Model Checking a Path (Preliminary Report)
Nicolas Markey, Philippe Schnoebelen

To cite this version:
Nicolas Markey, Philippe Schnoebelen. Model Checking a Path (Preliminary Report). Proceedings of the 14th International Conference on Concurrency Theory (CONCUR’03), 2003, Marseilles, France. pp.251-265, 10.1007/b11938. hal-01194626

HAL Id: hal-01194626
https://hal.archives-ouvertes.fr/hal-01194626
Submitted on 7 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract. We consider the problem of checking whether a finite (or ultimately periodic) run satisfies a temporal logic formula. This problem is at the heart of “runtime verification” but it also appears in many other situations. By considering several extended temporal logics, we show that the problem of model checking a path can usually be solved efficiently, and profit from specialized algorithms. We further show it is possible to efficiently check paths given in compressed form.

1 Introduction

Model checking, introduced in the early 80’s, has now become a widely used approach to the verification of all kinds of systems [CGP99,BBF+01]. The name “model checking” covers a variety of techniques dealing with various subproblems: how to model systems by some kind of Kripke structures?, how to express properties in temporal logics or some other formalisms?, how to use symbolic techniques for dealing with large state spaces?, and, most importantly, how to algorithmically check that a model satisfies a property?

These techniques rest upon a solid body of foundational knowledge regarding the expressive power of temporal logics and the computational complexity of their model checking problems [Sch03].

In this paper, we consider the problem of model checking a single path. This problem appears in several situations, most notably in runtime verification [Dru00,Hav00,FS01]. There are situations where thousands of paths are checked one by one, e.g. the Monte-Carlo approach for assessing the probability that a random run satisfies some property [YS02,LP02]. Less standard situations exist: [RG01] advocates using temporal logic for describing patterns of intrusive behaviors recorded in log files. Such a log file, where a series of system events are recorded, is just a long path on which the temporal formula will be evaluated.

We do not restrict to finite paths and also consider checking ultimately periodic paths (given as finite “lasso-shaped” loop). Checking a path is much simpler
than checking a Kripke structure, so much so that the problem may appear triv-
ial: using standard dynamic programming methods “à la CTL model checking”,
a path can obviously be checked in bilinear, i.e. $O(|model| \times |formula|)$, time.

This may explain why the problem, while ubiquitous, has not been isolated
and studied from a theoretical viewpoint. For example, it is not known whether
checking a simple temporal formula over a finite path can be done more efficiently
than with the “bilinear time” method, e.g. with memory-efficient algorithms in
SC, or with fast parallel algorithms in NC. Indeed this open problem was only
identified recently [DS02].

With this paper, we aim to show that the problem is worthy of more funda-
mental investigations. Of course, the problem is a generic one, with many
variants (which temporal logic? what kind of paths?) and here we only start
scratching its surface.

More specifically, we present results (some of them folklore) showing that

**Checking a path is easier:** As we show in this paper, model checking a path
is often much easier than checking a Kripke structure. We exhibit examples
of richly expressive temporal logics that allow polynomial-time algorithms
for checking a single path, while checking all paths of a Kripke structure
is highly untractable. It is even possible to achieve polynomial-time when
checking compressed paths (i.e. exponentially long paths that are given and
stored in compressed form).

**Checking a path relies on specific techniques:** These efficient algorithms
rely on specific aspects of the problem. Checking a path definitely comes with
its own set of notions, technical tricks, and conceptual tools. For example,
all our algorithms for checking ultimately periodic paths rely on a specific
reduction technique to checking some kind of short finite prefix of the infinite
path.

Outline of the paper. We define the basic problem of model checking $LTL$
formulæ over finite or ultimately periodic paths (section 2). This problem is still not
satisfactorily solved, but we argue that its intrinsic difficulty is already present
in the case of finite paths (section 3). We then show that model checking a path
is much easier than model checking a Kripke structure by looking at various
rich temporal logics: the monadic first-order logic of order, the extension of $LTL$
with Chop, or the extension of $LTL$ with forgettable past (section 4). We pro-
vide polynomial-time algorithms for the last two instances. Finally we look at
the problem of checking paths given in compressed form (section 5).

Related works. Model checking a path is a central problem in runtime verifica-
tion. In this area, the problem is seen through some specific practical applica-
tions, sometimes with an emphasis on online algorithms, with the result that the
fundamental complexity analysis has not received enough attention.

