{C}_1 \oplus \mathcal{C}_2) & \triangleq & \{\mathcal{C}_1 \oplus \mathcal{C}_2\} \cup FL(\mathcal{C}_1) \cup FL(\mathcal{C}_2) \\
FL([\beta]\mathcal{C}) & \triangleq & FL^{\Box}([\beta]\mathcal{C}) \cup FL(\mathcal{C}) \\
FL^{\Box}([\beta_{\&}]\mathcal{C}) & \triangleq & \{[\beta_{\&}]\mathcal{C}\} \\
FL^{\Box}([\beta \cdot \beta']\mathcal{C}) & \triangleq & \{[\beta \cdot \beta']\mathcal{C}\} \cup FL^{\Box}([\beta][\beta']\mathcal{C}) \cup FL^{\Box}([\beta']\mathcal{C}) \\
FL^{\Box}([\beta+\beta']\mathcal{C}) & \triangleq & \{[\beta + \beta']\mathcal{C}\} \cup FL^{\Box}([\beta]\mathcal{C}) \cup FL^{\Box}([\beta']\mathcal{C}) \\
FL^{\Box}([\beta^*]\mathcal{C}) & \triangleq & \{[\beta^*]\mathcal{C}\} \cup FL^{\Box}([\beta][\beta^*]\mathcal{C}) \\
FL^{\Box}([\mathcal{C}_1?]\mathcal{C}_2) & \triangleq & \{[\mathcal{C}_1?]\mathcal{C}_2\} \cup FL(\mathcal{C}_1) \\
FL(O_{\mathcal{C}}(\alpha_{\&})) & \triangleq & \{O_{\mathcal{C}}(\alpha_{\&})\} \cup FL(\mathcal{C}) \\
FL(O_{\mathcal{C}}(\alpha \cdot \alpha')) & \triangleq & \{O_{\mathcal{C}}(\alpha \cdot \alpha')\} \cup FL(O_{\mathcal{C}}(\alpha)) \cup FL(O_{\mathcal{C}}(\alpha')) \\
FL(O_{\mathcal{C}}(\alpha+\alpha')) & \triangleq & \{O_{\mathcal{C}}(\alpha+\alpha')\} \cup FL(O_{\bot}(\alpha)) \cup FL(O_{\bot}(\alpha')) \cup FL(\mathcal{C}) \\
FL(P(\alpha)) & \triangleq & \{P(\alpha)\} \\
FL(F_{\mathcal{C}}(\alpha_{\&}) & \triangleq & \{F_{\mathcal{C}}(\alpha_{\&}\} \cup FL(\mathcal{C}) \\
FL(F_{\mathcal{C}}(\alpha \cdot \alpha')) & \triangleq & \{F_{\mathcal{C}}(\alpha \cdot \alpha')\} \cup FL(F_{\bot}(\alpha)) \cup FL(F_{\mathcal{C}}(\alpha')) \\
FL(F_{\mathcal{C}}(\alpha + \alpha')) & \triangleq & \{F_{\mathcal{C}}(\alpha + \alpha')\} \cup FL(F_{\mathcal{C}}(\alpha)) \cup FL(F_{\mathcal{C}}(\alpha'))
\end{array}
$$

Table 5. The complete definition of the Fischer-Ladner closure for $\mathcal{CL}$

One may ponder upon some of the definitions of FL and thus of the transition function of the alternating automaton like e.g. $FL(O_{\mathcal{C}}(\alpha \cdot \alpha'))$ and related $\rho(O_{\mathcal{C}}(\alpha \cdot \alpha'), \gamma) = \rho(O_{\mathcal{C}}(\alpha), \gamma) \wedge O_{\mathcal{C}}(\alpha')$. By looking at the semantics one would choose the transition $\rho([\alpha]O_{\mathcal{C}}(\alpha'), \gamma)$ instead of just $O_{\mathcal{C}}(\alpha')$, and similar for FL. Note that this is not needed, as it is guaranteed by the conjunction and the obligation in $\rho(O_{\mathcal{C}}(\alpha), \gamma)$ that α is to be executed and therefore the box $[\alpha]$ becomes superfluous.

The following result gives the dimension (i.e. cardinality) of the Fischer-Ladner closure $FL(\cdot)$ in terms of the dimension of a formula. It proves to be also linear as in the case of propositional dynamic logic. Naturally, the dimension of a formula (denoted $|\mathcal{C}|$) is the number of symbols it contains; e.g. for an action $|a+b| = 3$ where $a, b \in \mathcal{A}_B$, and for a formula $|O_{\mathcal{C}}(\alpha)| = |\alpha| + |\mathcal{C}|$. We use the same notation for the dimension of the closure $|FL(\mathcal{C})|$.

