Automata-based Verification of Linear Temporal Logic Models with Bounded Variability

Carlo A. Furia
ETH Zurich, Switzerland
caf@inf.ethz.ch

Paola Spoletini
Università degli Studi dell'Insubria, Italy
paola.spoletini@uninsubria.it

***Abstract*—A model has variability bounded by v/k when the state changes at most v times over any linear interval containing k time instants. When interpreted over models with bounded variability, specification formulae that contain redundant metric information—through the usage of *next* operators—can be simplified without affecting their validity. This paper shows how to harness this simplification in practice: we present a translation of LTL into Büchi automata that removes redundant metric information, hence makes for more efficient verification over models with bounded variability. To show the feasibility of the approach, we also implement a proof-of-concept translation in ProMeLa and verify it using the Spin off-the-shelf model checker.**

I. Introduction and Overview

Linear temporal logic (LTL) formulae model linear sequences of discrete states that evolve "one step at a time". The state may change between some pairs of adjacent steps and not change between others; we say that a linear model has variability *bounded by* v/k (read "v over k") when the state changes at most v times over any interval containing exactly k steps.

Bounded variability limits the amount of information necessary to define a model: characterizing the state over v time instants is sufficient to completely describe the temporal behavior over a window of length k. Consider now an LTL formula ϕ that describes *distances* among states by means of the *next* operator X: for example, $\mathsf{X}\mathsf{X}\mathsf{X}\,p$ says that p holds in the future, at a distance of 3 steps. If ϕ describes distances as large as k (i.e., k nested occurrences of X) but we are interested only in its models with variability bounded by v/k for $v < k$, ϕ contains metric information that is redundant, as all the "action" occurs only over up to v steps.

In previous work [1], we showed how to remove the redundant information and produce a *smaller* formula ϕ' whose validity is equivalent to ϕ's validity over models with variability bounded by v/k; the size of ϕ' depends on v but not on k, hence the size of ϕ' is significantly smaller than the size of ϕ when $v \ll k$. Unfortunately, this result turns out to have mostly *theoretical* interest: the transformation from ϕ to ϕ' reduces the size due to k but also introduces a polynomial blow-up that, while it does not affect the worst-case complexity of the validity problem, is too conspicuous in practice: ϕ' is often still too large to be verified with standard tools for LTL.

The present papers shows how to get around this practical shortcoming. The solution has two elements. First, Section III presents a *different* transformation of ϕ into a formula Φ that is equivalent for models with variability bounded by v/k. The size of Φ also does not depend on k but, surprisingly, is otherwise *exponential* in another of ϕ's size parameters! This exponential blow-up is, however, completely immaterial: Section IV describes a direct translation of Φ into equivalent Büchi automata that can be analyzed efficiently in practice.

More precisely, some components of the translation use alternating Büchi automata extended with finite-domain counters; these make for a direct implementation of them into the ProMeLa language input to the Spin model checker. Section V presents a few details of the ProMeLa implementation and describes experiments where we used the technique presented in the rest of the paper to check for satisfiability—over models with bounded variability—an LTL example specification of a simple "elections" scenario. The large distances used in the specification, required to describe time units at different levels of granularity, make its verification unfeasible for standard LTL techniques that do not exploit the bounded variability assumption.

Related work: One of the original motivations to study models with bounded variability comes from the desire to reason about systems with heterogeneous time granularities [2], where large and small distances coexist. A few authors have discussed the modeling challenges brought by such systems [3], [4], [5], [6], [7], [8], and have demonstrated their relevance in some application domains such as medical data.

Model checking and verification using LTL and automata is a subject extensively studied for over three decades. The complexity of checking the validity of LTL formulae [9], [10] and the relations between LTL and Büchi automata [11], [12] are well-known; Section II recalls some of the main results, which are used in the rest of the paper. The *automata-theoretic* approach [11], [12] to LTL has produced not only a comprehensive theoretical framework, but also practical scalable implementations such as the Spin model checker [13], which we used in the experiments discussed in Section V. The characterization of repeated "stuttering" steps in LTL models has been studied by Lamport [14] and others [15], [16], [17].

In spite of the extensive work on LTL, to our knowledge the present paper and our previous work [1] are the first approaches that target LTL verification *for models with bounded variability*. Some of the ideas used in our work are inspired—and named after—similar notions introduced for *dense* time models; in particular, Pnueli operators [18] and bounded variability itself [19], [20].

As we explain in Section III, our previous work [1] represents a theoretical counterpart to the present paper; in particular, we re-use some techniques of [1] to establish the correctness of a similar transformation. While the presentation of the present paper is entirely self-contained, we warn readers already familiar with [1] that we have occasionally changed the notation and the details of some definitions to better focus the presentation on the novel contributions.

II. Preliminaries

A. Words and Variability

Words and trees: An *ω-word* (or simply *word*) w over a set S of propositional letters is a mapping $w : \mathbb{N} \to 2^S$ that associates to every discrete instant i (also called *step*) the subset $w(i) \subseteq S$ of propositions that hold at i. $\mathcal{W}(S)$ denotes the set of all words over S.

A *tree* is a set $T \subseteq \mathbb{N}^*$ of sequences of natural numbers that is *prefix-closed*, that is, if $x \cdot c \in T$—for $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$—then $x \in T$ as well. The elements of T are called *nodes*; the empty sequence ϵ denotes T's *root*; and if $x \cdot c \in T$ we say that $x \cdot c$ is a *successor* of x. A node without successors is called *leaf*. A *path* of T is a sequence $\pi_0, \pi_1, \ldots$ of nodes of T such that $\pi_0 = \epsilon$ and, for every $j \geq 0$, either π_j is a leaf or π_{j+1} is a successor of π_j. A *S-labeled tree* is a pair $\langle T, \lambda \rangle$ where T is a tree and $\lambda : T \to S$ assigns an element of S to each node of T.

