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 $\text{Ctl}^*$, $\text{Ctl}$ or $\text{Ltl}$ [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 effi-
cient 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 scenar-
ios, 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 a arbitrary predicates, even when there is no inter-process communications. 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 CCTL∗, CCTL and LTL model checking over finite partial order traces. We show that over such partial order traces, CCTL∗ and CCTL 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 \( P \) of boolean propositions. Moreover, let \( B \) denote the set of Boolean values, i.e. \( B = \{ \text{tt}, \text{ff} \} \) where \( \text{tt} \) stands for true and \( \text{ff} \) for false.

**Propositional Boolean Formulae** A Propositional Boolean Formula (PBF) \( \varphi \) is defined using the following grammar: \( \varphi ::= \top | p | \neg \varphi | \varphi \lor \varphi \), where \( \top \) denotes the true formula, and \( p \in P \). Moreover, let \( \bot \) denotes the formula \( \neg \top \) (the false formula). Other standard Boolean operators (\( \land, \Rightarrow, \Leftrightarrow \)) are derived as usual, and \( \Phi \equiv \Psi \) denotes logical equivalence between both formulae. The (finite) set of propositions appearing in a PBF formula \( \varphi \) is denoted by \( P(\varphi) \). A PBF \( \varphi \) is interpreted using a valuation of \( P(\varphi) \), i.e. a function \( v : P(\varphi) \rightarrow B \). The satisfaction of a PBF \( \varphi \) by a valuation \( v \), denoted \( v \models \varphi \), is defined as usual. The PBF \( \varphi \) is satisfiable if there exists a valuation \( v \) such that \( v \models \varphi \). The size of the PBF \( \varphi \), 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 \( \varphi \), the PBF-SAT problem consists in determining if \( \varphi \) is satisfiable. This problem is known to be NP-complete [8].
Quantified Boolean Formulae  A Quantified Boolean Formula (QBF) $\psi$ is a formula of the form $Q_1p_1 \cdot Q_2p_2 \cdot \ldots Q_rp_r \cdot \varphi$ where (i) $\varphi$ is a PBF over $P$ and (ii) $Q_i \in \{\exists, \forall\}$ and $p_i \in P(\varphi)$ 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 $P(\varphi)$. As in the PBF case, $P(\psi)$ denotes the set of propositions appearing in the QBF $\psi$. A valuation $v : P(\psi) \mapsto B$ satisfies a QBF $\psi$ 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 $v[p \mapsto tt] \models \psi'$ and $v[p \mapsto ff] \models \psi'$. On the other hand, if $\psi = \exists p \cdot \psi'$, then $v \models \psi$ iff $v[p \mapsto tt] \models \psi'$ or $v[p \mapsto ff] \models \psi'$. Note that the truth value of a QBF formula $\psi$ depends only on the valuation of its free propositions, i.e. not quantified in $\psi$. In particular, if $\psi$ is a fully QBF, its truth value does not depend on $v$. Similarly to PBF, a QBF $\psi$ is satisfiable if there exists a valuation $v$ such that $v \models \psi$. The size of a QBF $\psi = Q_1p_1 \cdot Q_2p_2 \cdot \ldots Q_rp_r \cdot \varphi$ where $\varphi$ is a PBF, noted $|\psi|$, is equal to $|\varphi|$. Note that the number of quantifiers in $\psi$ is bounded by $|\varphi|$. Given a (fully) QBF $\psi$, the QBF-SAT problem consists in deciding if $\psi$ 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 $T = (E, P_0, \alpha, \beta, \preceq)$ where (i) $E$ is a finite set of events; (ii) $P_0 \subseteq P$ is the finite set of propositions true at the beginning of the trace; (iii) $\alpha : E \mapsto 2^P$ and $\beta : E \mapsto 2^P$ 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 $\forall e, e' \in E : ((\alpha(e) \cup \beta(e)) \cap (\alpha(e') \cup \beta(e')) \neq \emptyset) \Rightarrow (e \preceq e' \lor e' \preceq e)$, 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 $|e| = \{e' \in E | e' \preceq e\}$, the past of $e$ (including $e$ itself).

