# Energy Büchi Problems

Sven Dziadek, Uli Fahrenberg, Philipp Schlehuber-Caissier

# Energy Büchi Problems *

Sven Dziadek[0000−0001−6767−7751], Uli Fahrenberg[0000−0001−9094−7625], and Philipp Schlehuber-Caissier[0000−0002−6611−9659]

EPITA Research Laboratory (LRE), Paris, France



**Abstract.** We show how to efficiently solve energy Büchi problems in finite weighted automata and in one-clock weighted timed automata. Solving the former problem is our main contribution and is handled by a modified version of Bellman-Ford interleaved with Couvreur's algorithm. The latter problem is handled via a reduction to the former relying on the corner-point abstraction. All our algorithms are freely available and implemented in a tool based on the open-source platforms TChecker and Spot.

**Keywords:** weighted timed automaton; weighted automaton; energy problem; generalized Büchi acceptance; energy constraints

## 1 Introduction

Energy problems in weighted (timed) automata pose the question whether there exist infinite runs in which the accumulated weights always stay positive. Since their introduction in [7], much research has gone into different variants of these problems, for example energy games [12, 16, 27], energy parity games [11], robust energy problems [2], etc., and into their application in embedded systems [17, 19], satellite control [5, 25], and other areas. Nevertheless, many basic questions remain open and implementations are somewhat lacking.

The above results discuss *looping* automata [28], *i.e.,* $\omega$-automata in which all states are accepting. In practice, looping automata do not suffice because they cannot express all liveness properties. For model checking, formal properties (*e.g.,* in LTL) are commonly translated into (generalized) Büchi automata [9] that provide a simple model for the larger class of $\omega$-regular languages.

In this work, we extend energy problems with transition-based generalized Büchi conditions and treat them for weighted automata as well as weighted timed automata with precisely one clock. On weighted automata we show that they are effectively decidable using a combination of a modified Bellman-Ford algorithm with Couvreur's algorithm. For weighted timed automata we show that one can use the corner-point abstraction to translate the problem to weighted (untimed) automata.

For looping automata, the above problems have been solved in [7]. (This paper also treats energy games and so-called universal energy problems, both of

* Partially funded by ANR project Ticktac (ANR-18-CE40-0015)

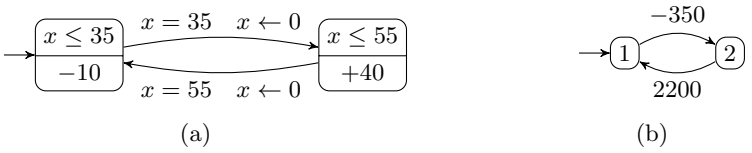



Fig. 1: Satellite example: two representations of the base circuit. (a) as weighted timed automaton $A$; (b) as a (finite) weighted automaton.

which are of no concern to us here.) While we can re-use some of the methods of [7] for our Büchi-enriched case, our extension is by no means trivial. First, in the setting of [7] it suffices to find *any* reachable and energy positive loop; now, our algorithm must consider that such loops might not be accepting in themselves but give access to new parts of the automaton which are. Secondly, [7] mostly treat the energy problem with unlimited upper bound, whereas we consider that energy has a ("weak") upper bound beyond which it cannot increase. [7] claim that the weak-upper-bound problem can be solved by slight modifications to their solution of the unbounded problem; but this is not the case. For example, the typical Bellman-Ford detection of positive cycles might not work when the energy levels attained in the previous step are already equal to the upper bound.

As a second contribution, we have implemented all of our algorithms in a tool based on the open-source platforms TChecker[1] [22] and Spot[2] [13] to solve generalized energy Büchi problems for one-clock weighted timed automata. We first employ TChecker to compute the zone graph and then use this to construct the corner-point abstraction. This in turn is a weighted (untimed) generalized Büchi automaton, in which we also may apply a variant of Alur and Dill's Zeno-exclusion technique [1]. Finally, our main algorithm to solve generalized energy Büchi problems on weighted finite automata is implemented using a fork of Spot. Our software is available at `https://github.com/PhilippSchlehuberCaissier/wspot`.

