Electronic Communications of the EASST
Volume 72 (2015)

Proceedings of the 15th International Workshop on Automated Verification of Critical Systems (AVoCS 2015)

Computing Bounds for Counter Automata

Maximilien Colange, Dimitri Racordon, Didier Buchs

15 pages

Guest Editors: Gudmund Grov, Andrew Ireland
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Computing Bounds for Counter Automata

Maximilien Colange[1], **Dimitri Racordon**[1], **Didier Buchs**[1]

[1] first.last@unige.ch
Centre Universitaire d'Informatique
University of Geneva, Switzerland

Abstract: Qualitative formal verification, that seeks Boolean answers about the behavior of a system, is often insufficient for practical purposes. Observing quantitative information is of greater interest, e.g. for the calibration of a battery or a real-time scheduler. Historically, the focus has been on quantities in continuous domain, but recent years showed a renewed interest for discrete quantitative domains.

Counter Automata (CA) is a quantitative extension of classical ω-automata. Recently a nice theory has been developed for them that extends the qualitative setting, with counterparts in terms of logics, automata and algebraic structure. We propose an adaptation, with plenty of practical applications, of this formalism to express properties over discrete quantitative domains. The behavior of a Counter Automaton defines a function from infinite words to integers. Finding the bounds of such a function over a given set of words can be seen as an extension of qualitative universal and existential model-checking. Although the problem of determining whether such bounds are finite have already been addressed, efficient algorithms to compute their exact values still lack.

We propose an non-naive method for the computation of the exact values of these bounds. It relies on a generalization of the emptiness problem of ω-automata. To solve this generalized emptiness problem, we propose an algorithm that extends emptiness check algorithms based on SCC enumeration.

Keywords: model checking, non-stochastic quantitative properties, counter automata, ω-automata, emptiness check

1 Introduction

Qualitative verification, asking questions with Boolean answers about a system may be too strict for various applications. Calibrating a battery, timing a scheduler, measuring quality of service are practical problems of systems designers for which formal verification can offer a guarantee. Many works focus on the case of continuous quantitative domains (stochastic systems, real-time systems ...), and the case of discrete domains have long been overlooked.

The ability to count events is an important feature, e.g. to evaluate logical time (number of actions done by a robot, number of context switches done by a scheduler ...). Such measurements are of primary interest to evaluate the behavior of a system at early stages of development. Logical time can also serve as a first approximation of real-time, when events have a known bounded

duration. We seek in this paper to use a model of automata able to count events in a system with infinite behaviors, with a focus on applicability.

Among numerous quantitative extensions to finite automata, we focus on the one defined by Colcombet and Bojańczyk [BC06]. Finite automata are extended with a finite set of counters that can be incremented and reset. A special operation *observe* allows to store the current value of a counter to further determine the value of a run, as the infimum or the supremum of such stored values. They are part of a vast theory that nicely extends the finite automata theory, with their logical and algebraic counterparts, closure properties, over finite and infinite words and finite trees. Such automata define functions from words to integers, termed *cost functions*. Due to the undecidability of comparing two cost functions, many interesting features of this theory rely on the consideration of cost functions up to an equivalence relation that erases the exact values and only retains their boundedness of functions.

Regarding infinite words, on which we focus in this paper, the theory of cost functions has links with other extensions of automata or logics, by bounding a discrete quantity: bounding the maximal time between two returns to an accepting state in ω-automata [AHK10], or bounding the wait time for the *finally* operator in LTL [KPV07]. Considering exact values to count events is nevertheless a great tool for verification. Think for example to the maximal number of energy units consumed by a robot between two returns to its charging base, to calibrate a battery. Or the maximal number of simultaneous threads in a parallel computation, to tune an appropriate scheduler. Or the number of false steps permitted to a human operator before a safeguard restriction occurs. For such properties, determining whether the bound is finite or not is of little help. We thus propose to use the tools and methods developed towards cost function theory (over infinite words) to practical model-checking.

Our contribution is threefold:

- we propose a method to compute the bound of a counter automaton, through the construction of a so-called capped configuration automaton (Section 3);
- we define a generalized version of the emptiness check problem for ω-automata, to which our bound problem is reduced (Section 3);
- we adapt existing emptiness check algorithms to solve the latter problem (Section 4).

The paper is organized as follows: Section 2 recalls essential definitions and concepts of ω-automata, then introduces Counter Automata and their semantics through their *configuration automata*. Section 3 defines the bound value problem, and its reduction to a problem on a finite automaton, labeled the *capped configuration automaton*. Section 4 discusses algorithmic solutions for the latter problem, by adapting existing emptiness check algorithms for ω-automata. Section 5 presents related work. Finally Section 6 concludes the paper and presents perspectives for future work.

Notations $\mathbb{N}$ denotes the set of nonnegative integers, and $\leq$ denotes the usual order on integers. We define $\mathbb{N}_\infty = \mathbb{N} \cup \{\omega\}$, and we extend $\leq$ to $\mathbb{N}_\infty$ such that $n \leq \omega$ for every $n \in \mathbb{N}$.

Throughout the paper, Σ denotes a finite alphabet. Σ^n denotes the set of words of length n for $n \in \mathbb{N}$, Σ^* denotes the set of finite words over Σ and the empty word is noted ε. Σ^ω denotes the set of infinite words, or ω-words over Σ. The length of a word $u \in \Sigma^* \cup \Sigma^\omega$ is noted $|u|$. For $i < |u|$, the i-th letter of u is noted $u(i)$, so that $u = (u(i))_{i<|u|}$.

Given a set A and $\equiv$ an equivalence relation on A, the equivalence class of $a \in A$ is noted $[a]_{\equiv}$.

For $M \in \mathbb{N}$, we define the *cap relation* $\sim_M$ on $\mathbb{N}_\infty$ as follows: $n_1 \sim_M n_2$ iff $n_1 = n_2$ and $n_1 < M$, or both n_1 and n_2 are greater than M. Thus, $\sim_M$ is an equivalence relation that has $M+1$ equivalence classes $[0]_{\sim_M}$, $[1]_{\sim_M}$, ..., $[M]_{\sim_M}$, among which the M first are singletons. $\sim_M$ is compatible with $\leq$ on $\mathbb{N}_\infty$: for every $n < p$, for every $n' \sim_M n$ and $p' \sim_M p$, we have $n' < p'$ or $n' \sim_M p'$. Thus $\leq$ is well-defined on the quotient $\mathbb{N}_\infty / \sim_M$: $[0]_{\sim_M} < [1]_{\sim_M} < \cdots < [M]_{\sim_M}$.