The finite set of propositions used by $T$ is denoted by $P(T)$, i.e $P(T) = P_0 \cup_{e \in E} \left(\alpha(e) \cup \beta(e)\right)$. A cut is a subset $C \subseteq E$ with $\forall e \in C : |e| \subseteq C$. The set of cuts is denoted by $cuts(T)$. Given a cut $C \in cuts(T)$, we define enabled$(C) = \{e \in E \setminus C | (\{e\} \cup \{e\}) \subseteq C\}$ the set of events enabled in $C$, and $C/p = \{e \in C | p \in \alpha(e) \cup \beta(e)\}$ the set of events of $C$ that modifies the truth value of $p$. Note that the condition imposed on $\preceq$, implies that for every proposition $p$, the set $C/p$ is totally ordered. Furthermore, since $\preceq$ is a partial order, enabled$(C)$ is not empty for any cut $C \not\in E$.

Hence, the set of propositions holding in a cut $C$ is independant from the order in which the events from $C$ have been executed. Formally, the set of propositions holding in a cut $C$, noted $P_C$ is defined as $\{p \in P(T) | (C/p = \emptyset \lor p \in P_0) \lor (C/p \neq \emptyset \land p \in \alpha(max(C/p)))\}$. Any event $e$ enabled in the cut $C$ can be fired from $C$ leading to $C' = C \cup \{e\}$, noted $C \triangleright e$. A path $\sigma$ is a sequence
\[ \sigma = C_0 \ldots C_k \in \text{cuts}(T)^* \text{ such that } k \geq 0 \text{ and } \forall i \in [0,k) : C_i \triangleright C_{i+1}. \] The size \(|\sigma|\) of the sequence \(\sigma\) is the number of firings from \(C_0\) in \(\sigma\) (i.e. \(k\) here)\(^1\); and we note \(\sigma^i\) the suffix \(C_i, C_{i+1}, \ldots C_k\). \(\sigma^i\) is left undefined if \(i > |\sigma|\). A \textit{run} from a cut \(C\) is a path \(\sigma = C_0 \ldots C_k\) with (i) \(C_0 = C\) and (ii) \(C_k = E\). The set of runs starting in a cut \(C \in \text{cuts}(T)\) is denoted by \(\text{runs}(C)\). The \textit{size of the po-trace} \(T = (E, P, A, \beta, \preceq)\), noted \(|T|\), is equal to \(|E| + |\preceq| + |P(T)|\).

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 \(P_C\).

\[ \text{CTL}^* \quad \text{Formulae in the temporal logic } \text{CTL}^* \text{ are defined using the following grammar:} \]
\[ \Psi ::= \top \mid p \mid \neg \Psi \mid \Psi \lor \Psi \mid \exists \Phi \mid \forall \Phi \quad \Phi ::= \Psi \mid \neg \Phi \mid \Phi \lor \Phi \mid \Phi \cup \Phi \mid \Phi = \Phi \]
where \(\Psi\) is a \textit{state} formula, \(\Phi\) is a \textit{path} formula, \(p \in P\), \(U\) is the \textit{until} operator and \(\bigcirc\) is the \textit{next} operator. Other Boolean constructs \((\perp, \land, \rightarrow, \leftrightarrow)\) are defined as in the PBF case. In our case, \(\text{CTL}^*\) state (resp. path) formulae are interpreted over cuts \(C\) (resp. paths \(\sigma\)) of a po-trace \(T\). The satisfaction relation, noted \(\models_C \) (resp. \(\models_{\sigma}\)) for state (resp. path) formulae, is the smallest relation that satisfies the following:

\[(T, C) \models_C \top \]
\[(T, C) \models_C \neg \Psi \quad \text{iff} \quad \langle T, C \rangle \not\models_C \Psi \]
\[(T, C) \models_C \Psi_1 \lor \Psi_2 \quad \text{iff} \quad \langle T, C \rangle \models_C \Psi_1 \lor \langle T, C \rangle \models_C \Psi_2 \]
\[(T, C) \models_C \exists \Phi \quad \text{iff} \quad \exists \sigma \in \text{runs}(C) : \langle T, \sigma \rangle \models_{\sigma} \Phi \]
\[(T, C) \models_C \forall \Phi \quad \text{iff} \quad \forall \sigma \in \text{runs}(C) : (T, \sigma) \models_{\sigma} \Phi \]
\[(T, \sigma) \models_{\sigma} \Psi \quad \text{iff} \quad \langle T, C_0 \rangle \models_C \Psi \]
\[(T, \sigma) \models_{\sigma} \neg \Phi \quad \text{iff} \quad \langle T, \sigma \rangle \not\models_{\sigma} \Phi \]
\[(T, \sigma) \models_{\sigma} \Phi_1 \lor \Phi_2 \quad \text{iff} \quad \langle T, \sigma \rangle \models_{\sigma} \Phi_1 \lor \langle T, \sigma \rangle \models_{\sigma} \Phi_2 \]
\[(T, \sigma) \models_{\sigma} \bigcirc \Phi \quad \text{iff} \quad |\sigma| > 0 \land \langle T, \sigma^1 \rangle \models_{\sigma} \Phi \]
\[(T, \sigma) \models_{\sigma} \Phi_1 U \Phi_2 \quad \text{iff} \quad \exists i \in [0,|\sigma|] : (((T, \sigma^i) \models_{\sigma} \Phi_2) \land (\forall j \in [0,i) : \langle T, \sigma^j \rangle \models_{\sigma} \Phi_1)) \]

where \(\sigma = C_0 \ldots C_k\), \(\Phi, \Phi_1, \Phi_2\) are path formulae and \(\Psi, \Psi_1, \Psi_2\) are state formulae.

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

\textbf{Computation Tree Logic (CTL)} is a fragment of \text{CTL}^* in which each \(\bigcirc\) and \(U\) operators must be immediately preceded by a path quantifier. Formally, a \text{CTL} formula \(\Psi\) is defined using the grammar:

\[ \Psi ::= \top \mid p \mid \neg \Psi \mid \Psi \lor \Psi \mid \exists \bigcirc \Psi \mid \forall \bigcirc \Psi \mid \exists [\Psi U \Psi] \mid \forall [\Psi U \Psi] \]

\textbf{Linear Time Logic (LTL)} is another fragment of \text{CTL}^* in which each formula has the form \(\forall \Phi\) and the only state sub-formulae permitted are \(\top\) and atomic propositions \(p \in P\). Formally, a \text{LTL} formula \(\Psi\) is defined using the grammar:

\[ \Psi ::= \forall \Phi \quad \Phi ::= \top \mid p \mid \neg \Phi \mid \Phi \lor \Phi \mid \bigcirc \Phi \mid \Phi U \Phi \]

\(^1\)Note that \(|\sigma|\) could also have been defined as its number of states (i.e. \(k + 1\) here)
Given a po-trace $T$ and a formula $\Psi$, the model checking problem consists in determining if $T \models \Psi$. In the remainder of the paper, we investigate the complexity of the model checking problem for $\text{CTL}^*$, $\text{CTL}$ and $\text{LTL}$ formulae.

4 $\text{CTL}^*$ and $\text{CTL}$ Model Checking

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

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

The po-trace $T_{P(\psi)} = \langle E, P_0, \alpha, \beta, \preceq \rangle$ 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 $\text{CTL}$ formula $\Psi_\psi$ is defined inductively as follows:

$$
\Psi_\psi = \begin{cases} 
\psi[p_1 \leftarrow \text{eval}^{\Pi}_1, \ldots, p_r \leftarrow \text{eval}^{\Pi}_r] & \text{if } \psi \text{ is a PBF} \\
\exists \circ (((\text{eval}^{\Pi} \lor \text{eval}^{\Pi}_1) \land \Psi_{\psi_1}) & \text{if } \psi = \exists p_i \cdot \psi_1 \\
\forall \circ (((\text{eval}^{\Pi} \lor \text{eval}^{\Pi}_1) \Rightarrow \Psi_{\psi_1}) & \text{if } \psi = \forall p_i \cdot \psi_1
\end{cases}
$$