Stuttering: A step $i \in \mathbb{N}$ is *stuttering* in a word w if $w(i+1) = w(i)$ and there exists a later step $j > i$ such that $w(j) \neq w(i)$. Conversely, a *non-stuttering* step (*nss* for short) i is such that either $w(i+1) \neq w(i)$ or, for all $j > i$, $w(j) = w(i)$. Intuitively, a stuttering step is redundant: it records information that is repeated identically in the next step, hence it only carries information about distance between states. A word is *stutter-free* if it has no stuttering steps. Two words are *stutter-equivalent* if removing all their stuttering steps reduces both to the same stutter-free word.

Bounded variability: Given two positive integers v, k, a word w has *variability bounded by v/k* (or simply *is v/k bounded*) if, for all $i \in \mathbb{N}$, at most v steps among $i, i+1, \ldots, i+k-1$ are non-stuttering in w. A set W of words has variability bounded by v/k if every word $w \in W$ is v/k bounded. $\mathcal{W}(S, v/k)$ denotes the set of all words over S with variability bounded by v/k.

B. Linear Temporal Logic

LTL *syntax:* LTL formulae are defined by:

$$\mathtt{LTL} \ni \phi ::= \top \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \,\mathsf{U}\, \phi_2 \mid \mathsf{X}\,\phi$$

where p ranges over a set $\mathcal{P}$ of propositional letters; we use the standard abbreviation for $\bot$ (false), $\vee$ (or), $\Rightarrow$ (implies), $\Leftrightarrow$ (iff); and for the derived temporal operators $\mathsf{F}\phi$ (eventually: $\top \,\mathsf{U}\, \phi$), $\mathsf{G}\phi$ (always: $\neg\mathsf{F}\neg\phi$), and $\mathsf{X}^{\,k}\phi$ (distance: $\underbrace{\mathsf{X}\,\mathsf{X}\cdots\mathsf{X}}_{k}\,\phi$), where $k \geq 0$. The *size* $|\phi|$ of an LTL formula ϕ is the size of its encoding as a string.

Qualitative and propositional LTL*:* $\mathtt{L}(\mathsf{U})$ denotes the *qualitative* subset of LTL, which does not use the X operator. $\mathtt{P}(\mathcal{P})$ denotes the purely *propositional* subset of LTL over $\mathcal{P}$, which does not use any temporal operator. A formula $\pi \in \mathtt{P}(\mathcal{P})$ unambiguously identifies a subset of $2^{\mathcal{P}}$; for example, $\neg p$ corresponds to all subsets X of $\mathcal{P}$ such that $p \notin X$. Based on this correspondence, we will use the two notations equivalently when convenient. $\mathtt{P}^+(\mathcal{P})$ denotes the *positive* subset of $\mathtt{P}(\mathcal{P})$ which only uses $\wedge$ and $\vee$.

LTL *semantics:* The satisfaction relation $\models$ defines the semantics of a generic LTL formula ϕ, interpreted at position $i \in \mathbb{N}$ over a word w.

$$\begin{array}{lll}
w, i \models \top & & \\
w, i \models p & \text{iff} & p \in w(i) \\
w, i \models \neg\phi & \text{iff} & w, i \not\models \phi \\
w, i \models \phi_1 \wedge \phi_2 & \text{iff} & w, i \models \phi_1 \text{ and } w, i \models \phi_2 \\
w, i \models \phi_1 \,\mathsf{U}\, \phi_2 & \text{iff} & \text{for some } j \geq i,\ w, j \models \phi_2 \\
 & & \text{and for all } i \leq k < j,\ w, k \models \phi_1 \\
w, i \models \mathsf{X}\,\phi & \text{iff} & w, i+1 \models \phi \\
w \models \phi & \text{iff} & w, 0 \models \phi
\end{array}$$

$[\![\phi]\!]$ denotes the set $\{w \in \mathcal{W}(\mathcal{P}) \mid w \models \phi\}$ of all models of ϕ. ϕ is *satisfiable* if $[\![\phi]\!] \neq \emptyset$ and is *valid* if $[\![\phi]\!] = \mathcal{W}(\mathcal{P})$. Two formulae ϕ_1, ϕ_2 are *equivalent* if $[\![\phi_1]\!] = [\![\phi_2]\!]$; they are *equi-satisfiable* if they are either both satisfiable or both unsatisfiable.

Proposition 1.
- *[9] The satisfiability problems for* LTL *and for* $\mathtt{L}(\mathsf{U})$ *are PSPACE-complete.*
- *[14] The set $[\![\psi]\!]$ of all models of any qualitative formula $\psi \in \mathtt{L}(\mathsf{U})$ is closed under stutter-equivalence.*

Pnueli operators: The *qualitative extended Pnueli operators* (or simply *Pnueli operators*) are defined by:

$$\mathsf{P}_k^{n;\langle n_1,\ldots,n_k\rangle}(\psi_1, \ldots, \psi_k)$$

for $k, n \in \mathbb{N}$, $n_1, \ldots, n_k \in \mathbb{N} \cup \{\infty\}$, and $\psi_1, \ldots, \psi_k \in \mathtt{L}(\mathsf{U})$. $\mathtt{L}(\mathsf{U}, \mathsf{Pn})$ denotes LTL extended with the Pnueli operators. The *size* $|\phi|$ of a $\mathtt{L}(\mathsf{U}, \mathsf{Pn})$ formula ϕ is the size of its encoding as a string, assuming a unary encoding for $k, n, n_1, \ldots, n_k$.

The satisfaction relation for Pnueli operators

$$w, i \models \mathsf{P}_k^{n;\langle n_1,\ldots,n_k\rangle}(\psi_1, \ldots, \psi_k)$$

holds if there exist k positions $i = i_0 \leq i_1 \leq \cdots \leq i_k$ such that, for all $1 \leq j \leq k$, the following hold: (1) $w, i_j + 1 \models \psi_j$; (2) if $n_j \neq \infty$ then there are no more than n_j nss in $[i_{j-1}, i_j - 1]$; (3) there are no more than n nss in $[i, i_k]$.