In our approach to solve the latter problem, we do not fully separate the energy and Büchi conditions (contrary to, for example, [11] who reduce energy parity games to energy games). We first determine the strongly connected components (SCCs) of the unweighted automaton. Then we degeneralize each Büchi accepting SCC one by one, using the standard counting construction [20]. Finally, we apply a modified Bellman-Ford algorithm to search for energy feasible lassos that start on the main graph and loop in the SCC traversing the remaining Büchi condition.

*Running example 1.* To clarify notation and put the concepts into context, we introduce a small running example. A satellite in low-earth orbit has a rotation time of about 90 minutes, 40% of which are spent in earth shadow. Measuring time in minutes and (electrical) energy in unspecified "energy units", we may thus model its simplified base electrical system as shown in Fig. 1a.

[1] See `https://github.com/ticktac-project/tchecker`
[2] See `https://spot.lrde.epita.fr/`

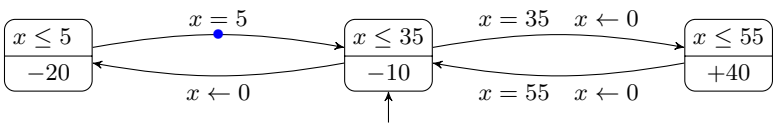



Fig. 2: Weighted timed automaton $A_1$ for satellite with work module.

This is a weighted timed automaton (the formalism will be introduced in Sect. 3) with one clock, $x$, and two locations. The clock is used to model time, which progresses with a constant rate but can be reset on transitions. The initial location on the left (modeling earth shadow) is only active as long as $x \leq 35$, and given that $x$ is initially zero, this means that the model may stay here for at most 35 minutes. Staying in this location consumes 10 energy units per minute, corresponding to the satellite's base consumption.

After 35 minutes the model transitions to the "sun" location on the right, where it can stay for at most 55 minutes and the solar panels produce 50 energy units per minute, from which the base consumption has to be subtracted. Note that the transitions can only be taken if the clock shows exactly 35 (resp. 55) minutes; the clock is reset to zero after the transition, as denoted by $x \leftarrow 0$. This ensures that the satellite stays exactly 35 minutes in the shadow and 55 minutes in the sun, roughly consistent with the "physical" model.

Figure 1b shows a translation of the automaton of Fig. 1a to a weighted untimed automaton. State 1 corresponds to the "shadow" location, transitions are annotated with the corresponding weights, the rate of the location multiplied by the time spent in it. In Sect. 3 we will show how to obtain a weighted automaton from a weighted timed automaton with precisely one clock.

One may now pose the following question: for a given battery capacity $b$ and an initial charge $c$, is it possible for the satellite to function indefinitely without ever running out of energy? It is clear that for $c < 350$ or $b < 350$, the answer is no: the satellite will run out of battery before ever leaving Earth's shadow; for $b \geq 350$ and $c \geq 350$, it will indeed never run out of energy.

Now assume that the satellite also has some work to do: once in a while it must, for example, send some collected data to earth. Given that we can only handle weighted automata with precisely one clock (see Sect. 3), we model the combined system as in Fig. 2. That is, work (modeled by the leftmost location) takes 5 minutes and costs an extra 10 energy units per minute. The dot on the outgoing transition of the work state marks a (transition-based) Büchi condition which forces us to see the transition infinitely often in order for the run to be accepted. As a consequence, all accepting runs also visit the "work" state indefinitely often, consistent with the demand to send data once in a while. In order to model the system within the constraints of our modeling formalism, we must make two simplifying assumptions, both unrealistic but conservative:

- work occurs during earth shadow;
- work prolongs earth shadow time.

The reason for the second property is that the clock $x$ is reset to 0 when entering the work state; otherwise we would not be able to model that it lasts 5 minutes without introducing a second clock. It is clear how further work modules may be added in a similar way, each with their own accepting color.

We will come back to this example later and, in particular, argue that the above assumptions are indeed conservative in the sense that any behavior admitted in our model is also present in a more realistic model which we will introduce.

## 2 Energy Büchi Problems in Finite Weighted Automata

We now define energy Büchi problems in finite weighted automata and show how they may be solved. The similar setting for weighted *timed* automata will be introduced in Sect. 3.