where $\text{eval}^{\Pi} = \{q_i \land \neg q'_i\}$, $\text{eval}^{\Pi}_1 = \{\neg q_i \land q'_i\}$, and where $\psi[p_1 \leftarrow \varphi_1, \ldots, p_r \leftarrow \varphi_r]$ denotes the formula $\psi$ where every occurrence of proposition $p_i$ is replaced by the formula $\varphi_i$ for $i \in [1, r]$. It is clear that the sizes of $\Psi_\psi$ and $T_{P(\psi)}$ are polynomial in the size of $\psi$. Indeed, each proposition in $\psi$ 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 $C^\Pi_i = \{e_i\}$, resp. $C^{\Pi}_i = \{e'_i\}$, the minimal cut satisfying $\text{eval}^{\Pi}_i$, resp. $\text{eval}^{\Pi}_i$. Formally, $\langle T_{P(\psi)}, C^\Pi_i \rangle \models C^\Pi_i$ and $\langle T_{P(\psi)}, C^{\Pi}_i \rangle \models C^{\Pi}_i$ since $P_{C^\Pi_i} = \{q_i\}$ and $P_{C^{\Pi}_i} = \{q'_i\}$. The underlying idea behind the encoding is the following. A path $C_0C_1 \ldots C_k$ of $T_{P(\psi)}$ 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 $\text{CTL}$ 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 \( \psi \), \( \mathcal{T}_{P(\psi)} \models \Psi \psi \) iff \( \psi \) is satisfiable.

**Proof.** We prove by induction on \( |P(\psi)| \) that \( \psi \) is satisfiable iff \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \Psi \psi \).

**Base cases.** If \( |P(\psi)| = 0 \), then \( P(\psi) = \emptyset \) and \( \psi \) is a Boolean combination of \( \top \). If \( \psi \equiv \top \) (resp. \( \psi \equiv \bot \)), by definition of \( \Psi \psi \) we have \( \Psi \psi = \psi[p_1 \leftarrow \text{eval}_{p_1}^\Psi, \ldots, p_r \leftarrow \text{eval}_{p_r}^\Psi] = \psi \equiv \top \) (resp. \( \equiv \bot \)). Hence, \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \Psi \psi \) iff \( \psi \) 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, \( \psi \) is satisfiable iff \( \psi_1[p_i \leftarrow \top] \) or \( \psi_1[p_i \leftarrow \bot] \) is satisfiable. By induction, this is equivalent to \( \langle \mathcal{T}_{P(\psi)} \setminus \{p_i\}, \emptyset \rangle \models C \Psi_{\psi_1[p_i \leftarrow \top]} \) or \( \langle \mathcal{T}_{P(\psi)} \setminus \{p_i\}, \emptyset \rangle \models C \Psi_{\psi_1[p_i \leftarrow \bot]} \). By definition of \( \Psi_{\psi_1} \), this holds if \( \langle \mathcal{T}_{P(\psi)} \setminus \{p_i\}, \emptyset \rangle \models C \Psi_{\psi_1}[q_i \leftarrow \bot] \) or \( \langle \mathcal{T}_{P(\psi)} \setminus \{p_i\}, \emptyset \rangle \models C \Psi_{\psi_1}[q_i \leftarrow \top] \), and by definition of \( C_{\psi_1}^\Psi \) and \( C_{\psi_1}^\exists \), iff \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\Psi \rangle \models C \Psi_{\psi_1} \) or \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\exists \rangle \models C \Psi_{\psi_1} \). Now since \( C_{\psi_1}^\Psi \) (resp. \( C_{\psi_1}^\exists \)) contains the only event that can satisfy \( \text{eval}_{p_i}^\Psi \) (resp. \( \text{eval}_{p_i}^\exists \)), we deduce that \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\Psi \rangle \models C \Psi_{\psi_1} \) iff \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\exists \rangle \models C \Psi_{\psi_1} \). Similar to the first case, we can therefore conclude that \( \psi \) is satisfiable iff \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \exists \bigcirc (\text{eval}_{p_i}^\Psi \land \Psi_{\psi_1}) \) or \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \exists \bigcirc (\text{eval}_{p_i}^\exists \land \Psi_{\psi_1}) \), or equivalently if \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \bigcirc (\exists \bigcirc (\text{eval}_{p_i}^\Psi \land \Psi_{\psi_1}) \lor (\exists \bigcirc (\text{eval}_{p_i}^\exists \land \Psi_{\psi_1})) \). Finally, this last formula is equivalent to \( \exists \bigcirc ((\text{eval}_{p_i}^\Psi \lor \text{eval}_{p_i}^\exists) \land \Psi_{\psi_1}) = \Psi \psi \).