When a variable denotes a vector, we add an arrow above it to distinguish it from scalar values. For instance, $\vec{x} \in \mathbb{N}^A$ denotes a vector $([x_a])_{a \in A}$ of nonnegative integers indexed by set A.

2 Counter ω-Automata

This section recalls the definition of ω-automata, and extends them to Counter ω-Automata.

2.1 ω-Automata

We first recall the concept of ω-automaton, a finite automaton running on infinite words. An ω-automaton defines (or *recognizes*) a *regular* ω-language. They are largely used to represent sets of infinite behaviors, such as the *actual* behaviors (or abstraction thereof) of an actual discrete event system, or the *desired* behaviors of a specification. Depending on the representation, each letter of a word represents a *state* or an *action* of said system or specification. A common practice uses sets of *system observables* as letters. We use ω-automata to represent the infinite behaviors of systems to be verified. ω-automata satisfy numerous closure properties (especially union, intersection, complementation) of great importance for verification algorithms design.

We use in this paper transition-based generalized Büchi conditions, for several reasons:

- ω-automata are often used to check whether a LTL formula holds on every execution of a system. In this approach, the (negation of the) LTL formula is translated into an equivalent ω-automaton. For such a translation, using generalized transition-based Büchi condition is quite natural, and almost all translation algorithms produce automata with generalized transition-based Büchi conditions [DG12]. Similarly, Counter Automata can be produced from a quantitative variant of LTL, named Cost LTL [KB12]. The translation algorithms are little affected by the addition of quantitative information, and also produce Counter Automata with generalized transition-based Büchi conditions.
- generalized conditions are more concise than degeneralized ones, i.e. they allow smaller automata for a given regular ω-language. Similarly, transition-based conditions are usually more concise than state-based ones. The complexity of emptiness check algorithms depends on the size of the input automaton, so conciseness is primordial. We will see later that conciseness is also particularly important when it comes to Counter Automata.

Definition 1 An ω-automaton is a tuple $\mathscr{A} = \langle Q, \Sigma, \Delta, Q_0, F \rangle$ such that:

- Q is a finite set of states;
- Σ is a finite alphabet;
- $\Delta \subseteq Q \times \Sigma \times 2^F \times Q$ is the transition relation;
- $Q_0 \subseteq Q$ is a set of initial states.

- F is a finite set of acceptance marks;

Let $t = \langle q_s, a, f, q_t \rangle \in \Delta$ be a transition, we say that q_s is its *source* and q_t its *target*. A *step* is a triple $q_1 \xrightarrow{a} q_2$ where q_1 and q_2 are states, $a \in \Sigma$ and $(q_1, a, f, q_2) \in \Delta$ for some $f \in 2^F$. Let $u \in \Sigma^* \cup \Sigma^\omega$ be a word. A *path* on u is a sequence of transitions $(t_i)_{i<|u|}$ such that the source of t_{i+1} is the target of t_i for every $i+1 < |u|$, and such that the i-th letter of u is the label of t_i, for every $i < |u|$. An *execution* on u is a path $(t_i)_{i<|u|}$ on u together with a state q, from which the path is to be executed. A *run* on u is an execution starting from a state in Q_0. A run is *accepting* if it sees all acceptance marks of F infinitely often. For $u \in \Sigma^\omega$, $Acc_{\mathscr{A}}(u)$ is the set of accepting runs on u in the automaton $\mathscr{A}$.

Note that several kinds of acceptance conditions for infinite words have been proposed over the years, namely Büchi, Rabin, Streett, Müller, parity. They happen to be equivalent in expressivity (on non-deterministic automata), although some allow more compact automata than others. As explained above, our choice of transition-based generalized Büchi conditions is not arbitrary. But our constructions and results actually do not depend on the type of acceptance condition, and therefore naturally extend to any acceptance flavor. Note however that Büchi are positive: they describe what transitions are to be seen infinitely often. On the other hand, all other types of acceptance conditions describe both required and forbidden sets: transitions to be seen infinitely often, and transitions to be seen *finitely* often. The positive formulation allows simple enumeration algorithms, as needed in Section 4. Furthermore, due to its practical interest for LTL verification, Büchi acceptance conditions have received almost all the attention for the design of efficient verification algorithms. This provides a handful of algorithms to choose from in Section 4.

2.2 Counter Automata

We now equip ω-automata with a finite set of integer-valued *counters*. They are initially set to 0. Transitions in the automaton act on counters, by incrementing, resetting or observing them. Note that the counter values do not constraint the behavior of the automaton (there are no guards on transitions): counters act as *observers* of the behavior. Observation keeps track of the values *encountered* during a run to associate it a *value*. Specifically, the value of a run is the smallest value observed during the run (or ∞ if no observation has been undertaken).

Formally, $\mathbb{C} = \{\mathtt{i}, \mathtt{or}, \mathtt{r}, \tau\}$ denotes the set of counter actions: `i` increments a counter, `r` resets a counter to 0 and `or` *observes* the value of the counter and resets it to 0. τ is a no-op. If a counter has value n, its value after an action $\zeta \in \mathbb{C}$ is noted $n.\zeta$, and we have $n.\tau = n$; $n.\mathtt{i} = n+1$; $n.\mathtt{r} = n.\mathtt{or} = 0$. This operation naturally extends to sequences of counter actions. It is also compatible with $\sim_M$ for $M \in \mathbb{N}$: for $\zeta \in \mathbb{C}$ and $n_1, n_2 \in \mathbb{N}$, if $n_1 \sim_m n_2$ then $n_1.\zeta \sim_m n_2.\zeta$. Thus, the operation of $\mathbb{C}$ on the quotient $\mathbb{N}_\infty / \sim_m$ is well-defined.

Definition 2 A *counter automaton* is a tuple $\mathscr{A} = \langle Q, \Sigma, \Delta, \Gamma, Q_0, F \rangle$ where:

- Q is a finite set of states;
- Σ is a finite alphabet;
- Γ is a finite set of counters;

- F is a finite set of acceptance marks;
- $\Delta \subseteq Q \times \Sigma \times \mathbb{C}^\Gamma \times 2^F \times Q$ is the transition relation;
- $Q_0 \subseteq Q$ is a set of initial states.