**Definition 1 (WBA).** *A* weighted *(transition-based, generalized)* Büchi automaton *(WBA) is a structure* $A = (\mathcal{M}, S, s_0, T)$ *consisting of a finite set of colors* $\mathcal{M}$*, a set of states* $S$ *with initial state* $s_0 \in S$*, and a set of transitions* $T \subseteq S \times 2^{\mathcal{M}} \times \mathbb{R} \times S$*.*

A transition $t = (s, M, w, s') \in T$ in a WBA is thus annotated by a set of colors $M$ and a real weight $w$, denoted by $s \xrightarrow{w}_M s'$; to save ink, we may omit any or all of $w$ and $M$ from transitions and $\mathcal{M}$ from WBAs. The automaton $A$ is *finite* if $S$ and $T \subseteq S \times 2^{\mathcal{M}} \times \mathbb{Z} \times S$ are finite (thus finite implies integer-weighted).

A *run* in a WBA is a finite or infinite sequence $\rho = s_1 \to s_2 \to \cdots$. We write $\mathsf{first}(\rho) = s_1$ for its starting state and, if $\rho$ is finite, $\mathsf{last}(\rho)$ for its final state. *Concatenation* $\rho_1\rho_2$ of runs is the usual partial operation defined if $\rho_1$ is finite and $\mathsf{last}(\rho_1) = \mathsf{first}(\rho_2)$. Also *iteration* $\rho^n$ of finite runs is defined as usual, for $\mathsf{first}(\rho) = \mathsf{last}(\rho)$, and $\rho^\omega = \operatorname{inj}\lim_{n\to\infty} \rho^n$ denotes infinite iteration.

For $c, b \in \mathbb{N}$ [3] and a run $\rho = s_1 \xrightarrow{w_1} s_2 \xrightarrow{w_2} \cdots$, the $(c, b)$*-accumulated weights* of $\rho$ are the elements of the finite or infinite sequence $\mathsf{weights}_{c\downarrow b}(\rho) = (e_1, e_2, \dots)$ defined by $e_1 = \min(b, c)$ and $e_{i+1} = \min(b, e_i + w_i)$. Hence the transition weights are accumulated, starting with $c$, but only up to the maximum bound $b$; increases above $b$ are discarded. We call $c$ the *initial credit* and $b$ the *weak upper bound*.

*Running example 2.* In Fig. 1b, and choosing $c = 360$ and $b = 750$, we have a single infinite run $\rho = 1 \xrightarrow{-350} 2 \xrightarrow{2200} 1 \xrightarrow{-350} 2 \xrightarrow{2200} 1 \xrightarrow{-350} \cdots$, with $\mathsf{weights}_{c\downarrow b}(\rho) = (360, 10, 750, 400, 750, \dots)$.

A run $\rho$ as above is said to be $(c, b)$*-feasible* if $\mathsf{weights}_{c\downarrow b}(\rho)_i \geq 0$ for all indices $i$, that is, the accumulated weights of all prefixes are non-negative. (This is the case for the example run above.)

An infinite run $\rho = s_1 \to_{M_1} s_2 \to_{M_2} \cdots$ is *Büchi accepted* if all colors in $\mathcal{M}$ are seen infinitely often along $\rho$, that is, for all $m \in \mathcal{M}$ and any index $i \in \mathbb{N}$, there exists $j > i$ such that $m \in M_j$.

We fix a weak upper bound $b \in \mathbb{N}$ for the rest of the paper and write $c$-feasible instead of $(c, b)$-feasible.

[3] Natural numbers include 0.

**Definition 2.** *The* energy Büchi problem *for a finite WBA $A$ and initial credit $c \in \mathbb{N}$ is to ask whether there exists a Büchi accepted $c$-feasible run in $A$.*

Energy problems for finite weighted automata without Büchi conditions, asking for the existence of any $c$-feasible run, have been introduced in [7] and extended to multiple weight dimensions in [16] where they are related to vector addition systems and Petri nets. We extend them to (transition-based generalized) Büchi conditions here but do not consider an extension to multiple weight dimensions.