Proposition 2 ([1])**.** *The satisfiability problem for* $\mathsf{L}(\mathsf{U}, \mathsf{Pn})$ *is PSPACE-complete.*

Example 3. Consider the word w:

$$\begin{array}{ccccccccc} 1 & \underline{\mathbf{2}} & \underline{\mathbf{3}} & \underline{\mathbf{4}} & 5 & \underline{\mathbf{6}} & 7 & 8 & 9 \\ a & a & \overline{a} & \overline{a} & a & a & a & a & a \\ b & b & \overline{b} & b & \overline{b} & \overline{b} & b & b & b \end{array}$$

where nss are in bold and underlined, and $\overline{x}$ denotes $\neg x$. For the positions $1, 2, 6, 6$, $w, 1 \models \mathsf{P}_4^{5;\langle 3,2,\infty,3\rangle}(a, \neg b, a \wedge b, b)$ holds. On the contrary, $w, 1 \not\models \mathsf{P}_4^{3;\langle 3,2,2,1\rangle}(a, \neg b, a \wedge b, b)$: if $i_1, \ldots, i_4$ are the positions that match the semantics of the operator, it must be $i_3 \geq 6$, but there are $4 > 3$ nss in the interval $[1, 6]$.

C. Alternating Büchi Automata

An alternating (Büchi) automaton A is a tuple $\langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ is the finite input *alphabet*, Q is the finite set of *states*, $q_0 \in Q$ is the *initial* state, $\delta : Q \times \Sigma \to \mathsf{P}^+(Q)$ is the *transition function*, and $F \subseteq Q$ is the set of *final* states. If $\delta(q, \sigma)$ is purely disjunctive when it is defined, A is an ordinary *nondeterministic* (Büchi) automaton. The *size* $|A|$ of an alternating automaton A is $|Q| + |\delta|$, where $|\delta|$ is the sum of the formula sizes $|\delta(q, \sigma)|$ for all q, σ where δ is defined.

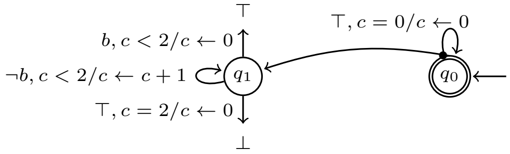
A *run* r of an alternating automaton over a word w over Σ is a Q-labeled tree $\langle T_r, \lambda_r\rangle$ such that: (1) the root $\epsilon \in T_r$ has label $\lambda_r(\epsilon) = q_0$; (2) if $x \in T_r$ is a node in the tree with label $\lambda_r(x) = q$ and the transition function defines $\delta(q, w(|x|)) = \theta$, then there is a set $\Pi = \{q_1, \ldots, q_k\} \subseteq Q$ such that $\pi \models \theta$—where we define $\pi(0) = \Pi$—and, for all $1 \leq j \leq k$, $x \cdot j \in T_r$ is a node of T_r with $\lambda_r(x \cdot j) = q_j$.

A path π of a run r is any path in T_r, hence an infinite sequence $q_0, q_1, \ldots$ of states; it is *accepting* if $q_i \in F$ for infinitely many i's. A run r is accepting if all its infinite paths are accepting. A word w is accepted if there exists an accepting run on it. $[\![A]\!]$ denotes the set of all words accepted by A (also called the *language* of A).

Proposition 4 ([12])**.**
- *The emptiness problem for languages defined by nondeterministic Büchi automata is NLOGSPACE-complete.*
- *For any alternating automaton of size n there exists an equivalent nondeterministic automaton of size $2^{\mathrm{O}(n)}$.*
- *For any* $\mathtt{LTL}$ *formula ϕ there exists an equivalent alternating automaton of size* $\mathrm{O}(|\phi|)$.

Automata with counters: Consider n integer counters, each with finite domain $[0, C_i]$ for $i = 1, \ldots, n$. An alternating automaton A with n counters has the same components as a regular alternating automaton, but its transition function has signature $\delta : Q \times \Sigma \times C \to \mathsf{P}^+(Q \times C)$ where $C = [0, C_1] \times \cdots \times [0, C_n]$ is the Cartesian product of the counters' domains. We define the semantics of an alternating automaton A with counters as a regular alternating automaton $\overline{A}$ with: states $\overline{Q} = Q \times C$; initial state $\overline{q_0} = \langle q_0, 0, \ldots, 0\rangle$; and transition function $\overline{\delta}$ such that $\overline{\delta}(\langle q, c_1, \ldots, c_n\rangle, \sigma) = \theta$ if and only if $\delta(q, \sigma, c_1, \ldots, c_n) = \theta$. The size of $\overline{A}$ also defines the size of A.

Example 5. The following alternating automaton with one counter c over $[0, 2]$ accepts the language $[\![\mathsf{G}(\mathsf{X}\, b \vee \mathsf{X}^2 b)]\!]$.


$\top$
$\top, c = 0/c \leftarrow 0$
$b, c < 2/c \leftarrow 0$
$\neg b, c < 2/c \leftarrow c + 1$
q_1
q_0
$\top, c = 2/c \leftarrow 0$
$\bot$

In the figure, a bullet marks pair of edges corresponding to conjunctive transitions, and the outgoing edges to the symbols $\top$ and $\bot$ denote transitions equal to the logic values $\top$ and $\bot$. The automaton spawns a separate parallel computation at every step (corresponding to the G); the computation counts up to two time steps and terminates successfully only if a b is encountered before the counter reaches 2.

III. From $\mathtt{LTL}$ to $\mathsf{L}(\mathsf{U}, \mathsf{Pn})$

Consider a generic $\mathtt{LTL}$ formula ϕ; without loss of generality, we can write ϕ in *separated-next form* with the occurrences of X separated from the rest of the formula as

$$\phi \quad \equiv \quad \psi_{\mathsf{U}} \wedge \mathsf{G}\left(\bigwedge_{1 \leq i \leq M} (x_i \Leftrightarrow \mathsf{X}^{d_i} \pi_i)\right),$$

where: $\psi_{\mathsf{U}} \in \mathsf{L}(\mathsf{U})$ is the purely qualitative part of the formula; for $i = 1, \ldots, M$, $x_i \in \mathcal{P}$ is a propositional letter and $\pi_i \in \mathsf{P}(\mathcal{P})$ is a propositional formula; and $0 < d_1 \leq \cdots \leq d_M$ are nonnegative *distances*. Also, $m \leq M$ denotes the number of *distinct* distances $d_1 = D_1 < \cdots < D_m = d_M$ among $d_1, \ldots, d_M$. Our ultimate goal is to determine if ϕ is satisfiable over words in $\mathcal{W}(\mathcal{P}, v/V)$ where v is any positive integer and $V = d_M$ is the maximum distance appearing in ϕ.