Theorem 3 (the dimension of the Fischer-Ladner closure).

1. *For any formula $\mathcal{C}$ is the case that $|FL(\mathcal{C})| \leq |\mathcal{C}|$.*

2. *For any formula* $[\beta]\mathcal{C}$ *is the case that* $|FL^{\Box}([\beta]\mathcal{C})| \leq |\beta|$.

Proof: The proof is by using simultaneous induction on the structure of the formula. The proof of 2 is the same as in PDL [8]. For the proof of 1 we need to deal with the special constructions that $\mathcal{CL}$ introduces. We will not treat the Boolean operators $\wedge$, $\vee$, or $\oplus$.

The basic case for $\top$ and $\bot$ is clear:

$$|FL(\bot)| = 1 = |\ \bot\ |$$

From [8] we have:

$$\begin{array}{rcl} |FL([\beta]\mathcal{C})| & \leq & |FL^{\Box}([\beta]\mathcal{C})| + |FL(\mathcal{C})| \\ & \leq & |\beta| + |\mathcal{C}| \text{ by induction hypothesis 1 and 2} \\ & = & |[\beta]\mathcal{C}|. \end{array}$$

The proof for the other $\mathcal{CL}$ constructs is particular to our logic but it is similar to what is done for PDL in [8].

$$\begin{array}{rcl} |FL(O_{\mathcal{C}}(\alpha_{\&}))| & \leq & 1 + |FL(\mathcal{C})| \\ & \leq & 1 + |\mathcal{C}| \text{ by induction hypothesis 1} \\ & = & |\alpha_{\&}| + |\mathcal{C}| = |O_{\mathcal{C}}(\alpha_{\&})|. \end{array}$$

$$\begin{array}{rcl} |FL(O_{\mathcal{C}}(\alpha \cdot \alpha'))| & \leq & 1 + |FL(O_{\mathcal{C}}(\alpha))| + |FL(O_{\mathcal{C}}(\alpha'))| \\ & \leq & 1 + |\alpha| + |\mathcal{C}| + |\alpha'| \text{ by induction hypothesis 1} \\ & = & |O_{\mathcal{C}}(\alpha \cdot \alpha')|. \end{array}$$

Note that the reparation $\mathcal{C}$ is considered only once as when making the union of $FL(O_{\mathcal{C}}(\alpha)) \cup FL(O_{\mathcal{C}}(\alpha'))$ the elements of $FL(\mathcal{C})$ will appear only once. In general the subformulas of the reparation $\mathcal{C}$ will appear only once no matter how we have to decompose the obligations.

$$\begin{array}{rcl} |FL(O_{\mathcal{C}}(\alpha + \alpha'))| & \leq & 1 + |FL(O_{\bot}(\alpha))| + |FL(O_{\bot}(\alpha'))| + |FL(\mathcal{C})| \\ & \leq & 1 + |\alpha| + |\alpha'| + |\mathcal{C}| \text{ by induction hypothesis 1} \\ & = & |O_{\mathcal{C}}(\alpha + \alpha')|. \end{array}$$

Note again that the $FL(\bot)$ is included in $FL(\mathcal{C})$ so it is not considered in the calculation of the dimension of the closure.

The proof for $FL(P(\alpha))$ and $FL(F_{\mathcal{C}}(\alpha_{\&})$ are similar.

$$\begin{array}{rcl} |FL(F_{\mathcal{C}}(\alpha \cdot \alpha'))| & \leq & 1 + |FL(F_{\bot}(\alpha))| + |FL(F_{\mathcal{C}}(\alpha'))| \\ & \leq & 1 + |\alpha| + |\alpha'| + |\mathcal{C}| \text{ by induction hypothesis 1} \\ & = & |F_{\mathcal{C}}(\alpha \cdot \alpha')|. \end{array}$$

$$\begin{array}{rcl} |FL(F_{\mathcal{C}}(\alpha + \alpha'))| & \leq & 1 + |FL(F_{\mathcal{C}}(\alpha))| + |FL(F_{\mathcal{C}}(\alpha'))| \\ & \leq & 1 + |\alpha| + |\alpha'| + |\mathcal{C}| \text{ by induction hypothesis 1} \\ & = & |F_{\mathcal{C}}(\alpha + \alpha')|. \end{array}$$