**Degeneralization** As a first step to solving energy problems for finite WBAs, we show that the standard counting construction which transforms generalized Büchi automata into simple Büchi automata with only one color, see for example [20], also applies in our weighted setting. To see that, let $A = (\mathcal{M}, S, s_0, T)$ be a (generalized) WBA, write $\mathcal{M} = \{m_1, \dots, m_k\}$, and define another WBA $\bar{A} = (\bar{\mathcal{M}}, \bar{S}, \bar{s}_0, \bar{T})$ as follows:

$$\begin{aligned} \bar{\mathcal{M}} = \{m_a\} \qquad \bar{S} = S \times \{1, \dots, k\} \qquad \bar{s}_0 = (s_0, 1) \\ \bar{T} = \big\{((s,i), \emptyset, w, (s',i)) \;\big|\; (s, M, w, s') \in T, m_i \notin M\big\} \\ \cup \big\{((s,i), \emptyset, w, (s',i+1)) \;\big|\; i \neq k, (s, M, w, s') \in T, m_i \in M\big\} \\ \cup \big\{((s,k), \{m_a\}, w, (s',1)) \;\big|\; (s, M, w, s') \in T, m_k \in M\big\} \end{aligned}$$

That is, we split the states of $A$ into levels $\{1, \dots, k\}$. At level $i$, the same transitions exist as in $A$, except those colored with $m_i$; seeing such a transition puts us into level $i+1$, or 1 if $i = k$. In the latter case, the transition in $\bar{A}$ is colored by its only color $m_a$. Intuitively, this preserves the language as we are sure that all colors of the original automaton $A$ have been seen:

**Lemma 3.** *For any $c \in \mathbb{N}$, $A$ admits a Büchi accepted $c$-feasible run iff $\bar{A}$ does.*

**Reduction to lassos** An infinite run $\rho$ in $A$ is a *lasso* if $\rho = \gamma_1 \gamma_2^\omega$ for finite runs $\gamma_1$ and $\gamma_2$. The following lemma shows that it suffices to search for lassos in order to solve energy Büchi problems.

**Lemma 4.** *For any $c \in \mathbb{N}$, $A$ admits a Büchi accepted $c$-feasible infinite run iff it admits a Büchi accepted $c$-feasible lasso.*

Hence our energy Büchi problem may be solved by searching for Büchi accepted $c$-feasible lassos. We detail how to do this in Sect. 4, here we just sum up the complexity result which we prove at the end of Sect. 4.

**Theorem 5.** *Energy Büchi problems for finite WBA are decidable in polynomial time.*

## 3 Energy Büchi Problems for Weighted Timed Automata

We now extend our setting to weighted timed automata. Let $X$ be a finite set of clocks. We denote by $\Phi(X)$ the set of *clock constraints* $\varphi$ on $X$, defined by the grammar $\varphi ::= x \bowtie k \mid \varphi_1 \wedge \varphi_2$ with $x \in X$, $k \in \mathbb{N}$, and $\bowtie \in \{\leq, <, \geq, >, =\}$. A *clock valuation* on $X$ is a function $v : X \to \mathbb{R}_{\geq 0}$. The clock valuation $v_0$ is given by $v_0(x) = 0$ for all $x \in X$, and for $v : X \to \mathbb{R}_{\geq 0}$, $d \in \mathbb{R}_{\geq 0}$, and $R : X \to (\mathbb{N} \cup \{\bot\})$, we define the delay $v + d$ and reset $v[R]$ by

$$(v+d)(x) = v(x) + d, \qquad v[R](x) = \begin{cases} v(x) & \text{if } R(x) = \bot, \\ R(x) & \text{otherwise.} \end{cases}$$

Note that in $v[R]$ we allow clocks to be reset to arbitrary non-negative integers instead of only 0 which is assumed in most of the literature. It is known [24] that this does not change expressivity, but it adds notational convenience. A clock valuation $v$ *satisfies* clock constraint $\varphi$, denoted $v \models \varphi$, if $\varphi$ evaluates to true with $x$ replaced by $v(x)$ for all $x \in X$.