Dynamic programming algorithms for checking finite and ultimately periodic
paths are also used in bounded model checking [BCC+03]. In this area, the rel-
evant measures for efficiency are not the classical notions of running time and
memory space, but have more to do with, say, the number of Boolean variables introduced by the algorithm, or the pattern of dependencies between them.

Model checking a path has a lot in common with algorithmics on words (witness Section 5). However, our concern with temporal logics and ultimately periodic paths is not standard in that other area.

2 Linear-time temporal logic and paths

We assume familiarity with temporal logics (mainly LTL) and model checking: see [Eme90,CGP99,BBF+01].

Syntax of LTL + Past. Let \( AP = \{ p_0, p_1, p_2, \ldots \} \) be a countably infinite set of atomic propositions. The formulae of LTL + Past, are given by the following grammar:

\[
\varphi, \psi ::= \neg \varphi \mid \varphi \land \psi \mid X \varphi \mid X^-1 \varphi \mid \varphi U \psi \mid \varphi S \psi \mid p_0 \mid p_1 \mid p_2 \mid \cdots
\]

\( S \) (Since) and \( X^-1 \) (Previously) are the past-time mirrors of the well-known \( U \) (Until) and \( X \) (Next). We shall freely use the standard abbreviations \( \top, \varphi \Rightarrow \psi, \varphi \lor \psi, F \varphi (\overset{def}{=} \top U \varphi), G \varphi (\overset{def}{=} \neg F \neg \varphi), F^-1 \varphi (\overset{def}{=} \top S \varphi) \) and \( G^-1 \varphi (\overset{def}{=} \neg F^-1 \neg \varphi) \).

LTL, the well-known propositional linear-time temporal logic, is the fragment where \( S \) and \( X^-1 \) are not used, also called the pure-future fragment. While LTL + Past is not more expressive than LTL [GPSS80,Rab02], and not harder to verify [SC85], it can be (at least) exponentially more succinct than LTL [LMS02].

Semantics. Linear-time formulae are evaluated along paths. Formally, a path is a sequence \( \pi = s_0, s_1, \ldots \), finite or infinite, of states, where a state is a valuation \( s \in 2^{AP} \) of the atomic propositions. \( |\pi| \in \mathbb{N} \cup \{ \omega \} \) denotes the length of \( \pi \) and, for a position \( i < |\pi| \), one defines when a formula holds at position \( i \) of \( \pi = (s_l)_{l < |\pi|} \) by induction on the structure of formulae:

\[
\begin{align*}
\pi, i \models p & \iff p \in s_i & \text{for } p \in AP \\
\pi, i \models X \varphi & \iff \pi, i+1 \models \varphi & \text{(hence } i + 1 < |\pi|) \\
\pi, i \models X^-1 \varphi & \iff \pi, i-1 \models \varphi & \text{(hence } i > 0) \\
\pi, i \models \varphi U \psi & \iff \exists j \geq i : \left( \forall i \leq k < j : \pi, k \models \varphi \right) & \text{(hence } j < |\pi|) \\
\pi, i \models \varphi S \psi & \iff \exists j \leq i : \left( \forall j < k \leq i : \pi, k \models \varphi \right) & \text{(hence } j \geq 0)
\end{align*}
\]

omitting the usual clauses for negation and conjunction. We say a non-empty path \( \pi \) satisfies \( \varphi \), written \( \pi \models \varphi \), when \( \pi, 0 \models \varphi \), i.e. when \( \varphi \) holds at the beginning of \( \pi \).

Since only propositions that appear in \( \varphi \) are relevant for deciding whether \( \pi \models \varphi \), we usually assume that paths only carry valuations for the finite number of propositions that will be used later on them.
Model checking. We are interested in the computational problem of model checking a path against an LTL formula. This requires that the path argument be given in some finite way. In classical model checking, where we evaluate a temporal formula along all the paths of a finite Kripke structure (KS), the KS is the finite input that describes an infinite set of infinite paths. Here we assume that the given path is finite, or is ultimately periodic.

Ultimately periodic, or u.p., paths, are given via a pair \((u, v)\) of two finite paths, called a loop for short. A loop \((u, v)\) denotes the infinite path \(\pi = u.v^\omega\), called its unfolding, where an initial \(u\) prefix is followed by repeated copies of \(v\). For uniformity, we shall assume finite paths are given via loops too, only they have empty \(v\). We say loop \((u, v)\) has type \((m, p)\) when \(m\) is the length \(|u|\) of \(u\) and \(p\) is the length of \(v\).