A counter automaton is not interpreted as an automaton over the alphabet $\Sigma \times \mathbb{C}^\Gamma$, but as a *cost function* that associates integer values to words. The semantics of a counter automaton is given through its *configuration automaton*, which we describe below. A *configuration* of $\mathscr{A}$ is a triple $\eta = \langle q_1, m_1, \vec{v_1} \rangle$ where $q_1 \in Q$, $m_1 \in \mathbb{N}_\infty$ and $\vec{v_1} \in \mathbb{N}^\Gamma$. q_1 is the state of the configuration η, m_1 its value and $\vec{v_1}$ its counter values. For a configuration $\eta = \langle q_1, m_1, \vec{v_1} \rangle$ and a transition $t = (q_1, a, \vec{\zeta}, f, q_2) \in \Delta$, we write $\eta.t = \langle q_2, m_2, \vec{v_2} \rangle$ the configuration such that $\vec{v_2} = \vec{v_1}.\vec{\zeta}$ and m_2 is the infimum of m_1 and the values of the counters observed by $\vec{\zeta}$: $m_2 = \inf(\{m_1\} \cup \{\vec{v_1}(\gamma) \mid \gamma \in \Gamma \text{ s.t. } \vec{\zeta}(\gamma) = \mathtt{or}\})$. $\left\langle \langle q_1, m_1, \vec{v_1} \rangle, a, \vec{\zeta}, f, \langle q_2, m_2, \vec{v_2} \rangle \right\rangle$ is a transition in the configuration automaton if, and only if, $t = (q_1, a, \vec{\zeta}, f, q_2) \in \Delta$ for some $f \in 2^F$ and $\langle q_1, m_1, \vec{v_1} \rangle .t = \langle q_2, m_2, \vec{v_2} \rangle$. The initial states in the configuration automaton are the configurations $\eta = \left\langle q_0, \omega, \vec{0} \right\rangle$ with $q_0 \in Q_0$. Note that the configuration automaton has the same set of acceptance marks F as its initial counter automaton.

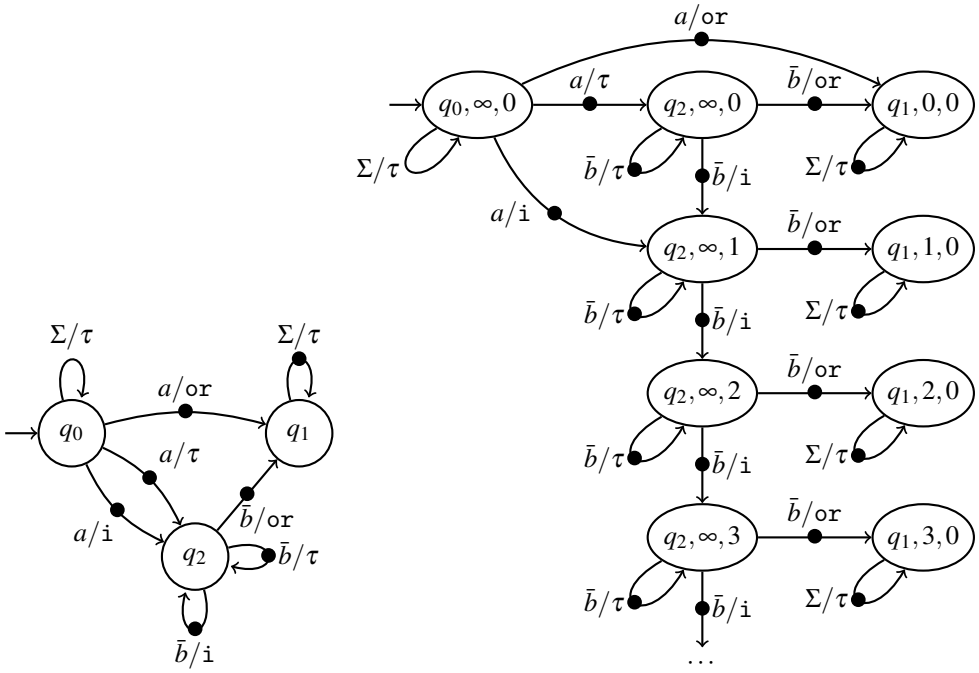
Let $\rho = ((t_i)_{i \in \mathbb{N}}, \eta_0)$ be an infinite run, let $\eta_{i+1} = \eta_i.t_i$ and $\eta_i = \langle q_i, m_i, \vec{v_i} \rangle$ for every $i \in \mathbb{N}$. The sequence $(m_i)_{i \in \mathbb{N}}$ is nonnegative and decreasing. Thus, it eventually stabilizes to a value $Val(\rho)$, the value of the run ρ. $\mathscr{A}$ thus defines a *cost function* $[\![\mathscr{A}]\!]$ that associates to each word $u \in \Sigma^\omega$ the value $[\![\mathscr{A}]\!](u) = \sup_{\rho \in Acc_\mathscr{A}(u)} Val(\rho)$.

We are interested in this paper in finding the *bound* over all words of a counter automaton, *i.e.* $\sup_{u \in \Sigma^\omega} [\![\mathscr{A}]\!](u)$. By definition of $[\![\mathscr{A}]\!]$, this bound is also $\sup_{\rho \in Acc_\mathscr{A}(\Sigma^\omega)} Val(\rho)$, the supremum value of all accepting runs in $\mathscr{A}$. We also write $\sup[\![\mathscr{A}]\!]$ to ease the notation.

Definition 3 The *sup-bound value problem* asks, for an input automaton $\mathscr{A}$, the value in $\mathbb{N}_\infty$ of $\sup[\![\mathscr{A}]\!]$.

Strictly speaking, information on the counters are all pushed in the states of the configuration automaton, so that there is no need to keep counter actions on its transition (although keeping them helps understanding the link with its initial counter automaton). Thus, a configuration automaton can rather be seen as a plain automaton, together with a *value function* that associates integers to states. Since the value of configurations decrease along a run, all configuration in an SCC have the same value. Thus, the value function may be defined on maximal SCCs instead of states. Our bound search problem can be thus restated as searching for the highest value among accepting SCCs of a (plain) automaton. Note however that at this point, there is no guarantee that the configuration automaton is finite.

Example 1 *Figure 1a presents an example of Counter Automaton. Non-deterministic, it features a single counter, and associates to a word the maximal distance between a a and the next b. $\bar{b}$ stands for any letter but b. The automaton has a single acceptance mark, represented by a black bullet on transitions. An accepting run necessarily leaves the initial state q_0 on a letter a. After leaving q_0, a b may be read only in state q_1. Thus, the automaton guesses the positions of the a and its next b that are the most distant in the input word. The maximum over accepting runs,*



(a) Example of Counter Automaton (b) A partial view of its configuration automaton

Figure 1: An example of counter automaton and its (partial) configuration automaton

in the definition of $[\![\mathscr{A}]\!]$*, eliminates wrong guesses. Note for example that* $[\![\mathscr{A}]\!]((a^k b)^\omega) = k-1$ *for every* $k \in \mathbb{N}$*, and* $[\![\mathscr{A}]\!](aba^2ba^3ba^4b\dots a^nba^{n+1}b\dots) = \infty$*, so that* $\sup[\![\mathscr{A}]\!]$ *is not finite.*

Such an automaton can be used to observe *the behavior of another one. For instance, when studying an autonomous robot, say event* a *represents the consumption of one unit of energy, and* b *represents a full recharge of the battery. A synchronized product between the automaton representing the behaviors of the robot and our example automaton produces a new counter automaton* $\mathscr{A}'$*, and* $\sup[\![\mathscr{A}']\!]$ *represents the maximal quantity of energy consumed between two recharges of the battery. Computing this bound can thus be used to properly calibrate the battery.*

We also show on Figure 1b the first states of the corresponding (infinite) configuration automaton. A configuration is a triple whose first component is a state of the original counter automaton, second component is the current value of the run, and third component is the current value of the counter. Note how counter actions act on the counter value and on the run value.