**Definition 6 (WTBA).** *A* weighted timed *(transition-based, generalized)* Büchi automaton *(WTBA) is a structure* $A = (\mathcal{M}, Q, q_0, X, I, E, r)$ *consisting of a finite set of colors* $\mathcal{M}$*, a finite set of locations* $Q$ *with initial location* $q_0 \in Q$*, a finite set of clocks* $X$*, location invariants* $I : Q \to \Phi(X)$*, a finite set of edges* $E \subseteq Q \times 2^{\mathcal{M}} \times \Phi(X) \times (\mathbb{N} \cup \{\bot\})^X \times Q$*, and location weight-rates* $r : Q \to \mathbb{Z}$*.*

As before, we may omit $\mathcal{M}$ from the signature and colors from edges if they are not necessary in the context. Note that the edges carry no weights here, which would correspond to discrete weight updates. In a WTBA, only locations are weighted by a rate. Even without Büchi conditions, the approach laid out here would not work for weighted edges. This was already noted in [7]; instead it requires different methods which are developed in [6] (see also [14, 15]). There, one-clock weighted timed automata (with edge weights) are translated to finite automata weighted with so-called *energy functions* instead of integers. We believe that our extension to Büchi conditions should also work in this extended setting, but leave the details to future work.

The *semantics* of a WTBA $A$ as above is the (infinite) WBA $[\![A]\!] = (\mathcal{M}, S, s_0, T)$ given by $S = \{(q, v) \in Q \times \mathbb{R}^X_{\geq 0} \mid v \models I(q)\}$ and $s_0 = (q_0, v_0)$. Transitions in $T$ are of the following two types:

- *delays* $(q, v) \xrightarrow{w}_{\emptyset}^{d} (q, v+d)$ for all $(q, v) \in S$ and $d \in \mathbb{R}_{\geq 0}$ for which $v + d' \models I(q)$ for all $d' \in [0, d]$, with $w = r(q)d$; [4]
- *switches* $(q, v) \xrightarrow{0}_{M}^{0} (q', v')$ for all $e = (q, M, g, R, q') \in E$ for which $v \models g$, $v' = v[R]$ and $v' \models I(q')$.

[4] Here we annotate transitions with the time $d$ which passes; we only need this to exclude Zeno runs below and will otherwise omit the annotation.

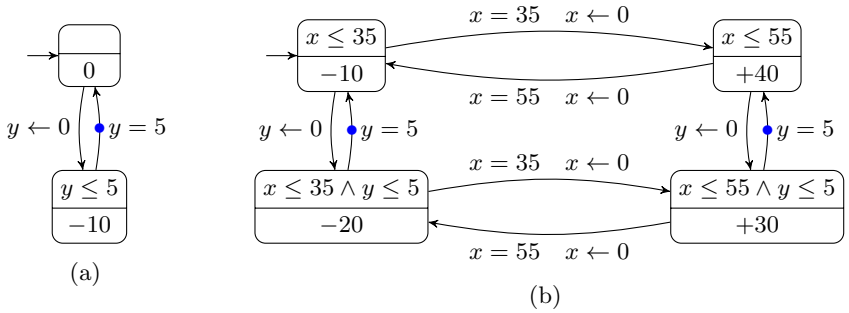



Fig. 3: Satellite example. (a) work module $W$; (b) product $B_1 = A \parallel W$

Each state in $[\![A]\!]$ corresponds to a tuple containing a location in $A$ and a clock valuation $X \to \mathbb{R}_{\geq 0}$. This allows to keep track of the discrete state as well as the evolution of the clocks. By abuse of notation, we will sometimes write $(q, v) \in [\![A]\!]$ instead of $(q, v) \in S$, for $S$ as defined above.

We may now pose energy Büchi problems also for WTBAs, but we wish to exclude infinite runs in which time is bounded, so-called Zeno runs. Formally an infinite run $(q_0, v_0) \to^{d_1} (q_1, v_1) \to^{d_2} \cdots$ is *Zeno* if $\sum d_i$ is finite: Zeno runs admit infinitely many steps in finite time and are hence considered unrealistic from a modeling point of view [1, 21].