• The second case is when \( \psi = \forall p_i \cdot \psi_1 \).

In this case, \( \psi \) 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 \( \Psi_{\psi_1} \), \( C_{\psi_1}^\Psi \) and \( C_{\psi_1}^\forall \), this holds if \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\Psi \rangle \models C \Psi_{\psi_1} \) and \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\forall \rangle \models C \Psi_{\psi_1} \). Note that for any cuts \( C \) such that \( \emptyset \triangleright C \) either \( C = C_i^b \) with \( b \in \mathbb{B} \) and \( C \models \text{eval}_{p_i}^b \), or \( C \not\models \text{eval}_{p_i}^b \lor \text{eval}_{p_i}^\forall \). We deduce that \( \langle \mathcal{T}_{P(\psi)}, C_{\psi_1}^\Psi \rangle \models C \Psi_{\psi_1} \) iff \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \forall \bigcirc (\text{eval}_{p_i}^\Psi \Rightarrow \Psi_{\psi_1}) \) for any \( b \in \mathbb{B} \). We can therefore conclude that \( \psi \) is satisfiable iff \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \forall \bigcirc (\text{eval}_{p_i}^\Psi \Rightarrow \Psi_{\psi_1}) \) and \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \forall \bigcirc (\text{eval}_{p_i}^\exists \Rightarrow \Psi_{\psi_1}) \), or equivalently if \( \langle \mathcal{T}_{P(\psi)}, \emptyset \rangle \models C \bigcirc (\forall \bigcirc (\text{eval}_{p_i}^\Psi \Rightarrow \Psi_{\psi_1}) \lor (\forall \bigcirc (\text{eval}_{p_i}^\exists \Rightarrow \Psi_{\psi_1})) \). Finally, the last formula is equivalent to \( \forall \bigcirc ((\text{eval}_{p_i}^\Psi \lor \text{eval}_{p_i}^\exists) \Rightarrow \Psi_{\psi_1}) = \Psi_{\psi_1} \).

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

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

**Proof.** Since QBF-SAT is PSPACE-complete, \( \Psi \psi \) and \( \mathcal{T}_{P(\psi)} \) have size polynomial w.r.t. the size of a fully QBF \( \psi \), 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 \textit{po-traces} is in PSPACE for CTL*
Proof. First, we exhibit a recursive algorithm that takes a partial order trace $T = \langle E, P_0, \alpha, \beta, \preceq \rangle$, a CTL* state formula $\Psi$ with a cut $C$ (resp. a CTL* trace formula $\Psi$ with a run $\sigma = C_0 C_1 \ldots C_k$ with $C = C_0$), and returns true iff $\langle T, C \rangle \models C \Psi$ (resp. $\langle T, \sigma \rangle \models C \Psi$). The recursion follows the structure of the inference which shows $\langle T, C \rangle \models C \Psi$ or $\langle T, \sigma \rangle \models C \Psi$. Then, we show inductively on the depth of the recursion, that this algorithm is polynomial space w.r.t. the size of the formula $\Psi$ and the size of the po-trace $T$.

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

Induction cases If $\Psi$ contains sub-formulae, then the algorithm works as follows:

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

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

• If $\Psi = Q \Psi_1$ with $Q \in \{\exists, \forall\}$, then $\Psi$ is a state formula. In this case, the algorithm enumerates all the runs $\sigma \in \text{runs}(C)$ as follows: from $C'$ initially equal to $C$, we enumerate the events $e \in E$ and then test if $e \in \text{enabled}(C')$ 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 $|T|$, the number of cuts stored in memory when enumerating all the runs $\sigma \in \text{runs}(C)$ is bounded by $|T|$. For each run $\sigma \in \text{runs}(C)$, the algorithms checks if $\langle T, \sigma \rangle \models C \Psi_1$ holds. In the case where $Q = \exists$ (resp. $Q = \forall$), the algorithm returns true iff for at least one run (resp. all the runs) $\sigma \in \text{runs}(C): \langle T, \sigma \rangle \models C \Psi_1$.

