On the Complexity of Partial Order Trace Model Checking*

Thierry Massart Cédric Meuter

Laurent Van Begin†

Université Libre de Bruxelles (U.L.B.),
Boulevard du Triomphe, CP-212, 1050 Bruxelles, Belgium
{tmassart,cmeuter,lvbegin}@ulb.ac.be

Keywords: computational complexity, distributed systems, formal methods

1 Introduction

The design of a distributed system is known to be a difficult task which can be eased by various techniques including validation and debugging. The model-based design (e.g. with UML or other adapted specification languages) abstracts the actions the system can do into *events* which change its *global state*. Depending on the assumptions the designer makes, the model can be either *centralized*, providing a global observation of the entire system or *distributed*, in which case each event is local to some process. Model validation then consists in checking whether the required properties, usually expressed as temporal logic formulae, like CTL*, CTL or LTL [2], are satisfied.

Unfortunately, in practice, this abstraction is generally not sufficient to avoid the *state-explosion problem* which prevents the designer from exhaustively verifying the whole system, even with efficient exploration techniques such as partial order reduction or symbolic model checking [2]. The designer may therefore want to analyse, or validate, simpler models which describe only some *facets* of the system. As such, it may be important, during the *design phase*, to check that *scenarios*, expressed e.g. as *Message Sequence Charts* [5], meet the required properties. Furthermore, during the *testing* and *deployment phases*, *executions* must be validated. *Runtime verification* techniques [4] are typically designed for that purpose. The practical validity of these methods depend on the number of test-cases, to give a *reasonable* confidence that the system is correct. Therefore, theoretical and practical efficiency of the algorithms for validating those models are crucial. In the centralized case, an execution of the system can be seen as a *sequence of events*. The complexity of determining if such an execution satisfies a property has been studied in [7]

*Supported by the Belgian Science Policy IAP-Phase VI: MoVES and Centre Fédéré en Vérification (FNRS-FRFC n 2.4530.02)

†Research fellow supported by the Belgian National Science Foundation (FNRS).

where it is shown that the problem can be solved efficiently. In the distributed case, however, the exact order in which two concurrent events occur in the execution is, in general, not always known or guaranteed. Nevertheless, by taking into account the communications between processes, a partial order on the events of the execution can still be obtained. Hence in this case, an execution can be viewed as a partially ordered set of events called *partial order trace*. The satisfaction of *global properties* on these partial order traces has been widely studied since the 90's. Chase and Garg have shown in [1], that the *global predicate detection problem*, i.e. the reachability of a system's state which satisfies some global predicate, is *NP-complete* for arbitrary predicates, even when there is no inter-process communication. However, various classes of properties can be checked efficiently in polynomial time (see e.g. [3, 6]). Sen and Garg extended the study to temporal operators and defined the RCTL logic [9], a subset of CTL for which the model checking is polynomial on partial order traces. In previous works, we developed symbolic LTL [3] and CTL [6] model checking techniques for partial order traces and showed their efficiency in practice.

In this paper, we focus on the theoretical complexity of CTL*, CTL and LTL model checking over finite partial order traces. We show that over such partial order traces, CTL* and CTL model checking are *PSPACE-complete* and that the LTL model checking is *coNP-complete*.

2 Preliminaries

In this section, we recall the satisfiability problems for propositional and quantified propositional formulae. In the remainder of the paper, we assume an infinite and countable set $\mathbb{P}$ of boolean propositions. Moreover, let $\mathbb{B}$ denote the set of Boolean values, i.e. $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ where tt stands for *true* and ff for *false*.

Propositional Boolean Formulae A *Propositional Boolean Formula* (PBF) φ is defined using the following grammar: $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi$, where $\top$ denotes the *true* formula, and $p \in \mathbb{P}$. Moreover, let $\bot$ denotes the formula $\neg\top$ (the *false* formula). Other standard Boolean operators ($\wedge$, $\Rightarrow$, $\Leftrightarrow$) are derived as usual, and $\Phi \equiv \Psi$ denotes logical equivalence between both formulae. The (finite) set of propositions appearing in a PBF formula φ is denoted by $\mathsf{P}(\varphi)$. A PBF φ is interpreted using a *valuation of* $\mathsf{P}(\varphi)$, i.e. a function $v : \mathsf{P}(\varphi) \mapsto \mathbb{B}$. The satisfaction of a PBF φ by a valuation v, denoted $v \models \varphi$, is defined as usual. The PBF φ is *satisfiable* if there exists a valuation v such that $v \models \varphi$. The *size of the PBF* φ, denoted $|\varphi|$, is defined inductively as 1 for the $\top$ and propositions, and as the sum of the size of the operands plus 1 otherwise. Finally, given a PBF φ, the *PBF-SAT* problem consists in determining if φ is satisfiable. This problem is known to be *NP-complete* [8].