**Definition 7.** *The* energy Büchi problem *for a WTBA $A$ and initial credit $c \in \mathbb{N}$ is to ask if there exists a Büchi accepted $c$-feasible non-Zeno run in $[\![A]\!]$.*

We continue our running example; but to do so properly, we need to introduce products of WTBAs. Let $A_i = (\mathcal{M}_i, Q_i, q_0^i, X_i, I_i, E_i, r_i)$, for $i \in \{1, 2\}$, be WTBAs. Their *product* is the WTBA $A_1 \parallel A_2 = (\mathcal{M}, Q, q_0, X, I, E, r)$ with

$$\begin{aligned} \mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2, \qquad Q = Q_1 \times Q_2, \qquad q_0 = (q_0^1, q_0^2), \qquad X = X_1 \cup X_2, \\ I((q_1, q_2)) = I(q_1) \wedge I(q_2), \qquad r((q_1, q_2)) = r(q_1) + r(q_2), \\ E = \big\{((q_1, q_2), M, g, R, (q_1', q_2)) \;\big|\; (q_1, M, g, R, q_1') \in E_1\big\} \\ \cup \big\{((q_1, q_2), M, g, R, (q_1, q_2')) \;\big|\; (q_2, M, g, R, q_2') \in E_2\big\}. \end{aligned}$$

*Running example 3.* Let $A$ be the basic WTBA of Fig. 1a and $A_1$ the combination of $A$ with the work module of Fig. 2. Now, instead of building $A_1$ as we have done, a principled way of constructing a model for the satellite-with-work-module would be to first model the work module $W$ and then form the product $A \parallel W$. We show such a work module and the resulting product $B_1$ in Fig. 3.

As expected, $W$ expresses that work takes 5 minutes and costs 10 energy units per minute, and the Büchi condition enforces that work is executed infinitely often. The product $B_1$ models the shadow-sun cycle together with the fact that work may be executed at any time, and contrary to our "unrealistic" model $A_1$ of Fig. 2, work does not prolong earth shadow time.

Now $B_1$ has *two* clocks, and we will see below that our constructions can handle only one. This is the reason for our "unrealistic" model $A_1$, and we can now state precisely in which sense it is conservative: if $[\![A_1]\!]$ admits a Büchi accepted $c$-feasible non-Zeno run, then so does $[\![B_1]\!]$. For a proof of this fact, one notes that any infinite run $\rho$ in $[\![A_1]\!]$ may be translated to an infinite run $\bar{\rho}$ in $[\![B_1]\!]$ by adjusting the clock valuation by 5 whenever the work module is visited.

**Bounding Clocks** As a first step to solve energy Büchi problems for WTBAs, we show that we may assume that the clocks in any WTBA $A$ are bounded above by some $N \in \mathbb{N}$, *i.e.,* such that $v(x) \leq N$ for all $(q, v) \in [\![A]\!]$ and $x \in X$. This is shown for reachability in [3]; the following lemma extends it to Büchi acceptance.

**Lemma 8.** *Let $A = (\mathcal{M}, Q, q_0, X, I, E, r)$ be a WTBA and $c \in \mathbb{N}$. Let $N$ the maximum constant appearing in any invariant $I(q)$, for $q \in Q$, or in any guard $g$, for $(q, M, g, R, q') \in E$. There is a WTBA $\bar{A} = (\mathcal{M}, Q, q_0, X, \bar{I}, \bar{E}, r)$ such that*

1. *$v(x) \leq N + 2$ for all $x \in X$ and $(q, v) \in [\![\bar{A}]\!]$, and*
2. *there exists a $c$-feasible Büchi accepted run in $[\![A]\!]$ iff such exists in $[\![\bar{A}]\!]$.*

**Corner-point abstraction** We now restrict to WTBAs with only *one* clock and show how to translate these into finite untimed WBAs using the corner-point abstraction. This abstraction may be defined for any number of clocks, but it is shown in [8] that the energy problem is undecidable for weighted timed automata with four clocks or more; for two or three clocks the problem is open.

Let $A = (\mathcal{M}, Q, q_0, X, I, E, r)$ be a WTBA with $X = \{x\}$ a singleton. Using Lemma 8 we may assume that $x$ is bounded by some $N \in \mathbb{N}$, *i.e.,* such that $v(x) \leq N$ for all $(q, v) \in [\![A]\!]$.