• if $\Psi = \bigcirc \Psi_1$ then $\Psi$ is a path formula and the algorithm returns true iff $\langle T, C_1 \ldots C_k \rangle \models C \Psi_1$ holds.

• if $\Psi = \Psi_1 \mathcal{U} \Psi_2$, then $\Psi$ is a path formula. For each $0 \leq i \leq k$, the algorithm first considers the sub-formula $\Psi_2$ and checks if $\langle T, C_1 \ldots C_k \rangle \models \Psi_2$ holds. If it is the case, the algorithm considers $\Psi_1$ and checks if $\langle T, C_{j} \ldots C_k \rangle \models C \Psi_1$ 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 $\Psi$ and the size of the partial order trace $T$. To simplify the presentation, we do not care about the memory used to store one cut, $P_C$, ... However, it is immediate that the memory used can be bounded by a polynomial in the size of $T$ 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 $|T| \cdot |\Psi|$. 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 $\Psi$ is evaluated on the cut $C$ and the algorithm builds $P_C$. This can be achieved in polynomial time without building new cuts by first computing for each $p \in P(T)$ the set $C_p$ by enumerating the events $e \in C$ and checking if $p \in \alpha(e) \cup \beta(e)$ (this can also be done by enumerating $\alpha$ and $\beta$). 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 $e_{\text{max}}$ w.r.t $\preceq$ in $C_p$, by enumerating the elements in $C_p$ and $\preceq$. Finally, $p$ is in $P_C$ iff $p \in \alpha(e_{\text{max}})$. Since $0 \leq |T| \cdot |\Psi|$, we conclude.

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

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

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

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

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

From Propositions 1 and 2, and since $\text{CTL}^*$ is a subset of $\text{CTL}^*$, we conclude that :

**Theorem 1** The model checking problem over po-trace is PSPACE-complete for $\text{CTL}^*$ and $\text{CTL}^*$.

**Remark 1** The reduction to prove PSPACE-hardness only uses modalities $\exists \bigotimes$ and $\forall \bigotimes$. Hence, Theorem 1 can be naturally extended to the branching time logic with only $\exists \bigotimes$ and $\forall \bigotimes$ 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 p-o-traces. For that purpose, we examine the dual problem of model checking LTL$^3$ formulae, i.e. formulae of the form $\exists \Phi$ where $\Phi$ 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 p-o-traces is in NP for LTL$^3$.

**Proof.** We exhibit a non-deterministic polynomial time algorithm. The algorithm works as follows: it first guesses a run $\sigma$ of the p-o-trace $T$, 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 $e \in \text{enabled}(C')$ as explained in proposition 2 and then builds the next cut $C' \cup \{e\}$. Note that the size of a run in runs($\emptyset$) has size $|E|$. Finally, 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 LTL$^3$. For that, we reduce the global predicate detection which is known to be NP-complete [1]. In our framework, this problem can be stated as follows. Given a p-o-trace $T$ and a PBF $\varphi$ over $P(T)$, the global predicate detection consists in determining if $\exists C \in \text{cuts}(T) : \langle T, C \rangle \models_C \varphi$.

**Proposition 4** The model checking problem over p-o-traces is NP-hard for LTL$^3$.

**Proof.** Let us first note that for every cut $C \in \text{cuts}(T)$ there exists a path $C_0 \ldots C_{\ell}$ 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 $C_0 \ldots C_{\ell}$ with $C_0 = \emptyset$ and $C_{\ell} = C$. Then, given a p-o-trace $T$, and a PBF $\varphi$, it is immediate that $\exists C \in \text{cuts}(T) : \langle T, C \rangle \models_C \varphi$ iff $T \models \exists(\top \cup \varphi)$. □

We can therefore conclude that, similarly to the global predicate detection problem [1], the model checking problem is NP-complete for LTL$^3$ and therefore coNP-complete for LTL, as stated in the following theorem.

**Theorem 2** The model checking problem over p-o-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 p-o-traces, the LTL model checking problem has a lower complexity than the CTL one.
References