□

We give here the full proof of the correctness of the automaton construction from **Theorem 2**. Recall that the theorem sais: *Given a $\mathcal{CL}$ formula $\mathcal{C}$, one can build an alternating Büchi automaton $A^{\mathcal{N}}(\mathcal{C})$ which will accept all and only the traces σ respecting the contract formula.*

Proof: Take a formula $\mathcal{C}$ of $\mathcal{CL}$, we construct the alternating Büchi automaton $A^{\mathcal{N}}(\mathcal{C}) = (S, \Sigma, s_0, \rho, F)$ as in Theorem 2. Recall the transition function ρ:

- $\rho(\top, \gamma) = \mathbf{true}$
- $\rho(\bot, \gamma) = \mathbf{false}$
- $\rho(P(\alpha), \gamma) = \mathbf{true}$
- $\rho(\mathcal{C}_1 \wedge \mathcal{C}_2, \gamma) = \rho(\mathcal{C}_1, \gamma) \wedge \rho(\mathcal{C}_2, \gamma)$
- $\rho(\mathcal{C}_1 \vee \mathcal{C}_2, \gamma) = \rho(\mathcal{C}_1, \gamma) \vee \rho(\mathcal{C}_2, \gamma)$
- $\rho(\mathcal{C}_1 \oplus \mathcal{C}_2, \gamma) = (\rho(\mathcal{C}_1, \gamma) \wedge \overline{\rho(\mathcal{C}_2, \gamma)}) \vee (\overline{\rho(\mathcal{C}_1, \gamma)} \wedge \rho(\mathcal{C}_2, \gamma))$
- $\rho(O_{\mathcal{C}}(\alpha_{\&}), \gamma) = \mathbf{true}$ if $\alpha_{\&} \subseteq \gamma$
- $\rho(O_{\mathcal{C}}(\alpha_{\&}), \gamma) = \mathcal{C}$ if $\alpha_{\&} \not\subseteq \gamma$
- $\rho(O_{\mathcal{C}}(\alpha \cdot \alpha'), \gamma) = \rho(O_{\mathcal{C}}(\alpha), \gamma) \wedge O_{\mathcal{C}}(\alpha')$
- $\rho(O_{\mathcal{C}}(\alpha + \alpha'), \gamma) = \rho(O_{\bot}(\alpha), \gamma) \vee \rho(O_{\bot}(\alpha'), \gamma) \vee \mathcal{C}$
- $\rho(F_{\mathcal{C}}(\alpha_{\&}), \gamma) = \mathbf{true}$ if $\alpha_{\&} \not\subseteq \gamma$
- $\rho(F_{\mathcal{C}}(\alpha_{\&}), \gamma) = \mathcal{C}$ if $\alpha_{\&} \subseteq \gamma$
- $\rho(F_{\mathcal{C}}(\alpha \cdot \alpha'), \gamma) = \rho(F_{\bot}(\alpha), \gamma) \vee F_{\mathcal{C}}(\alpha')$
- $\rho(F_{\mathcal{C}}(\alpha + \alpha'), \gamma) = \rho(F_{\mathcal{C}}(\alpha), \gamma) \wedge \rho(F_{\mathcal{C}}(\alpha'), \gamma)$
- $\rho([\alpha_{\&}]\mathcal{C}, \gamma) = \mathcal{C}$ if $\alpha_{\&} \subseteq \gamma$
- $\rho([\alpha_{\&}]\mathcal{C}, \gamma) = \mathbf{true}$ if $\alpha_{\&} \not\subseteq \gamma$
- $\rho([\beta \cdot \beta']\mathcal{C}, \gamma) = \rho([\beta][\beta']\mathcal{C}, \gamma)$
- $\rho([\beta + \beta']\mathcal{C}, \gamma) = \rho([\beta]\mathcal{C}, \gamma) \wedge \rho([\beta']\mathcal{C}, \gamma)$
- $\rho([\beta^*]\mathcal{C}, \gamma) = \rho(\mathcal{C}, \gamma) \wedge \rho([\beta][\beta^*]\mathcal{C}, \gamma)$
- $\rho([\mathcal{C}_1?]\mathcal{C}_2, \gamma) = \overline{\rho(\mathcal{C}_1, \gamma)} \vee (\rho(\mathcal{C}_1, \gamma) \wedge \rho(\mathcal{C}_2, \gamma))$

We complete here the proof for the correctness of the automaton construction we have given.