Quantified Boolean Formulae A *Quantified Boolean Formula* (QBF) ψ is a formula of the form $Q_1 p_1 \cdot Q_2 p_2 \cdot \ldots Q_r p_r \cdot \varphi$ where (i) φ is a PBF over $\mathbb{P}$ and (ii) $Q_i \in \{\exists, \forall\}$ and $p_i \in \mathsf{P}(\varphi)$ for $i \in [1, r]$. Note that a PBF is a QBF without quantifiers ($r = 0$). In the following, we assume, without loss of generality, that each proposition is quantified at most once. A *fully QBF* is a QBF where all propositions are quantified. QBF are also interpreted over valuations of $\mathsf{P}(\varphi)$. As in the PBF case, $\mathsf{P}(\psi)$ denotes the set of propositions appearing in the QBF ψ. A valuation $v : \mathsf{P}(\psi) \mapsto \mathbb{B}$ satisfies a QBF ψ is denoted $v \models \psi$. The satisfaction is derived from the propositional case as follows. If $\psi = \forall p \cdot \psi'$, then $v \models \psi$ iff $v[p \mapsto \mathsf{tt}] \models \psi'$ and $v[p \mapsto \mathsf{ff}] \models \psi'$. On the other hand, if $\psi = \exists p \cdot \psi'$, then $v \models \psi$ iff $v[p \mapsto \mathsf{tt}] \models \psi'$ or $v[p \mapsto \mathsf{ff}] \models \psi'$. Note that the truth value of a QBF formula ψ depends only on the valuation of its *free* propositions, i.e. not quantified in ψ. In particular, if ψ is a *fully* QBF, its truth value does not depend on v. Similarly to PBF, a QBF ψ is satisfiable if there exists a valuation v such that $v \models \psi$. The *size of a QBF* $\psi = Q_1 p_1 \cdot Q_2 p_2 \cdot \ldots Q_r p_r \cdot \varphi$ where φ is a PBF, noted $|\psi|$, is equal to $|\varphi|$. Note that the number of quantifiers in ψ is bounded by $|\varphi|$. Given a (fully) QBF ψ, the *QBF-SAT* problem consists in deciding if ψ is satisfiable. This problem is known to be *PSPACE-complete*, even for fully QBF [8].

3 CTL*, CTL and LTL over partial order traces

Partial Order Traces A *partial order trace* (po-trace) is a tuple $\mathcal{T} = \langle E, P_0, \alpha, \beta, \preceq \rangle$ where (i) E is a finite set of events; (ii) $P_0 \subseteq \mathbb{P}$ is the finite set of propositions true at the beginning of the trace; (iii) $\alpha : E \mapsto 2^{\mathbb{P}}$ and $\beta : E \mapsto 2^{\mathbb{P}}$ are two functions modelling the effect of each event e on the propositions of the trace. For an event e, $\alpha(e)$, resp. $\beta(e)$, gives the set of propositions e *changes* to true, resp. false; as a consequence, $\forall e \in E : \alpha(e) \cap \beta(e) = \emptyset$; and (iv) $\preceq \subseteq E \times E$ is a partial order relation on E such that $\forall e, e' \in E$: $((\alpha(e) \cup \beta(e)) \cap (\alpha(e') \cup \beta(e')) \neq \emptyset) \Rightarrow (e \preceq e') \vee (e' \preceq e)$, i.e. if the truth value of at least one proposition is modified by two events, then those events should be ordered. Given an event $e \in E$, we define $\downarrow e = \{e' \in E \mid e' \preceq e\}$, the past of e (including e itself). The finite set of propositions used by $\mathcal{T}$ is denoted by $\mathsf{P}(\mathcal{T})$, i.e $\mathsf{P}(\mathcal{T}) = P_0 \bigcup_{e \in E} \Big(\alpha(e) \cup \beta(e)\Big)$. A *cut* is a subset $C \subseteq E$ with $\forall e \in C : \downarrow e \subseteq C$. The set of cuts is denoted by $\mathsf{cuts}(\mathcal{T})$. Given a cut $C \in \mathsf{cuts}(\mathcal{T})$, we define $\mathsf{enabled}(C) = \{e \in E \setminus C \mid (\downarrow e \setminus \{e\}) \subseteq C\}$ the set of events *enabled* in C, and $C_{/p} = \{e \in C \mid p \in \alpha(e) \cup \beta(e)\}$ the set of events of C that modify the truth value of p. Any event e enabled in the cut C can be fired from C leading to $C' = C \cup \{e\}$, noted $C \triangleright C'$. Note that since $\preceq$ is a partial order, $\mathsf{enabled}(C)$ is not empty for any cut $C \neq E$. Furthermore, the condition imposed on $\preceq$, implies that for every proposition p, the set $C_{/p}$ is totally ordered. Hence, the set of propositions holding in a cut C is independent from the order in which the events of C have been executed. Formally, the set of propositions holding in a cut C, noted P_C is defined as $\{p \in \mathsf{P}(\mathcal{T}) \mid (C_{/p} = \emptyset \wedge p \in P_0) \vee (C_{/p} \neq \emptyset \wedge p \in \alpha(\max(C_{/p}))\}$. Note that for any cut

$C \in \mathsf{cuts}(\mathcal{T})$ we can build P_C in polynomial time as follows: for each proposition p, we first build $C_{/p}$ by enumerating the events $e \in C$ and checking if $p \in \alpha(e) \cup \beta(e)$ (by enumerating $\alpha(e)$ and $\beta(e)$). If $C_{/p}$ is empty then p is in P_C iff $p \in P_0$ (this can be checked by enumerating the elements of P_0). Otherwise, we find the maximal element e_{max} w.r.t $\preceq$ in $C_{/p}$ by enumerating the elements in $C_{/p}$ and $\preceq$. Finally, p is in P_C iff $p \in \alpha(e_{max})$.