Model checking a path. The generic computational problem we are considering is:

\[\text{PMC}(L) \quad (\text{Path Model Checking for } L)\]

**Input:** two finite paths \(u, v\) and a temporal formula \(\varphi\) of \(L\).

**Output:** yes iff \(u.v^\omega \models \varphi\), no otherwise.

Here \(L\) can be any temporal logic (but it is not meaningful to consider branching-time logics). We shall consider several problems: \(\text{PMC}(LTL)\), \(\text{PMC}(LTL + Past)\), etc. We denote by \(\text{PMC}_f(L)\) the restricted problem when only finite paths are considered (i.e., when \(v = \varepsilon\)). A recurring pattern in our results is that \(\text{PMC}(L)\) reduces to \(\text{PMC}_f(L)\) (by default, we consider logspace reductions).

### 3 How efficient can path model checking be?

We mentioned in the introduction that the following holds:

**Theorem 3.1.** \(\text{PMC}(LTL)\) can be solved in time \(O(|uv| \times |\varphi|)\).

**Proof.** Obvious since, over paths, CTL and LTL coincide. So that the well-known bilinear algorithm for CTL model checking can be used. \(\square\)

That polynomial-time algorithms also exist for \(LTL + Past\) is less obvious:

**Theorem 3.2.** \(\text{PMC}(LTL + Past)\) can be solved in time \(O(|uv| \times |\varphi|^2)\).

This can be obtained as a corollary of Theorem 4.5 but it is instructive to look at a direct proof, since it illustrates a recurring pattern.

We start with the simpler case where the path is finite:

**Proposition 3.3.** \(\text{PMC}_f(LTL + Past)\) can be solved in time \(O(|u| \times |\varphi|)\).

**Proof (Sketch).** The obvious dynamic programming algorithm works: starting from the innermost subformulae, we recursively fill a Boolean table \(T[i, \psi]\), where \(i\) is a position in the finite path, and \(\psi\) is a subformula of \(\varphi\), in such a way that \(T[i, \psi] = \top\) iff \(\pi, i \models \psi\). \(\square\)
This algorithm is too naive for u.p. paths: one cannot label uniformly states inside the loop. A state in the loop corresponds to different positions in the unfolded path, and these positions have different pasts.

However it is only necessary to unfold the loop a small number of times, something we present as a reduction from \textbf{PMC}(LTL + Past) to \textbf{PMC}(LTL + Past).

Let $\varphi$ be an LTL + Past formula, and $(u, v)$ a loop of type $(m, p)$. In the sequel, we write $h_F(\varphi)$ for the future temporal height of $\varphi$, i.e. its maximum number of nested future-time modalities. Similarly, $h_P(\varphi)$ denotes the past temporal height of $\varphi$. We write $H(\varphi)$ for $h_F(\varphi) + h_P(\varphi)$. E.g., for $\varphi = \text{FF}^{-1}XFp_1 \lor \text{F}^{-1}G^{-1}p_2$, we have $h_F(\varphi) = 3$, $h_P(\varphi) = 2$ and $H(\varphi) = 5$.

Lemma 3.4 ([Mar02]). For all subformulae $\psi$ of $\varphi$, and $k \geq m + h_P(\varphi)p$

\begin{equation}
  u.v^\omega, k \models \psi \iff u.v^\omega, k + p \models \psi.
\end{equation}

This may be proved by structural induction on formula $\psi$.

We now reduce model checking of $\varphi$ on the loop $(u, v)$ to a finite path model checking problem. We assume that $p \neq 0$ (otherwise the result is obvious), and build the finite path $\pi' = uv^{-1}H(\varphi)+1$ where $v'$ is like $v$ except that $v'_0$ carries a new proposition $q \notin AP$ (and we replace $AP$ by $AP' = AP \cup \{q\}$). Formally, $v'$ is given by:

$|v'| = p \quad v'_0 \overset{\text{def}}{=} v_0 \cup \{q\} \quad v'_i \overset{\text{def}}{=} v_i \quad \text{for } i > 0.
$

We also recursively build a set of formulae $\chi_k$ as follows:

$\chi_0 \overset{\text{def}}{=} \top \quad \chi_k \overset{\text{def}}{=} F(q \land X\chi_{k-1}).$

Obviously, $\pi', i \models \chi_k$ iff $i \leq m + (H(\varphi) - h_F(\psi))p$. Now $\overline{\varphi}$ is inductively given by:

$\frac{\varphi = p}{\overline{\varphi} = \neg \varphi} \quad \frac{\psi_1 \lor \psi_2 = \overline{\psi_1} \lor \overline{\psi_2}}{\overline{\psi_1} \cup \overline{\psi_2} = \overline{\psi_1 \cup \overline{\psi_2} \land \chi_{h_F(\psi_1 \cup \psi_2)}}} \quad \frac{\psi_1 S \psi_2 = \psi_1 S \psi_2}{\psi_1 S \psi_2 = \psi_1 S \psi_2}
$

Lemma 3.5. For all subformulae $\psi$ of $\varphi$, for all $i < m + (H(\varphi) - h_F(\psi))p$, we have:

$u.v^\omega, i \models \psi \iff \pi', i \models \overline{\psi}.
$

A direct corollary is the reduction we announced:

Theorem 3.6. For any LTL + Past formula $\varphi$, and loop $(u, v)$, one can build in logspace a formula $\overline{\varphi}$ and a finite path $\pi'$ s.t.

$u.v^\omega \models \varphi \iff \pi' \models \overline{\varphi}.
$

Since $|\pi'|$ is in $O(|uv||\varphi|)$, we obtain Theorem 3.2 by combining with Proposition 3.3.
An open question and a conjecture. We do not know whether the upper bounds given in Theorems 3.1 and 3.2 are tight. There is an obvious NC<sub>1</sub> lower bound that has nothing to do with model checking; evaluating a Boolean expression is NC<sub>1</sub>-hard. But the gap between NC<sub>1</sub> and PTIME is (assumed to be) quite large.

We have been unable to prove even LOGSPACE-hardness for PMC(LTL + Past), or to find an algorithm for PMC(LTL) (even for PMC(L(F))) that would be memory efficient (e.g. requiring only polylog-space) or that would be considered as an efficient parallel algorithm (e.g. in NC).

We consider that the open question of assessing the precise complexity of PMC(LTL) and PMC(LTL + Past) is one of the important open problems in model checking [DS02, section 4]. In view of how Theorem 3.6 reduces PMC(...)... to PMC(...), one thing we can tell about the open problem is that the difficulty does not come from allowing u.p. paths.

Our conjecture is that PMC(LTL) is not PTIME-hard. This conviction is grounded in our experience with all the PTIME-hardness proofs we can obtain for richer logics (see next sections) and the way they always exploit some powerful trick or other that LTL and LTL + Past do not allow.

4 Richly expressive temporal logics

Many temporal logics are more expressive, or more succinctly expressive, than LTL [Eme90, Rab02]. However this increased expressivity usually comes with an increased cost for verification, which explains why they are not so popular in the model checking community.

In this section we consider a few such temporal logics. Since we focus on logics with first-order definable modalities, the “rich expressiveness” should be understood as “succinct expressiveness”.

Our first example is FOMLO, the first-order logic of order with monadic predicates. This formalism is not really a modal logic, like LTL is, but it is fundamental since it encompasses all natural temporal logics. We show that model checking FOMLO on paths is PSPACE-complete, hence is much easier on paths than on Kripke structures (where it is nonelementary [Sto74]).

We then look at two more specific extensions of LTL. LTL + Chop has a non-elementary model checking problem on Kripke structures [RP86], but we show it leads to a PTIME-complete path model checking problem. Another PTIME-complete problem on paths appears with LTL + Past + Now, an extension of LTL + Past that has an EXPSPACE-complete model checking problem on Kripke structures [LMS02].

Fig. 1 summarizes our results. The obvious conclusion is that, when model checking paths, there is no reason to restrict oneself to LTL: much more expressive formalisms can be handled at (more or less) no additional price.

4.1 FOMLO, the first-order monadic logic of order

We will not recall here all the basic definitions and notations for FOMLO. Let us simply say that we use \( qh(\varphi) \) to denote the quantifier-height of \( \varphi \), that we write
Fig. 1. Checking richly expressive logics on paths
For example, assuming $m = 10$ and $p = 3$, the formula

$$\exists x \forall y (x > y \land p_0(y) \Rightarrow \exists z (y < z < x \land p_1(z)))$$  \quad (\varphi)$$

is relativized as

$$\exists x < 22 \forall y < 28 (x > y \land p_0(y) \Rightarrow \exists z < 31 (y < z < x \land p_1(z))).$$  \quad (\tilde{\varphi})$$