Soundness: Given an accepting run tree $(T, \mathcal{V})$ of $A^{\mathcal{N}}(\mathcal{C})$ over a trace σ we prove that $\forall x \in T$ a node of the run tree with depth $|x| = i,\ i \geq 0$, labeled by $\mathcal{V}(x) = \mathcal{C}_x$ a state of the automaton represented by a formula $\mathcal{C}_x \in FL(\mathcal{C})$, it is the case that $\sigma(i..) \models \mathcal{C}_x$. This implies that also $\sigma(0..) \models \mathcal{V}(\varepsilon) = \mathcal{C}$, which means that if the automaton $A^{\mathcal{N}}(\mathcal{C})$ accepts a trace σ then the trace respects the initial contract $\mathcal{C}$.

We use induction on the structure of the formula $\mathcal{C}_x$. A formula $\mathcal{C}'$ is said to be a subformula of $\mathcal{C}_x$ iff $\mathcal{C}' \in FL(\mathcal{C}_x)$. The induction method says that we have to prove the property (i.e. the soundness property) for the formula $\mathcal{C}_x$ by having as hypothesis that the property holds for all subformulas of $\mathcal{C}$.

For the truth formula $\top$ it is trivial as $\top$ is respected by any trace and thus by $\sigma(i..)$. For the other nonrecursive formulas the proof is simple by looking at the definition of the respecting relation $\models$ between traces and formulas, and at the construction of the transition relation ρ of the automaton. We take a case for each formula construction:

1. if $\mathcal{C}_x = \mathcal{C}' \wedge \mathcal{C}''$ and we are at depth $|x| = i$ it means that the transition relation is $\rho(\mathcal{C}' \wedge \mathcal{C}'', \sigma(i)) = \rho(\mathcal{C}', \sigma(i)) \wedge \rho(\mathcal{C}'', \sigma(i))$. We should understand the transition relation as follows: because we are in an accepting run tree on σ it means that the automaton from state $\mathcal{C}' \wedge \mathcal{C}''$ accepts $\sigma(i)$ iff the automaton accepts $\sigma(i)$ from both states $\mathcal{C}'$ and $\mathcal{C}''$. We can apply the induction hypothesis on the subformulas $\mathcal{C}'$ and $\mathcal{C}''$ because we know now that there is an accepting run from states $\mathcal{C}'$ and $\mathcal{C}''$ on the remaining trace $\sigma(i..)$. This means that we get both $\sigma(i..) \models \mathcal{C}'$ and $\sigma(i..) \models \mathcal{C}''$ which by the semantics it means that $\sigma(i..) \models \mathcal{C}' \wedge \mathcal{C}''$; i.e. the conclusion.
2. for $\mathcal{C}_x = \mathcal{C}' \vee \mathcal{C}''$ proof is similar as for $\wedge$.
3. if $\mathcal{C}_x = \mathcal{C}' \oplus \mathcal{C}''$ the proof follows the same arguments as before. Consider we are at depth $|x| = i$ it means that the transition relation is $\rho(\mathcal{C}' \oplus \mathcal{C}'', \sigma(i)) = (\rho(\mathcal{C}', \sigma(i)) \wedge \overline{\rho(\mathcal{C}'', \sigma(i))}) \vee (\overline{\rho(\mathcal{C}', \sigma(i))} \wedge \rho(\mathcal{C}'', \sigma(i)))$. The intuition for the transition relation is clear by now; if we know that the automaton accepts $\sigma(i)$ from state $\mathcal{C}' \oplus \mathcal{C}''$ then we know that the automaton accepts $\sigma(i)$ from state $\mathcal{C}'$ but it does not accept $\sigma(i)$ from state $\mathcal{C}''$ (or the other disjunction branch). By inductive reasoning we are ensured that $\mathcal{C}'$ holds in σ from point i on, and we are ensured that $\mathcal{C}''$ fails from this point on. By the semantics of $\oplus$ we get that $\sigma(i..) \models \mathcal{C}' \oplus \mathcal{C}''$. We make the same reasoning for the other disjunction choice.
4. if $\mathcal{C}_x = O_{\mathcal{C}'}(\alpha_{\&})$ and we are at depth $|x| = i$ it means that the transition relation can be of two kinds: first $\rho(O_{\mathcal{C}'}(\alpha_{\&}), \sigma(i)) = \mathbf{true}$ which, because the run is accepting, it means that $\alpha_{\&} \subseteq \sigma(i)$ or $\alpha_{\&} = \sigma(i)$ which from the definition of the respecting relation we conclude that $\sigma(i..) \models O_{\mathcal{C}'}(\alpha_{\&})$. The transition relation can also be $\rho(O_{\mathcal{C}'}(\alpha_{\&}), \sigma(i)) = \mathcal{C}'$ which because the run is accepting and from the induction hypothesis we conclude that $\sigma(i+1..) \models \mathcal{C}'$. By following the semantic definition we conclude that $\sigma(i..) \models O_{\mathcal{C}'}(\alpha_{\&})$.
5. if $\mathcal{C}_x = O_{\mathcal{C}'}(\alpha \cdot \alpha')$ because the run is accepting the rest of the trace $\sigma(i..)$, then by the transition relation it means that the automaton also accepts $\sigma(i..)$ from both the state $O_{\mathcal{C}'}(\alpha)$ and from the state $[\alpha]O_{\mathcal{C}'}(\alpha')$. By the semantics we have that $\sigma(i..) \models O_{\mathcal{C}'}(\alpha \cdot \alpha')$.
6. All remaining cases are similar with the exception of the recursion formula $[\beta^*]\mathcal{C}$. The transition relation says that the automaton must accept $\sigma(i..)$ from state $\mathcal{C}$ and also from the state $[\beta][\beta^*]\mathcal{C}$. Note that the accepting run $(T, \mathcal{V})$ over $\sigma(i..)$ has finite branching whenever $\beta \not\sqsubseteq \sigma(i)$. If $(T, \mathcal{V})$ has an infinite branch than this branch contains $[\beta^*]\mathcal{C}$ infinitely many times. Because of this and the induction hypothesis we have that $\sigma(i..) \models [\beta][\beta^*]\mathcal{C}$ and from the transition relation and the induction hypothesis again we have that $\sigma(i..) \models \mathcal{C}$. Now from the semantics of $[\beta^*]\mathcal{C}$ we finish the proof $\sigma(i..) \models [\beta^*]\mathcal{C}$.