3 From the bound value problem to generalized emptiness check

The answer to the sup-bound value problem in is $\mathbb{N}_\infty$, which raises the question of whether the sought value is finite or infinite. The latter question is known as the *boundedness problem*. It has already been investigated and shown decidable [Col09]. It seems quite natural that $\sup[\![\mathscr{A}]\!]$ can be obtained from the configuration automaton, which is possibly infinite. We present in this

section a reduction of the configuration automaton to a finite one, based on an analysis of the boundedness problem.

3.1 Boundedness

The main problem towards finding $\sup[\![\mathscr{A}]\!]$ is whether this supremum is finite or infinite, which we call the *boundedness* problem, known to be decidable [Col09].

For $M \in \mathbb{N}$, we extend the relation $\sim_M$ to configurations: $\langle q_1, m_1, \vec{v_1}\rangle \sim_M \langle q_2, m_2, \vec{v_2}\rangle$ iff $q_1 = q_2$, $m_1 \sim_M m_2$ and $\vec{v_1} \sim_M \vec{v_2}$, where $\sim_M$ is extended component-wise to vectors of integers.

Lemma 1 *Let σ be a finite path, and $M \in \mathbb{N}$. Let η_1 and η_2 be two configurations such that $\eta_1 \sim_M \eta_2$. Then $\eta_1.\sigma \sim_M \eta_2.\sigma$.*

Proof. This lemma is the consequence of the compatibility of both $<$ and the action of $\mathbb{C}$ with $\sim_M$: for every transition t and configurations η_1, η_2, if $\eta_1 \sim_M \eta_2$, then $\eta_1.t \sim_M \eta_2.t$. We then conclude by induction on the size of σ. □

We restate a result first established (although in a slightly different setting) by [Kup14]. We also provide a direct proof.

Proposition 1 *Let Θ be the length of the longest run (i.e. starting from an initial state) that visits every state at most once in $\mathscr{A}$, plus 1. The following three are equivalent:*

(a) $\sup[\![\mathscr{A}]\!] = \omega$.

(b) $\sup[\![\mathscr{A}]\!] \geq \Theta$.

(c) *$\mathscr{A}$ has an accepting ultimately periodic run in which, for every $\gamma \in \Gamma$, every occurrence of* `or` *for γ is preceded by a cycle that increments γ at least once and does not reset γ.*

Proof. Obviously, (a) implies (b), and (c) implies (a). We prove that (b) implies (c). Assume that $\sup_{u\in\Sigma^\omega}[\![\mathscr{A}]\!](u) \geq \Theta$, and consider an accepting run ρ such that $Val(\rho) \geq \Theta$. We first build an accepting lasso from ρ, and we will then show that it satisfies (c).

Consider the sequence $(\eta_i = \langle q_i, m_i, \vec{v_i}\rangle)_{i\in\mathbb{N}}$ of configurations visited by ρ, and let $\rho_{/\sim_\Theta} = (\langle q_i, [m_i]_{\sim_\Theta}, [\vec{v_i}]_{\sim_\Theta}\rangle)_{i\in\mathbb{N}}$. By definition of ρ, for every $i \in \mathbb{N}$, $m_i \geq \Theta$, so that $[m_i]_{\sim_\Theta} = [\Theta]_{\sim_\Theta}$. The $[\vec{v_i}]_{\sim_\Theta}$'s have a finite number of possible values, and Q is finite, so there is some $i_0 \in \mathbb{N}$ such that $\langle q_{i_0}, [\Theta]_{\sim_\Theta}, [\vec{v_{i_0}}]_{\sim_\Theta}\rangle$ occurs infinitely often in $\rho_{/\sim_\Theta}$. Consider $\pi = \rho(0)\dots\rho(i_0-1)$ and $\sigma = \rho(i_0)\dots\rho(j)$ for some $j > i_0$ such that $\langle q_j, m_j, \vec{v_j}\rangle \sim_\Theta \langle q_{i_0}, m_{i_0}, \vec{v_{i_0}}\rangle$ and all acceptance marks in F appear between positions i_0 and j. Such a σ exists because ρ is accepting. Let ρ' be the run $\pi\sigma^\omega$. By definition of π and σ, ρ' is an accepting run of $\mathscr{A}$. Furthermore, $\eta_{i_0}.\sigma \sim_\Theta \eta_{i_0}$ by definition of σ. By Lemma 1, a simple induction shows that $\eta_{i_0}.\sigma^k \sim_\Theta \eta_{i_0}$ for every $k \in \mathbb{N}$. Therefore $Val(\rho') \sim_\Theta Val(\rho)$, and $Val(\rho') \geq \Theta$.

Let $\gamma \in \Gamma$ and consider an occurrence of `or` for γ in ρ', say at position i. $Val(\rho') \geq \Theta$ implies that the value of γ is greater than or equal to Θ when observed, so that this occurrence of `or` for γ is preceded by at least Θ increments of γ. A transition carries at most one action for γ, so there are Θ positions $j_0 < \dots < j_{\Theta-1}$ before i in ρ' that increment γ. Obviously, γ is never reset between

positions j_0 and i. By definition of Θ, among the source states of the transitions $(\rho'(j_k))_{k<\Theta}$, there is one that occurs twice, say at positions j_p and j_q ($p<q$). Thus, $\rho'(j_p)\dots\rho'(j_q-1)$ is a cycle in $\mathscr{A}$ that occurs before position i, increments γ at least once and never resets γ. Such a cycle exists whatever the occurrence of `or` for γ in ρ', and whatever γ. Therefore, the accepting ultimately periodic run ρ' satisfies condition (c). □

Example 2 *We illustrate Proposition 1 on the automata of Figure 1. The longest run without loop in the automaton of Figure 1a is* $q_0 \to q_2 \to q_1$, *so that* $\Theta = 3$. *A run* ρ *of value greater than* 3 *necessarily goes through state* q_2, *passes at least once through the loop* $q_2 \xrightarrow{\bar{b}/\mathtt{i}} q_2$, *and goes to state* q_1 *through transition* $q_2 \xrightarrow{\bar{b}/\mathtt{or}} q_1$. *By repeating* n *times the loop* $q_2 \xrightarrow{\bar{b}/\mathtt{i}} q_2$, *one builds a new accepting run* ρ' *of value* $Val(\rho') = Val(\rho)+n$. *Thus* $\mathscr{A}$ *has accepting runs of arbitrarily high values, which shows that* $\sup[\![\mathscr{A}]\!]$ *is infinite. The key to this construction is the third item of Proposition 1: there must be an incrementing cycle to be repeated before* every *operation* `or`.