A *path* σ is a sequence $\sigma = C_0 \dots C_k \in \mathsf{cuts}(\mathcal{T})^*$ such that $k \geq 0$ and $\forall i \in [0, k) : C_i \triangleright C_{i+1}$. The size $|\sigma|$ of the sequence σ is the number of *firings* from C_0 in σ (i.e. k here)[1]; and we note σ^i the suffix $C_i, C_{i+1}, \dots C_k$. σ^i is left undefined if $i > |\sigma|$. A *run from a cut* C is a path $\sigma = C_0 \dots C_k$ with (i) $C_0 = C$ and (ii) $C_k = E$. The set of runs starting in a cut $C \in \mathsf{cuts}(\mathcal{T})$ is denoted by $\mathsf{runs}(C)$. The *size of the po-trace* $\mathcal{T} = \langle E, P_0, \alpha, \beta, \preceq \rangle$, noted $|\mathcal{T}|$, is equal to $|E| + |\preceq| + |\mathsf{P}(\mathcal{T})|$.

In essence, a po-trace can be interpreted as a Kripke structure, where states are cuts, transitions are events and where each cut C is labelled by the set of propositions holding in C, namely P_C.

CTL* *Formulae* in the temporal logic CTL* are defined using the following grammar:

$$\Psi ::= \top \mid p \mid \neg\Psi \mid \Psi \vee \Psi \mid \exists\Phi \mid \forall\Phi \qquad\qquad \Phi ::= \Psi \mid \neg\Phi \mid \Phi \vee \Phi \mid \Phi\mathcal{U}\Phi \mid \bigcirc\Phi$$

where Ψ is a *state* formula, Φ is a *path* formula, $p \in \mathbb{P}$, $\mathcal{U}$ is the *until* operator and $\bigcirc$ is the *next* operator. Other Boolean constructs ($\bot$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$) are defined as in the PBF case. In our case, CTL* state (resp. path) formulae are interpreted over cuts C (resp. paths σ) of a po-trace $\mathcal{T}$. The satisfaction relation, noted $\models_C$ (resp. $\models_\sigma$) for state (resp. path) formulae, is the relation that satisfies the following:

$\langle \mathcal{T}, C \rangle \models_C \top$

$\langle \mathcal{T}, C \rangle \models_C p$ iff $p \in P_C$

$\langle \mathcal{T}, C \rangle \models_C \neg\Psi$ iff $\langle \mathcal{T}, C \rangle \not\models_C \Psi$

$\langle \mathcal{T}, C \rangle \models_C \Psi_1 \vee \Psi_2$ iff $\langle \mathcal{T}, C \rangle \models_C \Psi_1 \vee \langle \mathcal{T}, C \rangle \models_C \Psi_2$

$\langle \mathcal{T}, C \rangle \models_C \exists\Phi$ iff $\exists\sigma \in \mathsf{runs}(C) : \langle \mathcal{T}, \sigma \rangle \models_\sigma \Phi$

$\langle \mathcal{T}, C \rangle \models_C \forall\Phi$ iff $\forall\sigma \in \mathsf{runs}(C) : \langle \mathcal{T}, \sigma \rangle \models_\sigma \Phi$

$\langle \mathcal{T}, \sigma \rangle \models_\sigma \Psi$ iff $\langle \mathcal{T}, C_0 \rangle \models_C \Psi$

$\langle \mathcal{T}, \sigma \rangle \models_\sigma \neg\Phi$ iff $\langle \mathcal{T}, \sigma \rangle \not\models_\sigma \Phi$

$\langle \mathcal{T}, \sigma \rangle \models_\sigma \Phi_1 \vee \Phi_2$ iff $\langle \mathcal{T}, \sigma \rangle \models_\sigma \Phi_1 \vee \langle \mathcal{T}, \sigma \rangle \models_\sigma \Phi_2$

$\langle \mathcal{T}, \sigma \rangle \models_\sigma \bigcirc\Phi$ iff $|\sigma| > 0 \wedge \langle \mathcal{T}, \sigma^1 \rangle \models_\sigma \Phi$

$\langle \mathcal{T}, \sigma \rangle \models_\sigma \Phi_1\mathcal{U}\Phi_2$ iff $\exists i \in [0, |\sigma|] \ : ((\langle \mathcal{T}, \sigma^i \rangle \models_\sigma \Phi_2) \wedge (\forall j \in [0, i) \ : \langle \mathcal{T}, \sigma^j \rangle \models_\sigma \Phi_1))$

where $\sigma = C_0 \dots C_k$, Φ, Φ_1, Φ_2 are path formulae and Ψ, Ψ_1, Ψ_2 are state formulae.