Let $\mathfrak{C}$ be the set of all constants which occur in invariants $I(q)$ or guards $g$ or resets $R$ of edges $(q, M, g, R, q')$ in $A$, and write $\mathfrak{C} \cup \{N\} = \{a_1, \dots, a_{n+1}\}$ with ordering $0 \leq a_1 < \dots < a_{n+1}$. The *corner-point regions* [3, 23] of $A$ are the subsets $\{a_i\}$, for $i = 1, \dots, n+1$, $[a_i, a_{i+1}[$, and $]a_i, a_{i+1}]$, for $i = 1, \dots, n$, of $\mathbb{R}_{\geq 0}$; that is, points, left-open, and right-open intervals on $\{a_1, \dots, a_{n+1}\}$.

These are equivalent to clock constraints $x = a_i$, $a_i \leq x < a_{i+1}$, and $a_i < x \leq a_{i+1}$, respectively, defining a notion of implication $\mathfrak{r} \Rightarrow \varphi$ for $\mathfrak{r}$ a corner-point region and $\varphi \in \Phi(\{x\})$.

The corner-point abstraction of $A$ is the finite WBA $\mathsf{cpa}(A) = (\mathcal{M} \cup \{m_z\}, S, s_0, T)$, where $m_z \notin \mathcal{M}$ is a new color, $S = \{(q, \mathfrak{r}) \mid q \in Q, \mathfrak{r}$ corner-point region of $A, \mathfrak{r} \Rightarrow I(q)\}$, $s_0 = (q_0, \{0\})$, and transitions in $T$ are of the following types:

- *delays* $(q, \{a_i\}) \xrightarrow{0}_{\emptyset} (q, [a_i, a_{i+1}[)$, $(q, [a_i, a_{i+1}[) \xrightarrow{w}_{\{m_z\}} (q, ]a_i, a_{i+1}])$ with $w = r(q)(a_{i+1} - a_i)$, and $(q, ]a_i, a_{i+1}]) \xrightarrow{0}_{\emptyset} (q, a_{i+1})$;
- *switches* $(q, \mathfrak{r}) \xrightarrow{0}_M (q', \mathfrak{r})$ for $e = (q, M, g, (x \mapsto \bot), q') \in E$ with $\mathfrak{r} \Rightarrow g$ and $(q, \mathfrak{r}) \xrightarrow{0}_M (q', \{k\})$ for $e = (q, M, g, (x \mapsto k), q') \in E$ with $\mathfrak{r} \Rightarrow g$.

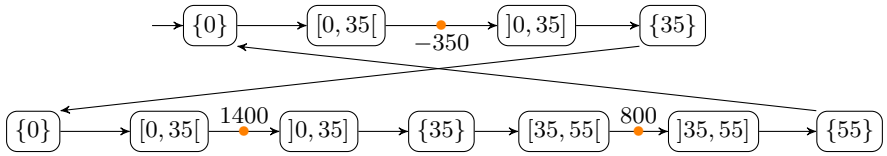



Fig. 4: Corner-point abstraction of base module of Fig. 1a.

The new color $m_z$ is used to rule out Zeno runs, see [1] for a similar construction: any Büchi accepted infinite run in $\mathsf{cpa}(A)$ must have infinitely many time-increasing delay transitions $(q, [a_i, a_{i+1}[) \xrightarrow{w}_{\{m_z\}} (q, ]a_i, a_{i+1}])$.

**Theorem 9.** *Let $A$ be a one-clock WTBA and $c \in \mathbb{N}$.*

1. *If there is a non-Zeno Büchi accepted $c$-feasible run in $[\![A]\!]$, then there is a Büchi accepted $c$-feasible run in $\mathsf{cpa}(A)$.*
2. *If there is a Büchi accepted $c$-feasible run in $\mathsf{cpa}(A)$, then there is a non-Zeno Büchi accepted $(c+\varepsilon)$-feasible run in $[\![A]\!]$ for any $\varepsilon > 0$.*

The so-called *infimum energy condition* [7] in the second part above, replacing $c$ with $c+\varepsilon$, is necessary in the presence of *strict* constraints $x < c$ or $x > c$ in $A$. The proof maps runs in $A$ to runs in $\mathsf{cpa}(A)$ by pushing delays to endpoints of corner-point regions, ignoring strictness of constraints, and this has to be repaired by introducing the infimum condition.