In this example, Proposition 1 also holds for $\Theta = 2$. *However, runs of value* 0 *or* 1 *are not guaranteed to go through the incrementing loop* $q_2 \xrightarrow{\bar{b}/\mathtt{i}} q_2$, *and can be the base of the above construction of accepting of arbitrarily high values.*

3.2 Reduction to a finite automaton

Proposition 1 simplifies our problem: it suffices to search $[\sup[\![\mathscr{A}]\!]]_{\sim_\Theta}$. Recall that the quotient $\mathbb{N}_\infty/\sim_\Theta$ is totally ordered, and that $[m]_{\sim_\Theta} < [n]_{\sim_\Theta}$ iff $m<n$ and $m<\Theta$. Thus, $[\sup_{u\in\Sigma^\omega}[\![\mathscr{A}(u)]\!]]_{\sim_\Theta} = \sup_{u\in\Sigma^\omega}[[\![\mathscr{A}]\!](u)]_{\sim_\Theta}$. Similarly, for $u\in\Sigma^\omega$, $[[\![\mathscr{A}]\!](u)]_{\sim_\Theta} = \sup_{\rho\in Acc_{\mathscr{A}}(u)}[Val(\rho)]_{\sim_\Theta}$.

We define the *configuration automaton capped at* Θ of $\mathscr{A}$ as follows:

- $(Q\times\mathbb{N}_\infty\times\mathbb{N}^\Gamma)_{/\sim_\Theta}$ is the set of states;
- $(Q_0\times\{\omega\}\times\left\{\vec{0}\right\})_{/\sim_\Theta}$ is the set of initial states;
- $\tilde{\eta}\xrightarrow{t}[\eta.t]_{\sim_\Theta}$ for every configuration $\eta\in\tilde{\eta}$ and every transition t. Lemma 1 ensures that this transition relation is well-defined, as $[\eta.t]_{\sim_\Theta}$ in fact does not depend on the choice of η in the class $\tilde{\eta}$. The acceptance marks of t in the capped configuration automaton are the same as in the original automaton.

Note that the constructed automaton is finite since Θ is finite. In the limit case where $\Theta=\omega$, the capped configuration automaton would be the original configuration automaton.

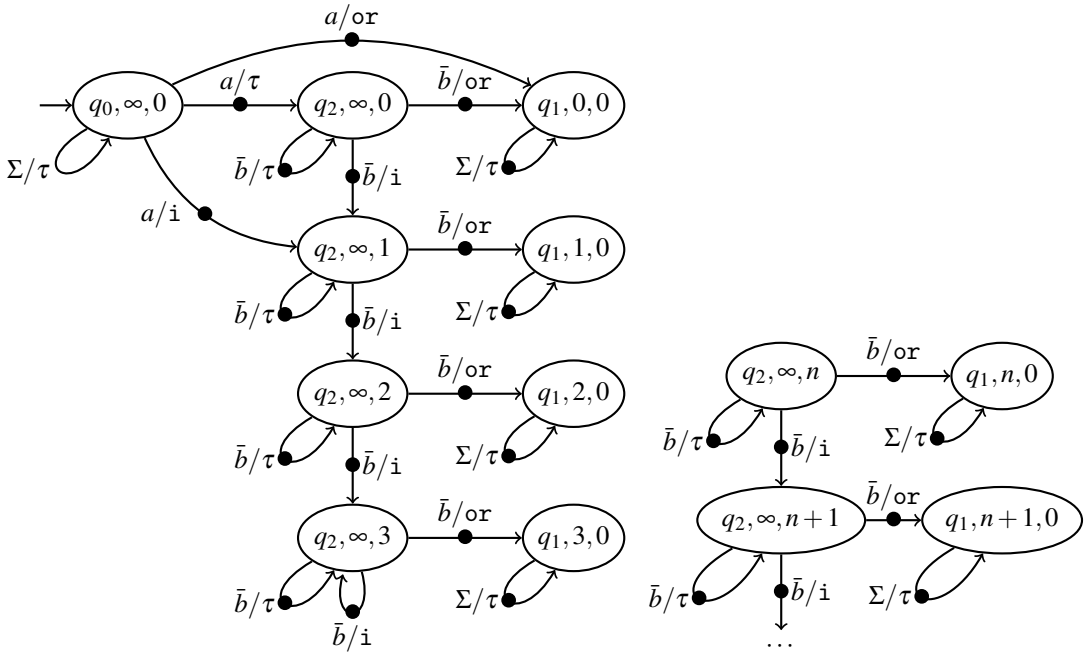
We define the value of a run as in the capped configuration automaton. Let $\rho=((t_i)_{i\in\mathbb{N}},\tilde{\eta}_0)$ be an infinite run, $\tilde{\eta}_{i+1}=\tilde{\eta}_i.t_i$ and $\tilde{\eta}_i=\langle\tilde{q}_i,\tilde{m}_i,\tilde{\vec{v}}_i\rangle$ for every $i\in\mathbb{N}$. The sequence $(\tilde{m}_i)_{i\in\mathbb{N}}$ is decreasing in the finite set $\mathbb{N}_\infty/\sim_\Theta$. Thus, it eventually stabilizes to a value noted $Val_{\sim_\Theta}(\rho)\in\mathbb{N}_\infty/\sim_\Theta$, the value of the run ρ in the capped configuration automaton.

Proposition 2 *For every accepting run* ρ *of* $\mathscr{A}$, *there is an accepting run* ρ' *in the capped configuration automaton such that* $Val_{\sim_\Theta}(\rho')=[Val(\rho)]_{\sim_\Theta}$. *Conversely, for every run* ρ' *in the capped configuration automaton, there is a run* ρ *of* $\mathscr{A}$ *such that* $Val_{\sim_\Theta}(\rho')=[Val(\rho)]_{\sim_\Theta}$.