A po-trace $\mathcal{T}$ satisfies a CTL* state formula Ψ, noted $\mathcal{T} \models \Psi$, iff $\langle \mathcal{T}, \emptyset \rangle \models_C \Psi$. $|\Psi|$, the *size of a* CTL* *formula* Ψ, is defined in the usual way. CTL* has in particular two useful fragments:

Computation Tree Logic (CTL) is a fragment of CTL* in which each $\bigcirc$ and $\mathcal{U}$ operators must

[1]Note that $|\sigma|$ could also have been defined as its number of states (i.e. $k + 1$ here)

be immediately preceded by a path quantifier. Formally, a CTL formula Ψ is defined using the grammar : $\Psi ::= \top \mid p \mid \neg\Psi \mid \Psi \vee \Psi \mid \exists \bigcirc \Psi \mid \forall \bigcirc \Psi \mid \exists[\Psi \mathcal{U} \Psi] \mid \forall[\Psi \mathcal{U} \Psi]$

Linear Time Logic (LTL) is another fragment of CTL* in which each formula has the form $\forall\Phi$ and the only state sub-formulae permitted are $\top$ and atomic propositions $p \in \mathbb{P}$. Formally, a LTL formula Ψ is defined using the grammar : $\Psi ::= \forall\Phi \qquad \Phi ::= \top \mid p \mid \neg\Phi \mid \Phi \vee \Phi \mid \bigcirc\Phi \mid \Phi \mathcal{U} \Phi$.

Given a po-trace $\mathcal{T}$ and a formula Ψ, the *model checking problem* consists in determining if $\mathcal{T} \models \Psi$. In the remainder of the paper, we investigate the complexity of the model checking problem for CTL*, CTL and LTL formulae.

4 CTL* and CTL Model Checking

We start with the model checking problem for CTL* and CTL. First, we will show that for CTL, the problem is PSPACE-hard. Since CTL is a fragment of CTL*, it implies that the problem for CTL* is also PSPACE-hard. Then, we show that, for CTL*, the problem is in PSPACE. Again, since CTL is a fragment of CTL*, it follows that the problem for CTL is also in PSPACE. Those results allow us to conclude that for CTL* and CTL, the model checking problem is PSPACE-complete.

In order to prove that for CTL, the model checking problem is PSPACE-hard, we exhibit a polynomial reduction of (fully) QBF-SAT that works as follows. Let ψ be a fully QBF with $\mathsf{P}(\psi) = \{p_1, \ldots, p_r\}$. We build a po-trace $\mathcal{T}_{\mathsf{P}(\psi)}$ and a CTL formula Ψ_ψ and prove that ψ is satisfiable iff $\mathcal{T}_{\mathsf{P}(\psi)} \models \Psi_\psi$.

The po-trace $\mathcal{T}_{\mathsf{P}(\psi)} = \langle E, P_0, \alpha, \beta, \preceq \rangle$ is built over set of propositions $\bigcup_{i\in[1,r]}\{q_i, q'_i\}$ as follows: (i) $E = \bigcup_{i\in[1,r]}\{e_i, e'_i\}$; (ii) $P_0 = \emptyset$; (iii) for any $i \in [1,r]$: $\alpha(e_i) = \{q_i\}$, $\beta(e_i) = \emptyset$, $\alpha(e'_i) = \{q'_i\}$, and $\beta(e'_i) = \emptyset$ and finally (iv) $\preceq = \emptyset$.

The CTL formula Ψ_ψ is defined inductively as follows:

$$\Psi_\psi = \begin{cases} \psi[p_1 \leftarrow \mathsf{eval}_1^{\mathsf{tt}}, \ldots, p_r \leftarrow \mathsf{eval}_r^{\mathsf{tt}}] & \text{if } \psi \text{ is a PBF} \\ \exists \bigcirc ((\mathsf{eval}_i^{\mathsf{tt}} \vee \mathsf{eval}_i^{\mathsf{ff}}) \wedge \Psi_{\psi_1}) & \text{if } \psi = \exists p_i \cdot \psi_1 \\ \forall \bigcirc ((\mathsf{eval}_i^{\mathsf{tt}} \vee \mathsf{eval}_i^{\mathsf{ff}}) \Rightarrow \Psi_{\psi_1}) & \text{if } \psi = \forall p_i \cdot \psi_1 \end{cases}$$

where $\mathsf{eval}_i^{\mathsf{tt}} = (q_i \wedge \neg q'_i)$, $\mathsf{eval}_i^{\mathsf{ff}} = (\neg q_i \wedge q'_i)$, and where $\psi[p_1 \leftarrow \varphi_1, \ldots, p_r \leftarrow \varphi_r]$ denotes the formula ψ where every occurrence of proposition p_i is replaced by the formula φ_i for $i \in [1,r]$. It is clear that the sizes of Ψ_ψ and $\mathcal{T}_{\mathsf{P}(\psi)}$ are polynomial in the size of ψ. Indeed, each proposition in ψ is replaced by a sub-formula of (constant) size 4 and each quantification is replaced by a construct of (constant) size 12. In the following, for any $i \in [1,r]$, we note $C_i^{\mathsf{tt}} = \{e_i\}$, resp. $C_i^{\mathsf{ff}} = \{e'_i\}$, the minimal cut satisfying $\mathsf{eval}_i^{\mathsf{tt}}$, resp. $\mathsf{eval}_i^{\mathsf{ff}}$. Formally, $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^{\mathsf{tt}} \rangle \models_C \mathsf{eval}_i^{\mathsf{tt}}$ and $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^{\mathsf{ff}} \rangle \models_C \mathsf{eval}_i^{\mathsf{ff}}$ since $P_{C_i^{\mathsf{tt}}} = \{q_i\}$ and $P_{C_i^{\mathsf{ff}}} = \{q'_i\}$. The underlying idea behind the encoding