A corollary of Lemma 4.3 is the following

**Lemma 4.4.** $u.v^\omega \models \varphi$ iff $u.v^\omega \models \tilde{\varphi}$.

We can now evaluate whether $u.v^\omega \models \varphi$ in polynomial-space, proving Proposition 4.2. Lemma 4.4 reduces this question to a bounded problem, where only a finite prefix of $u.v^\omega$ has to be examined. That prefix still has exponential size $O(m + p2^{q_\mathit{h}(\varphi)})$ but we do not have to build it. Rather, we only go over all values for the position variables in $\tilde{\varphi}$, storing them in binary notation (say) to ensure polynomial-space. Then it is easy to evaluate the predicates on these binary notations: the only dyadic predicate is $<$, and the monadic predicates reduces to simple arithmetical computations to find a congruent position in $u.v$.

### 4.2 LTL with forgettable past

"LTL with forgettable past" is $LTL + \mathit{Past} + \mathit{Now}$, i.e. $LTL + \mathit{Past}$ where we add a new unary modality $N$ (for "from Now on"). The semantics of $N$ is given by

$$\pi, i \models N \varphi \iff \pi \geq i, 0 \models \varphi.$$  \quad (\tilde{\varphi})$$

We refer to [LMS02] for motivations on $LTL + \mathit{Past} + \mathit{Now}$: that logic can be exponentially more succinct than $LTL + \mathit{Past}$, and its model checking problem is EXPSPACE-complete. $LTL + \mathit{Past} + \mathit{Now}$ is included in Fig. 1 because Theorem 4.5 was the first hint that $\mathit{PMC}$ allows dealing efficiently with rich logics.

**Theorem 4.5 ([LMS02]).** $\mathit{PMC}(LTL + \mathit{Past} + \mathit{Now})$ is PTIME-complete.

### 4.3 The Chop modality

"Chop", introduced in [HKP80] and studied in [RP86], is a two-place modality whose semantics is defined as follows:

$$\pi, 0 \models \varphi \land \psi \iff \exists k \geq 0 \text{ s.t. } \pi_{\geq k+1} \models \psi \text{ and } \pi_{\leq k} \models \varphi$$

where $\pi_{\geq k+1}$ is the suffix of $\pi$ starting at (and including) position $k + 1$, and $\pi_{\leq k}$ is the prefix of $\pi$ up to (and including) position $k$. It is useful in cases we want to see subruns inside a run (e.g. sessions, or specific segments) and state their temporal specifications [RP86].

Satisfiability for $LTL + \mathit{Chop}$ is non elementary [RP86]. Hence model checking $LTL + \mathit{Chop}$ properties on Kripke structures is non elementary too (there exists
a polynomial-space reduction from satisfiability to model checking, see [DS02, Prop. 3.1]).

However, model checking a path is easier:

**Proposition 4.6.** \( \text{PMC}(\text{LTL} + \text{Chop}) \) can be solved in time \( O(|uv|^2 \times |\varphi|^3) \).

The outline of the proof is similar to the case of \( \text{LTL} + \text{Past} \). First, we observe that, for a finite path \( \pi \), the following holds:

**Proposition 4.7.** \( \text{PMC} f(\text{LTL} + \text{Chop}) \) can be solved in time \( O(|\pi|^2 |\varphi|) \).

**Proof (Sketch).** Again dynamic programming techniques suffice. We fill a Boolean table \( T[i,j,\psi] \), for each positions \( i \leq j \) in \( \pi \), and subformula \( \psi \) of \( \varphi \), in such a way that

\[
T[i,j,\psi] = \top \text{ iff } \pi[i,j] \models \psi
\]

where \( \pi[i,j] = (\pi \leq j)_{\geq i} \). This can be done in quadratic time in the size of the path, and linear time in the size of the formula.

We now consider u.p. paths. The next lemma states that some transformations on paths do not affect the truth value of \( \text{LTL} + \text{Chop} \) formulae:

**Lemma 4.8.** Let \( \varphi \in \text{LTL} + \text{Chop} \), and \( m, n \geq |\varphi| \). Let \( u, v \) be two finite paths, and \( w \) be a (finite or infinite) path. Then

\[
u.v^m.w \models \varphi \text{ iff } u.v^n.w \models \varphi.
\]

We now perform the reduction from \( \text{PMC}(\text{LTL} + \text{Chop}) \) to \( \text{PMC}_f(\text{LTL} + \text{Chop}) \). We first exclude the trivial case when \( v \) is empty. We keep the notations of the proof of Theorem 3.6, and define new path and formulae: \( \pi' = uv^{\varphi|} \) and

\[
\psi_1 \psi_2 = \psi_1 (\psi_2 \land \chi_{|\psi_1|u\psi_2}) \quad \psi_1 C \psi_2 = \psi_1 (\psi_2 \land \chi_{|\psi_2|})
\]

For this path \( \pi' \), we now have \( \pi', i \models \chi_k \text{ iff } i \leq m + (|\psi| - k)p \).