When V is large, $|\phi|$ is large as well; however, if $v \lll V$, ϕ encodes information that is redundant to decide its satisfiability over words with variability bounded by v/V. In [1], we described a polynomial-time transformation of ϕ into an equi-satisfiable formula $\phi' \in \mathsf{L}(\mathsf{U}, \mathsf{Pn})$ whose size is $\mathrm{O}(|\mathcal{P}|^2 + vM^2 + v^2 M)$. Since the size of ϕ' does not depend on V, if v is significantly smaller than V—in particular, exponentially smaller—then checking the satisfiability of ϕ' is much easier than checking the original formula ϕ. The transformation from ϕ to ϕ' stands as a theoretical result but is impractical because of the quadratic increase in size it introduces, which even becomes $\mathrm{O}(v^4)$ when we express the Pnueli operators in plain $\mathsf{L(U)}$; this blow-up is normally too conspicuous to handle with standard $\mathtt{LTL}$ tools.

To overcome these practical shortcomings, we follow the same principles used for ϕ' in [1] to build

$$\Phi \quad \equiv \quad \psi_{\mathsf{U}} \wedge \psi_s \wedge \psi_{\mathsf{P}} ,$$

where $\psi_{\mathsf{U}}, \psi_s$ are qualitative formulae in $\mathsf{L(U)}$, and $\psi_{\mathsf{P}} \in \mathsf{L(U, Pn)}$ uses Pnueli operators. ψ_{U} is the same in ϕ and Φ. ψ_s marks the nss of propositions in $\mathcal{P}$ through a fresh letter $s \notin \mathcal{P}$ that toggles precisely when some proposition in $\mathcal{P}$ changes truth value. Namely, if no proposition ever changes value then s holds, and any proposition changing triggers s to do the same (see [1, Sec. IV] for a formal definition in $\mathsf{L(U)}$, which is however straightforward).

To present ψ_{P}, we have to introduce some more notation. Let $B \in \{0,1\}^M$ denote a sequence of M Boolean values $b_1, \ldots, b_M$; then, for $i = 1, \ldots, M$, $[b_i g]$ is g if $b_i = 1$ and $\neg g$ if $b_i = 0$. ψ_{P} enumerates all possible 2^M truth assignments to $x_1, \ldots, x_M$:

$$\psi_{\mathsf{P}} \quad \equiv \quad \mathsf{G} \left(\bigwedge_{B \in \{0,1\}^M} X_1 \wedge \cdots \wedge X_M \;\Rightarrow\; \psi_B \right) ,$$

where, using the notation just introduced, $X_i = [b_i x_i]$ for $i = 1, \ldots, M$. For each truth assignment B, ψ_{P} constrains the truth value of $\pi_1, \ldots, \pi_M$ over the following v nss:

$$\psi_B \quad \equiv \quad \mathsf{P}_m^{v;\langle \delta_1, \ldots, \delta_m \rangle}(Y_1, \ldots, Y_m) ,$$

where $Y_j = \bigwedge_{d_k = D_j} [b_k \pi_k]$ is the conjunction of all π_k's corresponding to the same distance D_j in ϕ, for $j = 1, \ldots, m$; and, for $j = 1, \ldots, m$ and $D_0 = 0$, δ_j is

$$\delta_j \quad \equiv \quad \begin{cases} D_j - D_{j-1} & \text{if } D_j - D_{j-1} \leq v - j + 1 , \\ \infty & \text{otherwise} . \end{cases}$$

The intuition behind the transformation from ϕ to Φ is as follows. At every step, ϕ constrains the future values of the π_i's over a time window of length $V = d_M$ according to the current value of the x_i's. If the variability is bounded by v/V, however, no more than v changes of the proposition in $\mathcal{P}$—which determine the value of the π_i's—are possible. Correspondingly, ψ_{P} converts the distance constraints in ϕ into qualitative constraints over nss: rather than saying that π_1 must occur at distance d_1, π_2 at distance d_2, and so on, ψ_{P} only requires that π_1 occurs first, followed by π_2, and so on. Since Φ is qualitative, we can transform one of its models into a model for ϕ by adding stuttering steps so that the π_i's follow the original metric constraints in ϕ. The additional constraints introduced by the δ_i's ensure that this "padding" is always possible (since we cannot remove nss, we must ensure that there are not "too many" of them between any two consecutive distances). Using the same techniques used in [1], it is not difficult to make this intuition rigorous and prove the following.

Theorem 6. *ϕ is satisfiable over words in $\mathcal{W}(\mathcal{P}, v/V)$ if and only if Φ is satisfiable over words in $\mathcal{W}(\mathcal{P} \cup \{s\})$.*

Section IV shows how to exploit automata-theoretic techniques to check efficiently the satisfiability of Φ, hence, thanks to Theorem 6, to decide the satisfiability of ϕ over models with bounded variability. Section V demonstrates that the automata-theoretic techniques are directly implementable using standard model-checking tools.

IV. From $\mathsf{L(U, Pn)}$ to Automata

This section describes an efficient translation of the components of Φ into automata. Each component ψ_k becomes an automaton $A^k = \langle \Sigma, Q^k, q_0^k, \delta^k, F^k \rangle$ such that $[\![\psi_k]\!] = [\![A^k]\!]$; whenever clear from the context, we drop the superscript k from the automaton components' names. For simplicity of presentation, we assume that the alphabet Σ of all automata equals the set of all propositional letters used in any formula, and omit the straightforward details of how to handle letters appearing only in certain components.