is the following. A path $C_0C_1 \dots C_k$ of $\mathcal{T}_{|\psi|}$ with $C_0 = \emptyset$ and $\{q_i, q'_i\} \not\subseteq C_j$ for all i, j simulates successive choices for the values of propositions in some order. The PBF nested into the QBF is encoded into CTL to take care of the propositions of the po-trace. The quantifiers are encoded by imposing that the next cut encodes the choice for the value of the proposition linked with that quantifier. Note that in the case of universal quantification, all the choices must be taken into account. However, the enabled events in a cut include events that do not encode the value of the quantified proposition. Hence, when simulating a universal quantification $\forall p_i$ we only consider the successor cuts corresponding to the event q_i or q'_i (formally encoded with $\Rightarrow$).

Lemma 1 *Given a fully QBF ψ, $\mathcal{T}_{\mathsf{P}(\psi)} \models \Psi_\psi$ iff ψ is satisfiable.*

Proof. We prove by induction on $|\mathsf{P}(\psi)|$ that ψ is satisfiable iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \Psi_\psi$.

Base cases. • If $|\mathsf{P}(\psi)| = 0$, then $\mathsf{P}(\psi) = \emptyset$ and ψ is a Boolean combination of $\top$. If $\psi \equiv \top$ (resp. $\psi \equiv \bot$), by definition of Ψ_ψ we have $\Psi_\psi = \psi[p_1 \leftarrow \mathsf{eval}_1^{\mathsf{tt}}, \dots, p_r \leftarrow \mathsf{eval}_r^{\mathsf{tt}}] = \psi \equiv \top$ (resp. $\equiv \bot$). Hence, $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \Psi_\psi$ iff ψ is satisfiable.

Induction cases. We have to consider two cases: • The first case is when $\psi = \exists p_i \cdot \psi_1$. In this case, ψ is satisfiable iff $\psi_1[p_i \leftarrow \top]$ or $\psi_1[p_i \leftarrow \bot]$ is satisfiable. By induction, this is equivalent to $\langle \mathcal{T}_{\mathsf{P}(\psi)\setminus\{p_i\}}, \emptyset \rangle \models_C \Psi_{\psi_1[p_i \leftarrow \top]}$ or $\langle \mathcal{T}_{\mathsf{P}(\psi)\setminus\{p_i\}}, \emptyset \rangle \models_C \Psi_{\psi_1[p_i \leftarrow \bot]}$. By definition of Ψ_{ψ_1}, this holds iff $\langle \mathcal{T}_{\mathsf{P}(\psi)\setminus\{p_i\}}, \emptyset \rangle \models_C \Psi_{\psi_1}[q_i \leftarrow \top, q'_i \leftarrow \bot]$ or $\langle \mathcal{T}_{\mathsf{P}(\psi)\setminus\{p_i\}}, \emptyset \rangle \models_C \Psi_{\psi_1}[q_i \leftarrow \bot, q'_i \leftarrow \top]$, and by definition of C_i^{tt} and C_i^{ff}, iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^{\mathsf{tt}} \rangle \models_C \Psi_{\psi_1}$ or $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^{\mathsf{ff}} \rangle \models_C \Psi_{\psi_1}$. Now since C_i^{tt} (resp. C_i^{ff}) contains the only event that can satisfy $\mathsf{eval}_i^{\mathsf{tt}}$ (resp. $\mathsf{eval}_i^{\mathsf{ff}}$), we deduce that $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^b \rangle \models_C \Psi_{\psi_1}$ iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \exists \bigcirc (\mathsf{eval}_i^b \wedge \Psi_{\psi_1})$ for any $b \in \mathbb{B}$. We can therefore conclude that ψ is satisfiable iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \exists \bigcirc (\mathsf{eval}_i^{\mathsf{tt}} \wedge \Psi_{\psi_1})$ or $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \exists \bigcirc (\mathsf{eval}_i^{\mathsf{ff}} \wedge \Psi_{\psi_1})$, or equivalently if $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C (\exists \bigcirc (\mathsf{eval}_i^{\mathsf{tt}} \wedge \Psi_{\psi_1})) \vee (\exists \bigcirc (\mathsf{eval}_i^{\mathsf{ff}} \wedge \Psi_{\psi_1}))$. Finally, this last formula is equivalent to $\exists \bigcirc ((\mathsf{eval}_i^{\mathsf{tt}} \vee \mathsf{eval}_i^{\mathsf{ff}}) \wedge \Psi_{\psi_1}) = \Psi_\psi$.