With Lemma 4.8 we can prove the following by induction on the structure of \( \psi \):

**Lemma 4.9.** For all subformula \( \psi \) of \( \varphi \), for all \( i < m + (|\varphi| - |\psi|)p \), we have:

\[
u.v^\varphi, i \models \psi \text{ iff } \pi', i \models \overline{\psi}.
\]

It now suffices to observe that \( |\pi'| \) is in \( O(|uv| \times |\varphi|) \), and we obtain Proposition 4.6 from Proposition 4.7.

It turns out that, as with \( \text{LTL} + \text{Past} + \text{Now} \), we have a case where model checking a path is \( \text{PTIME}\)-complete:

**Theorem 4.10.** \( \text{PMC}(\text{LTL} + \text{Chop}) \) is \( \text{PTIME-complete} \).
In fact, PTIME-hardness already occurs with finite paths, i.e. for $\text{PMC}(\text{LTL}+\text{Chop})$. We prove this by a reduction from the (Synchronous Alternating Monotone) Circuit-Value Problem [GHR95, problem A.1.6]. We illustrate the reduction on an example and consider the circuit $C$ of figure 2.

Let $E_C$ denote the set of links (pairs of gates) in $C$. With $C$ we associate the finite path $\pi_C$ given in figure 3.

Finally, we define the following sequence of formulae:

$$\varphi_0 \overset{\text{def}}{=} n_{14}, \quad \varphi_{2k+1} \overset{\text{def}}{=} \bigvee_{i \text{ at level } 2k+1} \big( i \wedge \text{FG}j \big) C (X \emptyset \wedge \varphi_{2k}) C \quad \varphi_{2k+2} \overset{\text{def}}{=} \neg \left( \bigvee_{i \text{ at level } 2k+2} \big( i \wedge \text{FG}j \big) C (X \emptyset \wedge \neg \varphi_{2k+1}) \right)$$

Lemma 4.11. For any gate $n_i$ at level $p$ in $C$, $n_i$ evaluates to true in $C$ iff $\pi_C, (3i - 2) \models \varphi_p$.

Proof (Idea). By induction on $p$. The base case where $p = 0$ is obvious. For the induction step we first consider the case where $p = 2k + 1$ is odd. Hence $n_i$ is a disjunctive gate. The right-hand side of $\varphi_{2k+1}$ requires that we “chop” the path between two nodes labeled by a same $n_j$, and that this node satisfies the formula corresponding to the level below (hence gate $n_j$ evaluates to true by ind. hyp.). The left-hand side of $\varphi_{2k+1}$ ensures that gate $n_j$ is a child of the current $n_i$ (a finite path satisfies some FG$\psi$ iff its last state satisfies $\psi$). Thus $\pi_C, 3i - 2 \models \varphi_p$ iff a child of $n_i$ evaluates to true iff $n_i$ evaluates to true.

When $p$ is even, a dual reasoning applies. \hfill $\square$

Thus $C$ evaluates to true iff $\pi_C, 1 \models \varphi_l$, where $l$ is the height of $C$. Hence PTIME-hardness.
5 Model checking compressed paths

In this section we show that it is possible to efficiently model check paths that are given in compressed form, i.e. via some succinct encoding. One such encoding is the exponent notation, e.g. writing paths as

\[ u = \left( [(s_1.s_2.s_3)^3(s_4)^{12}(s_5.s_6)^7]^3 \right)^{1000}. \]

5.1 Compressed words

Working directly on compressed words is a standard technique in some fields, e.g. when handling long DNA strings in gene-mapping applications. Several encoding schemes are possible, and the more interesting ones are those where a compressed word can be exponentially more succinct than the described word.

Here we follow [PR99] and adopt the standard framework of straight-line programs, or SLP’s: these are context-free grammars where the non-terminals \( N_1, \ldots, N_k \) are ordered (\( N_1 \) being the axiom), and where every non-terminal has a single production of the form \( N_i \rightarrow a \) for a terminal \( a \), or \( N_i \rightarrow N_j N_k \) for some \( j, k > i \) [PR99]. For an SLP \( P \), we write \( w(P) \) for the unique word described by \( P \).