Proof. ($\Rightarrow$) Let $\rho=\langle(t_i)_{i\in\mathbb{N}},\eta_0\rangle$ be an accepting run of $\mathscr{A}$. We write $\eta_{i+1}=\eta_i.t_i$ and $\eta_i=\langle q_i,m_i,\vec{v}_i\rangle$ for every $i\in\mathbb{N}$. By definition, $[\eta_0]_{\sim_\Theta}$ is an initial state of the capped configuration

automaton, and $[\eta_i]_{\sim_\Theta} \xrightarrow{t_i} [\eta_{i+1}]_{\sim_\Theta}$ is a step of the capped configuration automaton for every $i \in \mathbb{N}$. Thus, the $\rho' = \langle (t_i)_{i\in\mathbb{N}}, [\eta_0]_{\sim_\Theta} \rangle$ is a run in the capped configuration automaton. It is not hard to see that it is accepting iff ρ is accepting. Recall that the sequence $(m_i)_{i\in\mathbb{N}}$ converges to the value $Val(\rho)$, so that the sequence $([m_i]_{\sim_\Theta})_{i\in\mathbb{N}}$ converges to $[Val(\rho)]_{\sim_\Theta}$. But the limit of this sequence is also $Val_{\sim_\Theta}(\rho')$ by definition, so that $Val_{\sim_\Theta}(\rho') = [Val(\rho)]_{\sim_\Theta}$.

($\Leftarrow$) Let $\rho' = \langle (t_i)_{i\in\mathbb{N}}, \tilde{\eta}_0 \rangle$ be an accepting run of the capped configuration automaton. We call q_0 the state of the configuration $\tilde{\eta}_0$, $\eta_0 = \langle q_0, \infty, 0 \dots 0 \rangle$ and define $\rho = \langle (t_i)_{i\in\mathbb{N}}, q_0 \rangle$. The definition of the transition relation of the capped configuration automaton guarantees that ρ is a run of $\mathscr{A}$ (all steps are well-defined, and q_0 is an initial state of $\mathscr{A}$). It is also immediate that ρ is accepting iff ρ' is accepting. Let $\eta_{i+1} = \eta_i.t_i$ and $\tilde{\eta}_{i+1} = \tilde{\eta}_i.t_i$ for every $i \in \mathbb{N}$. Obviously $\eta_0 \in \tilde{\eta}_0$, and Lemma 1 then guarantees that $\eta_i \in \tilde{\eta}_i$ for every $i \in \mathbb{N}$. This last property ensures that the limit $Val(\rho)$ belongs to the equivalence class $Val_{\sim_\Theta}(\rho')$: $Val_{\sim_\Theta}(\rho') = [Val(\rho)]_{\sim_\Theta}$. □



(a) Example of capped configuration automaton

(b) The pattern in the configuration automaton abstracted away by the capped configuration automaton

Figure 2: Illustration of the capped configuration automaton construction

Example 3 *Figure 2a shows the capped configuration automaton of the automaton of Figure 1a. Its configuration automaton indeed exhibits the pattern of Figure 2b, a pattern abstracted two nodes of the capped configuration automaton. Following Proposition 1 (and its illustration in the previous example), the capped configuration automaton does not keep track of counter values beyond* $\Theta = 3$ *as this value suffices to build runs with arbitrarily high values. The exact behaviors of the original automaton must be kept for values below* Θ*, as they do not exhibit a pattern to be*

abstracted away.

4 SCC enumeration to find the bound

We now reformulate the sup-bound value problem, in order to provide algorithms to solve it. Suppose we are given an ω-automaton, along with a function that associate integer values to its maximal Strongly Connected Components (SCC). Our problem is to find the largest values among accepting maximal SCCs, which we call the *largest accepting value problem*. We will consider only maximal SCC, abusively called SCC in the remaining of the paper.

This problem generalizes the classical emptiness problem for Büchi automata. In the latter context, all SCCs have the same value $\top$, and $\bot$ denotes the neutral element for the operation sup, so that $\sup \emptyset = \bot$. The largest value among accepting SCCs is thus $\top$ if the language of the automaton is not empty, and $\bot$ otherwise.

To solve the largest accepting value problem, we naturally propose to take inspiration from classical Büchi emptiness check algorithms. Two families of emptiness check algorithms can be distinguished: those based on Nested Depth-First Search (NDFS), and those based on SCC enumeration. We focus on the latter category for two reasons. First, SCC enumeration matches exactly the definition of the largest accepting value problem, and thus seems more natural. Second, SCC enumeration-based algorithms are more suited to generalized Büchi conditions. While the nesting depth of DFS walks in NDFS-based algorithms depends on the number of acceptance conditions, SCC enumeration-based algorithms deal with generalized acceptance conditions in a single walk of the input automaton, which is a great advantage towards efficiency.

SCC-based emptiness checks enumerates the SCC of the input automaton, keeping track of the accepting conditions to detect accepting ones. As shown in algorithm 1, they typically return as soon as an accepting SCC is detected, but are in fact able to enumerate all accepting SCCs. The adaptation of such algorithms to the largest accepting value problem is straightforward, and is shown in algorithm 2. The algorithm keeps track at every moment of the largest accepting value encountered so far, and updates it whenever a new accepting SCC is found. At the end of the enumeration, the largest accepting value is known.

Algorithm 1: Typical SCC enumeration-based emptiness check algorithm

```
   // Based on a SCC
      enumeration keeping track
      of acceptance conditions
1  foreach maximal SCC s do
2      if s is accepting then
3          return "not empty"
4  return "empty"
```

Algorithm 2: SCC enumeration-based largest accepting value algorithm

```
   // ν is the SCC value
      function
1  value ← −∞
2  foreach maximal SCC s do
3      if s is accepting then
4          value ← max(value, ν(s))
5  return value
```

4.1 A smaller automaton

Informally, Proposition 1 indicates that there is no need to keep track of the exact run values above Θ to compute $\sup[\![\mathscr{A}]\!]$. Similarly, as the value of a single run ρ is the smallest one among the counter values observed in ρ, there is no need to keep track of the exact values of counters beyond the current run value. Once a value n has been observed, all counter values larger than n can be dismissed as they have no impact on the value of the run. We thus propose a slight improvement of our capped configuration automaton.

Formally, we first define a new equivalence relation over configurations: for $M \in \mathbb{N}$, $\langle q_1, m_1, \vec{v_1}\rangle \approx_M \langle q_2, m_2, \vec{v_2}\rangle$ iff $q_1 = q_2$, $m_1 \sim_M m_2$ and $\vec{v_1} \sim_{\min(m_1,M)} \vec{v_2}$. Note that the action of $\mathbb{C}$ on configurations is compatible with $\approx_\Theta$, so that we have an analogous of Lemma 1.