• The second case is when $\psi = \forall p_i \cdot \psi_1$. In this case, ψ is satisfiable iff $\psi_1[p_i \leftarrow \top]$ and $\psi_1[p_i \leftarrow \bot]$ are satisfiable. Similarly to the first case, by induction, definition of Ψ_{ψ_1}, C_i^{tt} and C_i^{ff}, this holds iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^{\mathsf{tt}} \rangle \models_C \Psi_{\psi_1}$ and $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^{\mathsf{ff}} \rangle \models_C \Psi_{\psi_1}$. Note that for any cuts C such that $\emptyset \triangleright C$ either $C = C_i^b$ with $b \in \mathbb{B}$ and $C \models \mathsf{eval}_i^b$, or $C \not\models \mathsf{eval}_i^{\mathsf{tt}} \vee \mathsf{eval}_i^{\mathsf{ff}}$. We deduce that $\langle \mathcal{T}_{\mathsf{P}(\psi)}, C_i^b \rangle \models_C \Psi_{\psi_1}$ iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \forall \bigcirc (\mathsf{eval}_i^b \Rightarrow \Psi_{\psi_1})$ for any $b \in \mathbb{B}$. We can therefore conclude that ψ is satisfiable iff $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \forall \bigcirc (\mathsf{eval}_i^{\mathsf{tt}} \Rightarrow \Psi_{\psi_1})$ and $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C \forall \bigcirc (\mathsf{eval}_i^{\mathsf{ff}} \Rightarrow \Psi_{\psi_1})$, or equivalently if $\langle \mathcal{T}_{\mathsf{P}(\psi)}, \emptyset \rangle \models_C (\forall \bigcirc (\mathsf{eval}_i^{\mathsf{tt}} \Rightarrow \Psi_{\psi_1})) \wedge (\forall \bigcirc (\mathsf{eval}_i^{\mathsf{ff}} \Rightarrow \Psi_{\psi_1}))$. Finally, the last formula is equivalent to $\forall \bigcirc ((\mathsf{eval}_i^{\mathsf{ff}} \vee \mathsf{eval}_i^{\mathsf{tt}}) \Rightarrow \Psi_{\psi_1}) = \Psi_\psi$. □

From Lem. 1, we get the PSPACE-hardness for CTL.

Proposition 1 *The model checking problem over po-traces is PSPACE-hard for* CTL.

Proof. Since QBF-SAT is PSPACE-complete, Ψ_ψ and $\mathcal{T}_{\mathsf{P}(\psi)}$ have size polynomial w.r.t. the size of a fully QBF ψ, we conclude by Lem. 1 that the proposition holds. □

Now, we show membership in PSPACE of the model checking problem for CTL* by exhibiting a polynomial space algorithm that solves the problem.

Proposition 2 *The model checking problem over po-traces is in PSPACE for* CTL*

Proof. First, we exhibit a recursive algorithm that takes a partial order trace $\mathcal{T} = \langle E, P_0, \alpha, \beta, \preceq \rangle$, a CTL* state formula Ψ with a cut C (resp. a CTL* trace formula Ψ with a run $\sigma = C_0 C_1 \dots C_k$ with $C = C_0$), and returns *true* iff $\langle \mathcal{T}, C \rangle \models_C \Psi$ (resp. $\langle \mathcal{T}, \sigma \rangle \models_\sigma \Psi$). The recursion follows the structure of the inference which shows $\langle \mathcal{T}, C \rangle \models_C \Psi$ or $\langle \mathcal{T}, \sigma \rangle \models_\sigma \Psi$. Then, we show inductively on the depth of the recursion, that this algorithm is polynomial space w.r.t. the size of the formula Ψ and the size of the po-trace $\mathcal{T}$.

Base cases. • When $\Psi = \top$ or $\Psi = p$. In the first case, the algorithm always returns *true*. In the second case, if Ψ is a state formula then it builds P_C and returns *true* iff $p \in P_C$.

Induction cases If Ψ contains sub-formulae, then the algorithm works as follows:

• First, if Ψ is evaluated on a trace $\sigma = C_0 \dots C_k$ but it is not of the form $\Psi_1 \vee \Psi_2$, $\neg\Psi_1$, $\bigcirc\Psi_1$ or $\Psi_1 \mathcal{U} \Psi_2$, then Ψ is also a state formula and the algorithm returns *true* iff $\langle \mathcal{T}, C_0 \rangle \models_C \Psi$.

• If $\Psi = \neg\Psi_1$ or $\Psi = \Psi_1 \vee \Psi_2$, then Ψ_1 and Ψ_2 are first evaluated and then the algorithm evaluates Ψ according to the usual semantics of boolean connectors.

• If $\Psi = Q\Psi_1$ with $Q \in \{\exists, \forall\}$, then Ψ is a state formula. In this case, the algorithm enumerates all the runs $\sigma \in \mathsf{runs}(C)$ as follows: from C' initially equal to C, we enumerate the events $e \in E$ and then test if $e \in \mathsf{enabled}(C')$ and if there is no $(e', e) \in \preceq$ such that $e' \notin C'$ by enumerating the elements of $\preceq$ and C'. If it is the case, we iterate from the cut $C' \cup \{e\}$ until we build E. At each step, the algorithm only keeps in memory the cuts of the current investigated run. Since the size of the runs are bounded by $|E|$, hence bounded by $|\mathcal{T}|$, the number of cuts stored in memory when enumerating all the runs $\sigma \in \mathsf{runs}(C)$ is bounded by $|\mathcal{T}|$. For each run $\sigma \in \mathsf{runs}(C)$, the algorithms checks if $\langle \mathcal{T}, \sigma \rangle \models_\sigma \Psi_1$ holds. In the case where $Q = \exists$ (resp. $Q = \forall$), the algorithm returns *true* iff for at least one run (resp. all the runs) $\sigma \in \mathsf{runs}(C) : \langle \mathcal{T}, \sigma \rangle \models_\sigma \Psi_1$.