SLP’s are equivalent (polynomial-time inter-reducible) to Lempel-Ziv compression schemes but are mathematically nicer. They are more general than the exponent notation. The algorithms we give for SLP’s easily adapt to these other compression schemes.

5.2 Model checking compressed paths

A compressed path is a pair \((P_1, P_2)\) of two SLP’s, encoding the u.p. path \( w(P_1).w(P_2)^\omega \). Since compressed paths are succinct descriptions, we should expect that model-checking paths given in compressed form is hard. This is indeed the case with LTL model checking:

**Theorem 5.1.** Model checking LTL formulae on compressed paths is PSPACE-complete.

However, the difficulty has more to do with the LTL formulae than with the compressed paths, as our next result shows:

**Theorem 5.2.** Checking whether a compressed path is recognized by a Büchi automaton is PTIME-complete.

Checking whether a compressed path \((P_1, P_2)\) satisfies an LTL formula \( \varphi \) can be done in time \((|P_1| + |P_2|)2^{O(|\varphi|)}\), hence for long paths and simple fixed formulae, model-checking compressed paths is essentially “linear-time”.

The rest of the section proves the above two theorems. We observe the usual pattern: u.p. paths are not harder than finite paths.
N. Markey and Ph. Schnoebelen

<table>
<thead>
<tr>
<th></th>
<th>checking paths</th>
<th>checking compressed paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTL formulae</td>
<td>PTIME-easy</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>Finite state automata</td>
<td>NL-complete</td>
<td>PTIME-complete</td>
</tr>
</tbody>
</table>

**Fig. 4.** Checking compressed paths

### 5.3 Compressed paths accepted by Büchi automata

**Proposition 5.3.** [PR99] Saying whether \( w(P) \) is recognized by \( A \) (for \( P \) an SLP, and \( A \) a finite-state automaton) can be done in time \( O(|P| \times |A|^3) \).

**Proof.** [PR99] describes a simple dynamical programming solution. For two states \( r, s \) of \( A \) and non-terminal \( X_i \) of \( P \), set \( T[r, s, i] = \text{true} \) iff \( w(X_i) \) labels a path going from \( r \) to \( s \) in \( A \). Obviously, if \( X_i \rightarrow X_j X_k \) is a rule in \( P \), then \( T[r, s, i] = \bigvee_u T[r, u, j] \land T[u, s, k] \). Hence the table \( T[...] \) is easy to fill. Then we can use \( T[...] \) to see whether \( w(P) \), i.e. \( w(X_1) \), labels an accepting path. \( \square \)

**Corollary 5.4.** Saying whether a compressed path \( (P_1, P_2) \) is recognized by \( A \) (a Büchi automaton) can be done in time \( O((|P_1| + |P_2|) \times |A|^3) \).

**Proof (Idea).** This is a simple extension of the previous algorithm. For example, one can build a second table \( T'[...] \) s.t. \( T'[r, s, i] = \text{true} \) iff a power of \( w(X_i) \) labels a path going from \( r \) to \( s \) and visiting an accepting state of \( A \). \( \square \)

**Proposition 5.5** ([MS03]). Saying whether \( w(P) \) is recognized by a deterministic finite-state automaton \( A \), is PTIME-hard (hence PTIME-complete).

This shows a situation where compressed words are harder than uncompressed words since recognizability by a FSA is NL-complete for uncompressed words.

### 5.4 Compressed paths satisfying LTL formulae

The easy part of Theorem 5.1 is the upper bound:

**Proposition 5.6.** Deciding whether a compressed path satisfies an LTL formula can be done in polynomial-space.

**Proof (Idea).** Model checking LTL formulae on products of concurrent Kripke structures is PSPACE-complete [HKV02], and compressed paths can easily be encoded in such products. \( \square \)

PSPACE-hardness is more involved. Note it already occurs with finite paths:

**Proposition 5.7.** Deciding whether a finite compressed path satisfies an LTL formula is PSPACE-hard.
We now sketch the proof. It is by reduction from Quantified Boolean Formula (QBF). Assume $I$ is a QBF instance of the form $\exists v_1 \forall v_2 \exists v_3 \ldots \forall x_n (\bigwedge_i \bigvee_j l_{i,j})$ where every $l_{i,j}$ is $\pm v_{m_{i,j}}$, i.e. a Boolean variable from $V = \{v_1, \ldots, v_n\}$ or its negation.