Lemma 2 *Let σ be a finite path, and $M \in \mathbb{N}$. Let η_1 and η_2 be two configurations such that $\eta_1 \approx_M \eta_2$. Then $\eta_1.\sigma \approx_M \eta_2.\sigma$.*

We define the *double capped configuration automaton* of $\mathscr{A}$ as:

- $(Q \times \mathbb{N}_\infty \times \mathbb{N}^\Gamma)_{/\approx_\Theta}$ is the set of states;
- $(Q_0 \times \{\omega\} \times \left\{\vec{0}\right\})_{/\approx_\Theta}$ is the set of initial states;
- $[\eta]_{\approx_\Theta} \xrightarrow{a} [\eta']_{\approx_\Theta}$ whenever $\eta_1 \xrightarrow{a} \eta_2$ is a step for $\mathscr{A}$.

Lemma 2 ensures that this transition relation is well-defined. Once again, current values of runs are positive are decreasing, and thus eventually stabilizes. This stabilization value defines the value $Val_{\approx_\Theta}(\rho)$ of a run in the double capped configuration automaton.

Proposition 3 *For every accepting run ρ of $\mathscr{A}$, there is an accepting run ρ' in the double capped configuration automaton such that $Val_{\approx_\Theta}(\rho') = [Val(\rho)]_{\approx_\Theta}$. Conversely, for every run ρ' in the capped configuration automaton, there is a run ρ of $\mathscr{A}$ such that $Val_{\approx_\Theta}(\rho') = [Val(\rho)]_{\approx_\Theta}$.*

We do not detail the proof of Proposition 3 as it does not really differ from the proof of Proposition 2.

4.2 A note on complexity

All in all, $\sup[\![\mathscr{A}]\!]$ can be computed by enumerating the accepting SCC of either the capped configuration automaton or the double capped configuration automaton. Many variants have been proposed, sequential or parallel, based either on Dijkstra's or Tarjan's SCC enumeration algorithms, to handle simple or generalized, state-based or transition-based Büchi acceptance conditions. It is not the scope of this paper to detail all existing algorithms and their variants, and we refer the reader to [RDKP13] for a survey. SCC enumeration algorithms have time and space complexity related (mostly linear) to the size of the input automaton (in terms of number of transitions and/or number of states). For the sake of generality, we estimate the size of the both capped configuration automata, rather than the complexity of a specific algorithm.

We denote by $|\mathscr{A}|_S$ the number of states of $\mathscr{A}$, and by $|\mathscr{A}|_T$ its number of transitions. In the capped configuration automaton, each state and each transition of $\mathscr{A}$ is duplicated at most $(\Theta+1)^{|\Gamma|+1}$ times. The capped configuration automaton is thus at most $(\Theta+1)^{|\Gamma|+1}$ larger than

$\mathscr{A}$. We recall that Θ depends on the graph structure of $\mathscr{A}$ only, not on the transition labels (letters and counter actions).

The ratio is slightly better for the double capped configuration automaton, as each state and each transition of the initial counter automaton is duplicated at most $\Sigma_{i=1}^{\Theta+1} i^{|\Gamma|}$. The latter sum is a polynomial in Θ of degree $|\Gamma|+1$, whose coefficient of highest degree is $\frac{1}{|\Gamma|+1}$. For a fixed $|\Gamma|$, the double capped configuration automaton is asymptotically smaller by a factor $|\Gamma|+1$ than the capped configuration automaton. This is a mild compensation of the size exponential in $|\Gamma|$.

It is easy to see that $\Theta \leq |\mathscr{A}|_S$, which gives in the worst-case a capped configuration automaton with $|\mathscr{A}|^{|\Gamma|+2}$ states. Yet, expressing the size ratio in terms of Θ is much finer. Observe that the capped and double capped configuration automaton construction are still correct when Θ is over-approximated: unnecessary values may be tracked, which only impacts efficiency. For best performance, knowing the exact value of Θ is thus desirable. Yet, Θ can be computed by enumerating the SCC of $\mathscr{A}$, i.e. in time linear in $|\mathscr{A}|$, a simple pre-computation dominated by the complexity of the subsequent search for the bound.

4.3 Improvement and optimizations

The (double) capped configuration automaton has a very specific structure that hints for several possible optimizations. The first feature to exploit is the fact that the current run value decreases along a run. A change of the current run value guarantees that a new SCC is entered. Such information, usually not accessible in the general setting of SCC enumeration, can be taken advantage of. When a new SCC is entered, say at state s, the exploration of the sub-automaton reachable from s does not need any information about the path from the root. This enables a recursive enumeration of the sub-automaton, with s as its initial state. Such recursive enumeration is particularly interesting for parallelism.

The second feature to exploit is the fact that a SCC value (current run value) is computable from a single state, and does not necessitate the whole maximal SCC to be known. Recall that the algorithm searches for the largest value among the accepting SCCs, by enumerating the SCC and maintaining the largest SCC value, say α, encountered so far. Those SCC whose values are smaller than the current α need not be explored, and can be pruned from the enumeration. This can considerably reduce the size of the automaton to be explored. A simple heuristic may further be used: when such a choice occurs, the algorithm should explore the SCC with the larger value first. If it happens to be a accepting SCC, it yields a larger α and maximized the previous pruning optimization.

5 Related Work

A famous problem in language theory is the star-height problem: given a language L (of finite words) and an integer k, is there a regular expression for L with at most k nested Kleene stars? Proposed in 1963 [Egg63], it was proven decidable in 1988 [Has88] by exhibiting an algorithm with non-elementary complexity, and a much more efficient algorithm was then proposed in 2005 [Kir05]. Both algorithms translate the problem to the existence of a bound for a function mapping words to integers, represented in both cases by an automaton equipped with counters

(distance automata for the former, nested distance desert automata for the latter). This problem of the existence of a bound is then shown decidable. This is the first of many problems that reduce to the existence of a bound for such automata.

This motivated an in-depth study of automata with counters (as we use it) as a general framework, that came up with a theory extending the one of regular languages, with logical and algebraic counter-parts [Col09]. On infinite words, the logical counter-part motivated the introduction and study of Cost Linear Temporal Logics [KB12], an extension of LTL able to count discrete events. This theory also encompasses *promptness* properties, a variant of liveness where a bound on the wait time of a recurring event must exist [KPV07, AHK10]. But all these works, motivated by the boundedness problem, overlook the exact values of the functions. On one hand, this relaxation enables nice closure properties (such as the equivalent expressiveness for inf-automata and sup-automata). On the other hand, it only allows to reason about the existence of a bound, not to compute values.