• if $\Psi = \bigcirc\Psi_1$ then Ψ is a path formula and the algorithm returns *true* iff $\langle \mathcal{T}, \sigma^1 \rangle \models_\sigma \Psi_1$ holds.

• if $\Psi = \Psi_1 \,\mathcal{U}\, \Psi_2$, then Ψ is a path formula. For each $0 \leq i \leq k$, the algorithm first considers the sub-formula Ψ_2 and checks if $\langle \mathcal{T}, \sigma^i \rangle \models_\sigma \Psi_2$ holds. If it is the case, the algorithm considers Ψ_1 and checks if $\langle \mathcal{T}, \sigma^j \rangle \models_\sigma \Psi_1$ holds for all $0 \leq j < i$. In the case of a positive answer for all $0 \leq j < i$, the algorithm returns *true*. Finally, if the algorithm does not conclude for any $0 \leq i \leq k$ then it returns *false*.

Let us now show that the algorithm uses only a polynomial space w.r.t. the size of the formula Ψ and the size of the partial order trace $\mathcal{T}$. To simplify the presentation, we do not care about the memory used to store one cut, $P_C, \dots$. However, it is immediate that the memory used can be bounded by a polynomial in the size of $\mathcal{T}$ by using, for instance, bit vectors to represent sets.

More precisely, we show that the number of cuts that are computed and stored at the same time into memory by the algorithm is bounded by $|\mathcal{T}| \cdot |\Psi|$. The proof is by induction on the depth of the recursion of the algorithm.

Base cases • If $\Psi = \top$, then the algorithm returns *true* in constant time without building cuts, hence the result.

• If $\Psi = p$ then Ψ is evaluated on the cut C and the algorithm builds P_C. As previously mentioned, this can be achieved in polynomial time without building new cuts. Since $0 \leq |\mathcal{T}| \cdot |\Psi|$, we conclude.

Induction cases • If $\Psi = \neg\Psi_1$ or $\Psi = \Psi_1 \vee \Psi_2$ then by induction hypothesis we know that the algorithm evaluates Ψ_1 and Ψ_2 (on runs or cuts) storing at most $|\mathcal{T}| \cdot (|\Psi| - 1)$ cuts (since $|\Psi_i| < |\Psi|$ for $i \in \{1, 2\}$), hence it evaluates Ψ by storing at most $|\mathcal{T}| \cdot (|\Psi| - 1) \leq |\mathcal{T}| \cdot |\Psi|$ cuts.

• If $\Psi = Q\Psi_1$ with $Q \in \{\exists, \forall\}$, then the algorithm enumerates all the runs $\sigma \in \mathsf{runs}(C)$. As previously explained, the enumeration of the runs $\sigma \in \mathsf{runs}(C)$ is done by keeping in memory a number of cuts bounded by $|\mathcal{T}|$. Then, by induction hypothesis, the algorithm uses memory bounded by $|\mathcal{T}| \cdot |\Psi_1|$ to check if $\langle \mathcal{T}, \sigma \rangle \models_\sigma \Psi_1$ holds. Since $|\Psi_1| + 1 = |\Psi|$, we conclude that the algorithm maintains at most $|\mathcal{T}| \cdot |\Psi|$ cuts in memory when evaluating Ψ.

• If $\Psi = \bigcirc\Psi_1$, the algorithm evaluates Ψ_1 on a trace. By induction hypothesis, this is achieved by storing at most $|\Psi_1| \cdot |\mathcal{T}|$ cuts. Furthermore, the trace over which Ψ_1 is evaluated has size bounded by $|\mathcal{T}|$. Since $|\Psi| = |\Psi_1| + 1$, we conclude that the algorithm stores at most $|\Psi| \cdot |\mathcal{T}|$ cuts.

• If $\Psi = \Psi_1 \,\mathcal{U}\, \Psi_2$, then assume that Ψ is evaluated on the run σ. By induction hypothesis, the number of cuts stored in memory when evaluating Ψ_1, resp. Ψ_2, on sub-sequences of σ is bounded by $|\mathcal{T}| \cdot |\Psi_1|$, resp. $|\mathcal{T}| \cdot |\Psi_2|$. Furthermore, Ψ_1 and Ψ_2 are evaluated on traces with size bounded by $|\mathcal{T}|$. Since $|\Psi_1| < |\Psi|$ and $|\Psi_2| < |\Psi|$, we conclude that the number of cuts stored in memory by the algorithm when checking Ψ is bounded by $|\Psi| \cdot |\mathcal{T}|$.

• Finally, in the case where the algorithm has to evaluate on a trace $\sigma = C_0 \dots C_k$ a formula Ψ that is not of the form $\neg\Psi_1$, $\Psi_1 \vee \Psi_2$, $\bigcirc\Psi_1$ or $\Psi_1 \mathcal{U} \Psi_2$, then the algorithm evaluates Ψ on C_0 as explained above and we directly conclude from the previous cases. □

From Propositions 1 and 2, and since CTL is a subset of CTL*, we conclude that :

Theorem 1 *The model checking problem over po-trace is PSPACE-complete for* CTL* *and* CTL.