Completeness: Given a trace σ s.t. $\sigma \models \mathcal{C}$ we prove that the constructed automaton $A^{\mathcal{N}}(\mathcal{C})$ accepts σ (i.e. there exists a run tree $(T, \mathcal{V})$ of $A^{\mathcal{N}}(\mathcal{C})$ over the trace σ).

The proof proceeds by constructing a run tree of $\mathcal{A}^{\mathcal{N}}(\mathcal{C})$ which maintains the following invariant: $\forall x \in T$ with $\mathcal{V}(x) = \mathcal{C}_x$ then $\sigma(|x|..) \models \mathcal{C}_x$. The run has to start in the initial state of the automaton so $\mathcal{V}(\varepsilon) = \mathcal{C}$ and since we know from the hypothesis that $\sigma \models \mathcal{C}$ the invariant is satisfied for ε. It is easy to see by the semantics of $\mathcal{CL}$ and by following the definition of ρ over the composing subformulas of $\mathcal{C}$ that the run can

always proceed s.t. for a node x for which the invariant holds the successor nodes all satisfy the invariant.

We give a few examples: Consider the instances of ρ when it ends up in $\mathbf{true}$; like $\rho(O_{\mathcal{C}}(\alpha_{\&}), \sigma(0))$ with $\alpha_{\&} \subseteq \sigma(0)$. It is clear from the semantics that $\sigma \models O_{\mathcal{C}}(\alpha_{\&})$ and the run tree is accepting since it goes into a $\mathbf{true}$ transition. For more complicated formulas like $O_{\mathcal{C}}(\alpha' \cdot \alpha'')$ because of the semantics we have that $\sigma \models O_{\mathcal{C}}(\alpha' \cdot \alpha'')$ implies $\sigma \models O_{\mathcal{C}}(\alpha')$ and $\sigma \models [\alpha']O_{\mathcal{C}}(\alpha'')$. By semantics again, the second relation gives $\sigma(1..) \models O_{\mathcal{C}}(\alpha'')$. The run tree must proceed according to the ρ function and thus it can advance one step into the tree by two means: either by respecting the first obligation and thus one of the successors of ε must be labeled by state $O_{\mathcal{C}}(\alpha'')$, but this satisfies the invariant. The other is by not satisfying the obligation and thus ending up in a state labeled by the reparation $\mathcal{C}$. □