With $I$ we associate a compressed word that lists all valuations for the $V$-variables in lexicographical order. This is given by the following SLP:

$$P_I = \begin{cases} N_1 = b_1.N_2.t_1.e_1.b_1.N_2.f_1.e_1 \\ N_2 = b_2.N_3.t_2.e_2.b_2.N_3.f_2.e_2 \\ \ldots \\ N_n = b_n.t_n.e_n.b_n.f_n.e_n \end{cases}$$

Here letters $t_i$ and $f_i$ state that we $v_i$ is true and, resp., false. Letters $b_i$ and $e_i$ are begin and end markers.

We now encode $I$ via $\varphi_I$, the following LTL formula:

$$[b_1 \land ([b_2 \Rightarrow ([b_3 \land \ldots [b_n \Rightarrow \bigwedge_{i,j} (t_{n_{i,j}} B e_n)]U e_{n-1} \ldots ]B e_2)]U e_1)]B \bot$$

where $\varphi B \psi$, defined as $(\neg \psi) U \varphi$, is short for “$\varphi$ at least once before a $\psi$”. In the above formula, $b_1 B \bot$ encodes “there is a position where a value for $v_1$ is picked”, $b_2 U e_1$ encodes “for all positions where a value for $v_2$ is picked before we change the value for $v_1$”, etc. When we look at a position where $v_n$ receives a value, the current valuation for all of $V$ can be recovered by writing “$t_k B e_n$” for of $v_k$.

Finally, the QBF instance $I$ is true iff $w(P_I) \models \varphi_I$. Hence we have provided a logspace reduction from QBF to model-checking LTL formulae on finite compressed paths.

### 6 Conclusions

We considered the problem of model checking a finite (or ultimately periodic) path. This is a fundamental problem in runtime verification, and it occurs in many other verification situations. This problem has not yet been the subject of serious fundamental investigation, probably because it looks like it is trivial.

We argue that “model checking a path” should be recognized as an interesting problem, and identified as such whenever it occurs. The main benefits one can expect are specialized algorithms that are more efficient than the usual algorithms we use (algorithms that were designed for the general case of model checking Kripke structures). We illustrate this with two kinds of specialized algorithms: model checking a path can be done efficiently (sometimes in polynomial-time) even when using richly expressive temporal logics that would usually be considered as highly intractable, and model checking a path can be done efficiently (sometimes in polynomial-time) even when the path is given in compressed form.

We feel this opens the door to a whole line of investigations, aiming at finding efficient algorithms for the whole variety of path model checking problems that naturally occur in practice.
From a more theoretical viewpoint, the basic problem of model checking
\( LTL \) formulae over finite or ultimately periodic paths should be considered as an
important open problem. It is not known whether the problem is PTIME-hard,
or whether it admits efficient parallel algorithms (e.g. in NC), or memory-efficient
algorithms (e.g. in SC). The gap between the known upper and lower bounds is
quite large, but we have been unable to narrow it.

Acknowledgement

The anonymous referees made several suggestions that greatly helped improve
this paper.

References

[BBF+01] B. Béard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and
Ph. Schnoebelen. Systems and Software Verification. Model-Checking Techni-

Bounded model checking. In Highly Dependable Software, volume 58 of

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. Model Checking. MIT Press,
1999.

queries in relational databases. In Proc. 9th ACM Symp. Theory of Com-

[Dru00] D. Drusinsky. The Temporal Rover and the ATG Rover. In SPIN Model
Checking and Software Verification, Proc. 7th Int. SPIN Workshop, Stan-
ford, CA, USA, Aug. 2000, volume 1885 of Lecture Notes in Computer Sci-

[DS02] S. Demri and Ph. Schnoebelen. The complexity of propositional linear tem-
poral logics in simple cases. Information and Computation, 174(1):84–103,
2002.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor,
Handbook of Theoretical Computer Science, volume B, chapter 16, pages

[FS01] B. Finkbeiner and H. Sipma. Checking finite traces using alternating au-
tomata. In Proc. 1st Int. Workshop on Runtime Verification (RV’01), Paris,

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. Limits to Parallel Computa-

of fairness. In Proc. 7th ACM Symp. Principles of Programming Languages

[Hav00] K. Havelund. Using runtime analysis to guide model checking of Java pro-
grams. In SPIN Model Checking and Software Verification, Proc. 7th Int.
SPIN Workshop, Stanford, CA, USA, Aug. 2000, volume 1885 of Lecture