Remark 1 *The reduction to prove PSPACE-hardness only uses modalities $\exists\bigcirc$ and $\forall\bigcirc$. Hence, Theorem 1 can be naturally extended to the branching time logic with only $\exists\bigcirc$ and $\forall\bigcirc$ as modal operators.*

5 LTL Model Checking

We now prove that the model checking problem for the linear-time temporal logic LTL is coNP-complete on po-traces. For that purpose, we examine the dual problem of model checking $\text{LTL}^{\exists}$ formulae, i.e. formulae of the form $\exists\Phi$ where Φ is a restricted path formula, as defined in the grammar of LTL. We first show that this problem is in NP.

Proposition 3 *The model checking over po-traces is in NP for* $\text{LTL}^{\exists}$

Proof. We exhibit a non-deterministic polynomial time algorithm. The algorithm works as follows: it first guesses a run σ of the po-trace $\mathcal{T}$, and then checks that the formula holds on that run. The algorithm starts from $C' = \emptyset$, and for each cut of the run it guesses an event e, checks in polynomial time that $e \in \mathsf{enabled}(C')$ as explained in Proposition 2 and then builds the next cut $C' \cup \{e\}$. Note that the size of a run in $\mathsf{runs}(\emptyset)$ is $|E|$.

Finally, since the set P_C of propositions that holds in a cut C can be computed in polynomial time, LTL model-checking on a run can be solved in polynomial time [7, Proposition 3.3]. □

Next, we show that the model checking problem is NP-hard for $\text{LTL}^{\exists}$. For that, we reduce the global predicate detection problem which is known to be NP-complete [1]. In our framework, this problem can be stated as follows. Given a po-trace $\mathcal{T}$ and a PBF φ over $\mathsf{P}(\mathcal{T})$, the global predicate detection problem consists in determining if $\exists C \in \mathsf{cuts}(\mathcal{T}) : \langle \mathcal{T}, C\rangle \models_C \varphi$.

Proposition 4 *The model checking problem over po-traces is NP-hard for* $\text{LTL}^{\exists}$.

Proof. Let us first note that for every cut $C \in \mathsf{cuts}(\mathcal{T})$ there exists a path $C_0 \dots C_\ell$ with $C_0 = \emptyset$ and $C_\ell = C$. Indeed, it is obviously true for the initial cut $\emptyset$. If we assume that the property holds for every cut containing $k \geq 0$ events, then, since $\preceq$ is a partial order, for any cut C that contains $k+1$ events we can find an event $e \in C$ such that there is no $e' \in C$ with $e' \neq e$ and $e \in \downarrow(e')$. Hence, $C \setminus \{e\}$ is a cut which contains k events, hence it is reachable from $\emptyset$. Furthermore, $\downarrow(e) \subseteq C \setminus \{e\}$, hence $C \setminus \{e\} \triangleright C$, and we conclude that there is a path $C_0 \dots C_\ell$ with $C_0 = \emptyset$ and $C_\ell = C$. Then, given a po-trace $\mathcal{T}$, and a PBF φ, it is immediate that $\exists C \in \mathsf{cuts}(\mathcal{T}) : \langle \mathcal{T}, C\rangle \models_C \varphi$ iff $\mathcal{T} \models \exists(\top \mathcal{U} \varphi)$. □

We can therefore conclude that, similarly to the global predicate detection problem [1], the model checking problem is NP-complete for $\text{LTL}^{\exists}$ and therefore coNP-complete for LTL, as stated in the following theorem.

Theorem 2 *The model checking problem over po-traces is coNP-complete for* LTL.

Hence, Thm 1 and Thm 2 show that the model checking problem, contrarily to complete systems, i.e. arbitrary Kripke structures [2], over po-traces, the LTL model checking problem has a lower complexity than the CTL one.

References

[1] C. M. Chase, V. K. Garg, Detection of global predicates: Techniques and their limitations, Distributed Computing 11 (4) (1998) 191–201.

[2] E. Clarke, O. Grumberg, D. Peled, Model Checking, The MIT Press, 1999.

[3] A. Genon, T. Massart, C. Meuter, Monitoring Distributed Controllers: When an Efficient LTL Algorithm on Sequences Is Needed to Model-Check Traces., in: J. Misra, T. Nipkow, E. Sekerinski (eds.), FM, vol. 4085 of LNCS, Springer, 2006.

[4] K. Havelund, Using runtime analysis to guide model checking of Java programs., in: K. Havelund, J. Penix, W. Visser (eds.), SPIN, vol. 1885 of LNCS, Springer, 2000.

[5] ITU-TS, ITU-TS Recommendation Z.120: Message Sequence Chart (MSC), Geneva (1999).

[6] G. Kalyon, T. Massart, C. Meuter, L. V. Begin, Testing distributed systems through symbolic model checking., in: J. Derrick, J. Vain (eds.), FORTE, vol. 4574 of LNCS, Springer, 2007.

[7] N. Markey, P. Schnoebelen, Model checking a path., in: R. M. Amadio, D. Lugiez (eds.), CONCUR, vol. 2761 of LNCS, Springer, 2003.

[8] C. H. Papadimitriou, Computational Complexity, Addison Wesley Longman, 1994.

[9] A. Sen, V. K. Garg, Detecting temporal logic predicates in distributed programs using computation slicing., in: J. Misra, T. Nipkow, E. Sekerinski (eds.), OPODIS, vol. 3144 of LNCS, Springer, 2003.