We do not discuss the translation of $\psi_{\mathsf{U}} \in \mathsf{L(U)}$ into A^{U}, because ψ_{U} can have any structure, hence we just rely on standard translations of $\mathtt{LTL}$ into alternating automata.

A. Marking Non-stuttering Steps

The automaton A^s for ψ_s has $2 \cdot 2^{|\mathcal{P}|} + 1$ states $\{q_0\} \cup \{q_P^+, q_P^- \mid P \subseteq \mathcal{P}\}$: two for each subset of $\mathcal{P}$, and a distinct initial state q_0; all states other than q_0 are accepting.

Computations start in the initial state q_0. According to the initial values of the propositions in $\mathcal{P}$, the automaton goes to a state in q^+ (if s holds) or in q^- (if s doesn't):

$$\delta(q_0, P \cup \{s\}) = q_P^+ \quad \text{and} \quad \delta(q_0, P) = q_P^- ,$$

for $P \in 2^{\mathcal{P}}$.

The automaton leaves states of the form q_P^+ only if the values of the propositions change and s turns false; otherwise, s must hold continuously:

$$\delta(q_P^+, P \cup \{s\}) = q_P^+ \quad \text{and} \quad \delta(q_P^+, \widehat{P}) = q_{\widehat{P}}^- ,$$

where $\widehat{P}$ ranges over the complement set of propositions $2^P \setminus P$. Similarly, A^s leaves states of the form q_P^- only if the values of the propositions change and s turns true; otherwise, s must not hold:

$$\delta(q_P^-, P) = q_P^- \quad \text{and} \quad \delta(q_P^-, \widehat{P} \cup \{s\}) = q_{\widehat{P}}^+ .$$

Figure 1 shows A^s for $\mathcal{P} = \{a, b\}$, where every transitions that enters a state q_P^+ has label $P \cup \{s\}$, every transition that enters a state q_P^- has label P, and the initial transitions are dotted for readability.

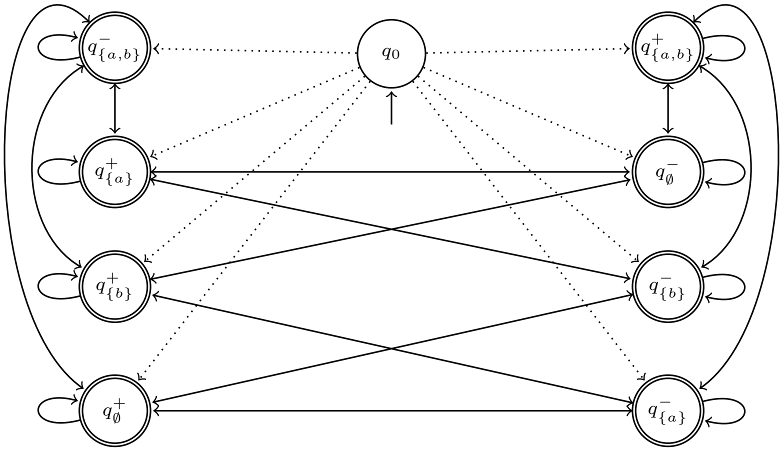


Figure 1. Automaton A^s for ψ_s, when $\mathcal{P} = \{a, b\}$.

B. Translating Pnueli Operators

The automaton A^B for ψ_B has $2m + 2$ states $\{q_0, q_F\} \cup \{q_i^+, q_i^- \mid 1 \leq i \leq m\}$ and two counters $c_G \in [0, v]$ and $c_L \in [0, \delta]$ where $\delta = \max_i \delta_i$; c_G is the "global" counter that counts the following v nss, whereas c_L is the "local" counter that measures each of the δ_i's. Computations start in the initial state q_0 and should end in the only accepting state q_F. At any point during a computation, A^B is in state q_i° when it is ready to read $Y_i, \ldots, Y_M$ before c_G reaches v, with $\circ = +$ if s held in the latest step, and $\circ = -$ otherwise. In the following, d_G ranges over $[0, v]$, e_G over $[0, v-1]$, d_L^i over $[0, \delta_i] \cap [0, v]$, and e_L^i over $[0, \delta_i - 1] \cap [0, v-1]$.

The automaton stays in q_i^+ (resp. q_i^-) without changing the counters if s holds (resp. does not hold) and Y_i is false:

$$\delta(q_i^+, s \wedge \neg Y_i, d_G, d_L^i) = \langle q_i^+, d_G, d_L^i \rangle \,,$$
$$\delta(q_i^-, \neg s \wedge \neg Y_i, d_G, d_L^i) = \langle q_i^-, d_G, d_L^i \rangle \,,$$

When s toggles but Y_i is false, A^B rotates between q_i^+ and q_i^- while incrementing both counters by one:

$$\delta(q_i^+, \neg s \wedge \neg Y_i, e_G, e_L^i) = \langle q_i^-, e_G + 1, e_L^i + 1 \rangle \,,$$
$$\delta(q_i^-, s \wedge \neg Y_i, e_G, e_L^i) = \langle q_i^+, e_G + 1, e_L^i + 1 \rangle \,,$$

with $i \leq m$. When Y_i holds, if A^B nondeterministically guesses that this is the "right" occurrence of Y_i it goes to $q_{i+1}^\pm$ and resets the local counter; otherwise, it remains in $q_i^\pm$. For $i < m$, define $\delta(q_i^+, s \wedge Y_i, d_G, d_L^i)$ as

$$\langle q_{i+1}^+, d_G, 0 \rangle \vee \langle q_i^+, d_G, d_L^i \rangle \,,$$

and $\delta(q_i^+, \neg s \wedge Y_i, e_G, e_L^i)$ as

$$\langle q_{i+1}^-, e_G + 1, 1 \rangle \vee \langle q_i^-, e_G + 1, e_L^i + 1 \rangle \,.$$

The transitions outgoing q_i^- when Y_i holds are obtained by duality with the substitutions $- \leftrightarrow +$ and $\neg s \leftrightarrow s$. From now on, we omit from the presentation all transitions from states of the form q_i^- that follow by duality.