In verification, not all questions have a boolean answer, so that various quantitative extensions of automata have been considered, such as weighted automata (see [DG07] for a survey). Despite their various domains of application, they have limited expressivity, as the domain of weights is required to be a semi-ring. An extension to arbitrary operations on weights have been recently proposed [ADD+13]. It encompasses various extensions of weighted automata, such as Discounted Sum Automata [AHM03] and Counter ω-Automata as considered in this paper. All these formalisms can be characterized by the absence of guards on register values. These extensions sometimes have equivalent logics (such as discounted linear temporal logics [ABK14] or counting LTL [LMP10]). We have already studied the problem of computing bounds for the aforementioned Cost Linear Temporal Logics [CRB15]. The present work on Counter Automaton is an extension of our previous work on CLTL, as Counter Automata are more expressive than CLTL.

Most of the works cited above only focus on expressivity, decidability and complexity problems, with little consideration to the practical use of such quantitative extensions of automata. This situation contrasts with older formalisms: ω-automata have already received great focus towards practical applications, illustrated by numerous emptiness checks algorithms (see [SE05] or [RDKP13] for an overview) and many implementations, principally oriented towards LTL model-checking (see [RV10] for a survey). Some quantitative extensions of automata possess a similar maturity towards practical applications, especially timed automata [BDL+06] and weighted automata [KNP11].

6 Conclusion

We have presented in this paper a method to compute the sup-bound value problem on Counter ω-automata. It is based on a reduction of the configuration automaton to a finite automaton, thanks to previous results on the boundedness of such automata. On this finite automaton, the sup-bound value problem is translated into a generalization of the emptiness check problem, as a single-player game with (generalized) Büchi objectives. This problem is itself solved by enumerating accepting SCCs of the finite automaton. Such enumeration algorithms being used for instance for ω-automata emptiness checks, our approach thus builds upon methods already

tried and tested for qualitative verification. To our knowledge, it is the first time that such a method for Counter Automata is proposed. It opens perspectives for the practical use of Counter ω-automata towards system verification.

Future works include two axis. First, the method we have presented should now be implemented, in order to perform experimental evaluation. Second, we believe that there is still rooms for optimization of the algorithms, and for heuristics that would increase performance. The setting of the intermediate problem of a single-player game with Büchi objectives would be an appropriate setting for such a study.

Bibliography

[ABK14] S. Almagor, U. Boker, O. Kupferman. Discounting in LTL. In Ábrahám and Havelund (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS 8413, pp. 424–439. Springer Berlin Heidelberg, 2014.

[ADD+13] R. Alur, L. Dantoni, J. Deshmukh, M. Raghothaman, Y. Yuan. Regular Functions and Cost Register Automata. In *Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium on*. Pp. 13–22. 2013.

[AHK10] S. Almagor, Y. Hirshfeld, O. Kupferman. Promptness in ω-Regular Automata. In *Proc. 8th International Symposium on Automated Technology for Verification and Analysis (ATVA'10)*. LNCS 6252, pp. 22–36. Springer, 2010.

[AHM03] L. de Alfaro, T. Henzinger, R. Majumdar. Discounting the Future in Systems Theory. In Baeten et al. (eds.), *Automata, Languages and Programming*. Lecture Notes in Computer Science 2719, pp. 1022–1037. Springer Berlin Heidelberg, 2003.

[BC06] M. Bojańczyk, T. Colcombet. Bounds in ω-Regularity. In *Proc. 21st Annual IEEE Symposium on Logic in Computer Science*. LICS '06, pp. 285–296. IEEE Computer Society, Washington, DC, USA, 2006.

[BDL+06] G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, M. Hendriks. UPPAAL 4.0. In *Proc. 3rd International Conference on the Quantitative Evaluation of Systems*. QEST '06, pp. 125–126. IEEE Computer Society, Washington, DC, USA, 2006.

[Col09] T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming*. Pp. 139–150. Springer, 2009.

[CRB15] M. Colange, D. Racordon, D. Buchs. A CEGAR-like Approach for Cost LTL Bounds. Technical report, CoRR, June 2015.
http://arxiv.org/abs/1506.05728

[DG07] M. Droste, P. Gastin. Weighted Automata and Weighted Logics. *Theoretical Computer Science* 380(1):69–86, 2007.

[DG12] S. Demri, P. Gastin. Specification and Verification using Temporal Logics. In D'Souza and Shankar (eds.), *Modern applications of automata theory*. IISc Research Monographs 2, chapter 15, pp. 457–494. World Scientific, July 2012.

[Egg63] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.* 10(4):385–397, 12 1963.

[Has88] K. Hashiguchi. Algorithms for determining relative star height and star height. *Information and Computation* 78(2):124 – 169, 1988.

[KB12] D. Kuperberg, M. V. Boom. On the expressive power of cost logics over infinite words. In *Automata, Languages, and Programming*. Pp. 287–298. Springer, 2012.

[Kir05] D. Kirsten. Distance desert automata and the star height problem. *RAIRO-Theoretical Informatics and Applications* 39(03):455–509, 2005.

[KNP11] M. Kwiatkowska, G. Norman, D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Gopalakrishnan and Qadeer (eds.), *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. LNCS 6806, pp. 585–591. Springer, 2011.

[KPV07] O. Kupferman, N. Piterman, M. Vardi. From liveness to promptness. In Damm and Hermanns (eds.), *Proc. 19th International Conference on Computer Aided Verification (CAV'07)*. LNCS 4590, pp. 406–419. Springer, 2007.

[Kup14] D. Kuperberg. Linear Temporal Logic for Regular Cost Functions. *Logical Methods in Computer Science* 10(1), 2014.

[LMP10] F. Laroussinie, A. Meyer, E. Petonnet. Counting LTL. In *Proc. 2010 17th International Symposium on Temporal Representation and Reasoning*. TIME '10, pp. 51–58. IEEE Computer Society, Washington, DC, USA, 2010.

[RDKP13] E. Renault, A. Duret-Lutz, F. Kordon, D. Poitrenaud. Three SCC-based Emptiness Checks for Generalized Büchi Automata. In McMillan et al. (eds.), *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'13)*. Lecture Notes in Computer Science 8312, pp. 668–682. Springer, Dec. 2013.

[RV10] K. Rozier, M. Vardi. LTL Satisfiability Checking. *International journal on software tools for technology transfer* 12(2):123–137, 2010.

[SE05] S. Schwoon, J. Esparza. A Note on On-The-Fly Verification Algorithms. In Halbwachs and Zuck (eds.), *Proceedings of the 11th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'05)*. Lecture Notes in Computer Science 3440, pp. 174–190. Springer, Edinburgh, Scotland, UK, Apr. 2005.