The transition from q_i^+ to q_{i+1}^- is also possible when the local counter *equals* δ_i: when s takes a new value we are already past the nss that occurred in the previous step, and the new nss is counted in the next round starting with $c_L = 1$:

$$\delta(q_i^+, \neg s \wedge Y_i, e_G, \delta_i) = \langle q_{i+1}^-, e_G + 1, 1 \rangle \,.$$

Notice that the value of e_L^i does not matter in transitions corresponding to $\delta_i = \infty$, hence we need not update the local counter (we omit the straightforward details).

Now, we define the initial and final transitions. Initially, A^B goes to q_1^+ or q_1^- according to the first value of s, e.g.:

$$\delta(q_0, s \wedge \neg Y_1, 0, 0) = \langle q_1^+, 0, 0 \rangle \,,$$
$$\delta(q_0, s \wedge Y_1, 0, 0) = \langle q_1^+, 0, 0 \rangle \vee \langle q_2^+, 0, 0 \rangle \,.$$

The last successful transition from q_m^+ may lead to q_F:

$$\delta(q_m^+, \neg s \wedge Y_m, e_G, e_L^i) = \langle q_F, 0, 0 \rangle \vee \langle q_m^-, e_G + 1, e_L^i + 1 \rangle,$$
$$\delta(q_m^+, s \wedge Y_m, d_G, d_L^i) = \langle q_F, 0, 0 \rangle \vee \langle q_m^+, d_G, d_L^i \rangle.$$

The transition to q_F is possible also when the local counter equals δ_m, the global counter equals v, or both:

$$\delta(q_m^+, \neg s \wedge Y_m, v, d_L^i) = \langle q_F, 0, 0 \rangle \,,$$
$$\delta(q_m^+, \neg s \wedge Y_m, d_G, \delta_m) = \langle q_F, 0, 0 \rangle \,.$$

Finally, the accepting state is absorbing:

$$\delta(q_F, \top, 0, 0) = \top \,.$$

The transitions introduced so far do not allow for some of the following steps to coincide. To account for this case, we introduce $\mathrm{O}(m^2)$ transitions that "jump" multiple states

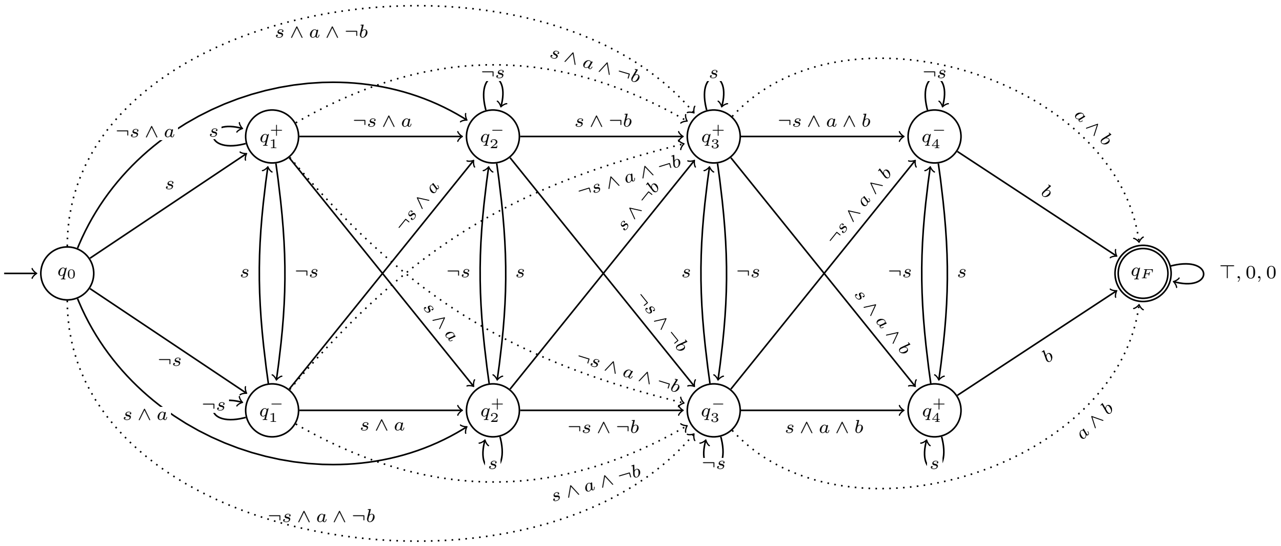


Figure 2. Automaton A^B for $\psi_B = \mathsf{P}_4^{6;\langle 3,2,\infty,1\rangle}(a, \neg b, a \wedge b, b)$.

while reading multiple Y_i's. For $1 \leq i \leq m$, $i < k < m$, we add the transitions:

$$\begin{aligned}
\delta(q_i^+, \neg s \wedge Y_i \wedge \cdots \wedge Y_k, e_G, d_L^i) &= \langle q_{k+1}^-, e_G + 1, 1\rangle \,, \\
\delta(q_i^+, s \wedge Y_i \wedge \cdots \wedge Y_k, d_G, d_L^i) &= \langle q_{k+1}^+, d_G, 0\rangle \,, \\
\delta(q_i^+, Y_i \wedge \cdots \wedge Y_m, d_G, d_L) &= \langle q_F, 0, 0\rangle \,, \\
\delta(q_0, s \wedge Y_1 \wedge \cdots \wedge Y_k, 0, 0) &= \langle q_{k+1}^+, 0, 0\rangle \,, \\
\delta(q_0, \neg s \wedge Y_1 \wedge \cdots \wedge Y_k, 0, 0) &= \langle q_{k+1}^-, 0, 0\rangle \,, \\
\delta(q_0, Y_1 \wedge \cdots \wedge Y_m, 0, 0) &= \langle q_F, 0, 0\rangle \,.
\end{aligned}$$

Figure 2 shows A^B for $\mathsf{P}_4^{6;\langle 3,2,\infty,1\rangle}(a, \neg b, a \wedge b, b)$; for readability, the "jump" edges we described last are dotted, and the update of the counters are omitted.

The automaton A^{P} for ψ_{P} combines the various A^B through conjunctive alternation, as illustrated in Figure 3. First, coalesce all their initial states into a single one, also called q_0, and all their final states into another one, also called q_F; this gives an automaton with $2 + m2^{M+1}$ states. Then, for each $B \in \{0,1\}^M$, replace each initial transition of the component A^B of the form $\delta^B(q_0^B, \mathcal{S}, 0, 0) = \theta^B$, $\mathcal{S} \in \mathsf{P}(\mathcal{P} \cup \{s\})$, with:

$$\delta(q_0, \mathcal{S} \wedge X_1 \wedge \cdots \wedge X_M, 0, 0) = \langle q_0, 0, 0\rangle \wedge \theta^B \,;$$

that is, at any step, the value of the propositions $x_1, \ldots, x_M$ activates a uniquely determined component A^B, while a parallel computation remains in q_0 ready for the next step.

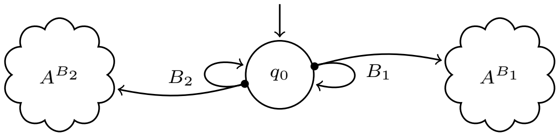


Figure 3. Schema of automaton A^{P}.

C. Overall Automaton

The automaton $\mathcal{A}$ for Φ is a nondeterministic Büchi automaton, given by the intersection of 3 components:

A^{U} is a standard encoding of `LTL` into nondeterministic Büchi automata, of size $2^{\mathrm{O}(|\psi_{\mathsf{U}}|)}$;

A^s is a nondeterministic Büchi automaton of size $\mathrm{O}\left(2^{|\mathcal{P}|}\right)$;

A^{P} is an *alternating* automaton of size $\mathrm{O}\left(2^M m^2 v^2\right)$, where the v^2 factor accounts for counters.

The size of A^{U} is fixed and reflects the fact that satisfiability of `LTL` is PSPACE-complete; we need not worry about it.

A^{P} is an alternating automaton; to compose it with the others, we have to convert it into a nondeterministic automaton, but this involves an additional exponential blow-up in the worst-case. The usage of universal alternation is, however, quite restricted in A^{P}: at every step, A^{P} activates exactly one new component of size $\mathrm{O}(m^2 v^2)$ in parallel with the others, and each of these components is active for no longer than v nss. Therefore, the exponential size of A^{P} is immaterial in practice: if we compute the intersection and check for emptiness on the fly, we have to deal with an active nondeterministic Büchi automaton of size $\mathrm{O}(2^M + m^2 v^3)$. The "actual" size of $\mathcal{A}$—given by the product of the size of its components as Büchi automata—is then only singly exponential in M, v, and $|\mathcal{P}|$; this matches the theoretical complexity of [1], and is manageable in practice as we demonstrate in Section V.

V. Example and Implementation

The example used in our experiments is repeated from [1], and it is representative of several similar examples in the literature on time granularity.

The elections example: Consider elections that occur every four years, in one of two consecutive days. Proposition q marks the first day of every quadrennial, hence it holds initially and then precisely every $365 \cdot 4 = 1460$ days. The elections e occur once within every quadrennial; precisely they occur 40 or 41 days before the end of the quadrennial. Formula ϕ in SNF describes this behavior:

$$\phi \equiv \underbrace{q \wedge \mathsf{G}\begin{pmatrix}(u \Leftrightarrow \neg e \,\mathsf{U}\, q)\\ \wedge\, (v \Leftrightarrow \neg q \wedge \neg q \,\mathsf{U}\, q)\\ \wedge\, (q \Rightarrow x_2 \wedge x_5)\\ \wedge\, (q \Rightarrow \neg x_1)\\ \wedge\, (e \Rightarrow \neg q \,\wedge\, x_1)\\ \wedge\, (e \Rightarrow x_3 \vee x_4)\end{pmatrix}}_{\psi_{\mathsf{U}}} \wedge \mathsf{G}\begin{pmatrix}(x_1 \Leftrightarrow \mathsf{X}^{\,d_1} u)\\ \wedge\, (x_2 \Leftrightarrow \mathsf{X}^{\,d_2} v)\\ \wedge\, (x_3 \Leftrightarrow \mathsf{X}^{\,d_3} q)\\ \wedge\, (x_4 \Leftrightarrow \mathsf{X}^{\,d_4} q)\\ \wedge\, (x_5 \Leftrightarrow \mathsf{X}^{\,d_5} q)\end{pmatrix}$$

with $d_1 = d_2 = 1$, $d_3 = 40$, $d_4 = 41$, $d_5 = 1460$, and $\mathcal{P} = \{q, e, u, v, x_1, \ldots, x_5\}$. A variability of $6/1460$ makes such specification tight, as it allows *at most* 6 nss over a windows of length 1460: 2 of them accounts for q becoming true and then false again once, another 3 nss mark a similar double transition of e, and one extra nss is required by the auxiliary propositions $u, v, x_1, \ldots, x_5$.

Implementation in ProMeLa: We transformed ϕ into Φ for the elections example, according to Section III. Then, we implemented the automaton $\mathcal{A}$ described in Section IV for such Φ in the ProMeLa language [13]. ProMeLa is an expressive process description language for the Spin explicit-state model checker, suitable to describe finite-state computations and their coordination. Our implementation of $\mathcal{A}$ in ProMeLa has two main components, grouped in a global environment that handles coordination between them.

The first component `PsiUOpModel` corresponds to Büchi automaton $A^{\psi_{\mathsf{U}}}$ and is a fairly standard encoding of the $\mathsf{L}(\mathsf{U})$ formula ψ_{U}; `PsiUOpModel` is a ProMeLa process that acts as a (nondeterministic) *generator* of all models of ψ_{U}. The other component `Pnueli` implements the alternating automaton A^{P}; the expressiveness of ProMeLa makes for a straightforward implementation of the counters, which are just process integer variables. The code for `Pnueli` is partitioned in four parts, corresponding to the pairs of states q_i^+, q_i^- for $i = 1, \ldots, 4$, plus a general coordination section that activates new components at every time step according to the semantics of conjunctive alternation (see Figure 3) and re-uses processes that have terminated for new computations. The process running `Pnueli` acts as an *acceptor* that filters out the models generated by `PsiUOpModel` that also satisfy ψ_{P}. Finally, the functionality corresponding to A^s, which is independent of the particular form of ϕ, is directly implemented in the global environment: at every new step, the previous value of the propositions is compared with the new value, and s is complemented whenever the two differ.

Experiments: We verified the ProMeLa model using the Spin model checker. The verification consisted in checking the absence of *invalid end states*; if successful, this guarantees that Spin can expand the complete ProMeLa model, hence it can perform emptiness check and exhaustive checking of arbitrary LTL properties. The experiments ran Spin 6.10 on a Ubuntu box (kernel 2.6.32) with Intel Quad-Core2 CPU at 2.40 GhZ with 4 GB of RAM.

We ran verification on the model with a variability bound $v/V = v/1460$, for values of v between 6 and 38; for all such values, the results were the following in terms of total running time (in seconds), memory used (in MB), number of primitive states generated (in millions), and maximum expansion depth reached (in thousands).

Time (s)	Mem. (MB)	# S (10^6)	D (10^3)
48	2012	41.8	569

Since v is merely a global *upper bound* on the relative distance between events, not a distance that must be reached, the performance is insensitive to changes in the value of v as long as changing v does not change the values of the δ_j's (defined in Section III); this is the case for the chosen range $6 \leq v \leq 38$, and allows for some scalability.

Even if this is only a proof-of-concept as the example is not overly complicated, it is representative of a larger class of systems, and it demonstrates that our approach is feasible and scales. On the contrary, verifying the original formula ϕ "as is" is unfeasible with state-of-the-art techniques because of the sheer size of ϕ due to the $V = 1460$ nested occurrences of X. For example, the tool LTL2BA [21]—that translates LTL formulae into Büchi automata encoded in ProMeLa—runs out of resources while generating the automaton, and even an *ad hoc* implementation in ProMeLa using features such as counters would become too large for Spin to exhaustively analyze. Our encoding is instead significantly more concise and requires a sustainable amount of resources.

VI. Future Work

In future work, we plan to build an automated translator of LTL into ProMeLa implementing the construction for bounded variability introduced in the paper. With the translator, we will be able to perform more experiments with some of the domain-specific examples available in the literature (see the related work in Section I). In addition, we will try to construct simplifications similar to those discussed in the paper that are directly applicable to automata model (as opposed to LTL formulae), with the ultimate goal of applying the model-checking paradigm—based on the combination of automata and logic—under the bounded variability assumption.

Acknowledgements: Thanks to the anonymous reviewers for useful comments.

References

[1] C. A. Furia and P. Spoletini, "On relaxing metric information in linear temporal logic," in *TIME*. IEEE, 2011, pp. 72–79.

[2] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi, "Modeling time in computing: a taxonomy and a comparative survey," *ACM Computing Surveys*, vol. 42, no. 2, pp. 1–59, 2010.

[3] A. Belussi, C. Combi, and G. Pozzani, "Formal and conceptual modeling of spatio-temporal granularities," in *IDEAS*. ACM, 2009, pp. 275–283.

[4] C. Combi and S. Degani, "Building logical specifications of temporal granularities through algebraic operators," in *TIME*. IEEE Computer Society, 2009, pp. 107–114.

[5] M. Franceschet and A. Montanari, "Temporalized logics and automata for time granularity," *TPLP*, vol. 4, no. 5-6, pp. 621–658, 2004.

[6] C. Combi, A. Montanari, and P. Sala, "A uniform framework for temporal functional dependencies with multiple granularities," in *SSTD*, ser. LNCS, vol. 6849. Springer, 2011, pp. 404–421.

[7] A. Burns and I. J. Hayes, "A timeband framework for modelling real-time systems," *Real-Time Systems*, vol. 45, no. 1–2, pp. 106–142, 2010.

[8] E. Corsetti, E. Crivelli, D. Mandrioli, A. Morzenti, A. Montanari, P. San Pietro, and E. Ratto, "Dealing with different time scales in formal specifications," in *Int. Workshop on Software Specification and Design*, 1991, pp. 92–101.

[9] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *JACM*, vol. 32, no. 3, pp. 733–749, 1985.

[10] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, 1990, pp. 996–1072.

[11] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *LICS*. IEEE, 1986, pp. 332–344.

[12] J. Esparza, O. Kupferman, and M. Vardi, "Verification," in *Handbook on Automata Theory*, 2012.

[13] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[14] L. Lamport, "What good is temporal logic?" in *IFIP Congress*. North Holland/IFIP, 1983, pp. 657–668.

[15] D. Peled and T. Wilke, "Stutter-invariant temporal properties are expressible without the next-time operator," *IPL*, vol. 63, no. 5, pp. 243–246, 1997.

[16] K. Etessami, "A note on a question of Peled and Wilke regarding stutter-invariant LTL," *IPL*, vol. 75, no. 6, pp. 261–263, 2000.

[17] A. Kučera and J. Strejček, "The stuttering principle revisited," *Acta Informatica*, vol. 41, no. 7/8, pp. 415–434, 2005.

[18] Y. Hirshfeld and A. M. Rabinovich, "Logics for real time: Decidability and complexity," *Fundamenta Informaticae*, vol. 62, no. 1, pp. 1–28, 2004.

[19] T. Wilke, "Specifying timed state sequences in powerful decidable logics and timed automata," in *FTRTFT*, ser. LNCS, vol. 863. Springer, 1994, pp. 694–715.

[20] C. A. Furia and M. Rossi, "MTL with bounded variability: Decidability and complexity," in *FORMATS*, ser. LNCS, vol. 5215. Springer, 2008, pp. 109–123.

[21] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *CAV*, ser. LNCS, vol. 2102, 2001, pp. 53–65.