*Running example 4.* We construct the corner-point abstraction of the base module $A$ of Fig. 1a. Its constants are $\{0, 35, 55\}$, yielding the following corner point regions:

$$\{0\}, \quad [0, 35[, \quad ]0, 35], \quad \{35\}, \quad [35, 55[, \quad ]35, 55], \quad \{55\}$$

The corner-point abstraction of $A$ now looks as in Fig. 4, with the states corresponding to the "shadow" location in the top row; the colored transitions correspond to the ones in which time elapses. Note that this WBA is equivalent to the one in Fig. 1b.

Using the corner-point abstraction, we may now solve energy Büchi problems for one-clock WTBAs by translating them into finite WBAs and applying the algorithms of Sect. 2 and the forthcoming Sect. 4.

## 4 Implementation

We now describe our algorithm to solve energy Büchi problems for finite WBA; all of this has been implemented and is available at `https://github.com/PhilippSchlehuberCaissier/wspot`.

We have seen in Sect. 2 that this problem is equivalent to the search for Büchi accepted $c$-feasible lassos. By definition, a lasso $\rho = \gamma_1 \gamma_2^\omega$ consists of two parts,

**Algorithm 1** Algorithm to find Büchi accepted lassos in WBA

**Input:** weak upper bound $b$

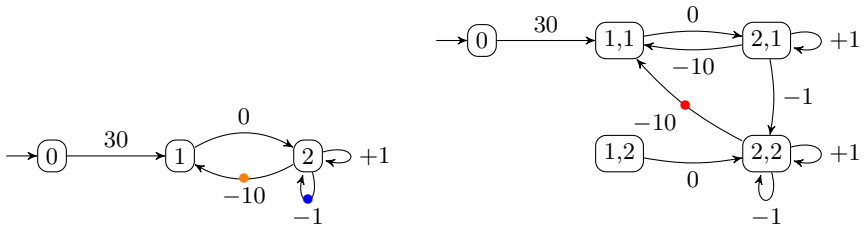
```
1: function BüchiEnergy(graph G, initial credit c)
2:     E ← FindMaxE(G, G.initial_state, c) // E : S → ℕ, mapping states to energy
3:     SCCs ← Couvreur(G)                                          // Find all SCCs
4:     for all scc ∈ SCCs do
5:         GS, backedges ← degeneralize(scc)
6:         for all t = src --w--> dst ∈ backedges do
7:             E' ← FindMaxE(GS, dst, E[dst])                  // t.dst is in G and GS...
8:             e' ← min(b, E'[src] + w)                        // ...(see Fig. 5b)
9:             if E[dst] ≤ e' then
10:                return ReportLoop()
11:            else                                            // Second iteration (see Fig. 5a)
12:                E'' ← FindMaxE(GS, dst, e')
13:                if e' ≤ min(b, E''[src] + w) then
14:                    return ReportLoop()
15:    return ReportNoLoop()
```

the lasso prefix $\gamma_1$ (possibly empty, only traversed once) and the lasso cycle $\gamma_2$ (repeated indefinitely). In order for $\rho$ to be Büchi accepted and $c$-feasible, both the prefix $\gamma_1$ and the cycle $\gamma_2$ must be $c$-feasible, however only the cycle needs to be Büchi accepted.

**Finding lassos** The overall procedure to find lassos is described in Alg. 1. It is based on two steps. In step one we compute all energy-optimal paths starting at the initial state of the automaton with initial credit $c$. This step is done on the original WBA, and we do not take into account the colors. Optimal paths found in this step will serve as lasso prefixes.

The second step is done individually for each Büchi accepting SCC. The Couvreur algorithm ignores the weights, and we can use the version distributed by Spot. We then degeneralize the accepting SCCs one by one, as described in Sect. 2; recall that this creates one copy of the SCC, which we call a level, per color. The first level roots the degeneralization in the original automaton; transitions leading back from the last to the first level are called back-edges. These back-edges play a crucial role as they are the only colored transitions in the degeneralized SCC and represent the accepting transitions.

Hence any Büchi accepting cycle in the degeneralization needs to contain at least one such back-edge, and we can therefore focus our attention on these. We proceed to check for each back-edge whether we can embed it in a $c$-feasible cycle within the degeneralized SCC. To this end, we compute the energy-optimal paths starting at the destination of the current back-edge (by construction a state in the first level) with an initial credit corresponding to its maximal prefix energy (as found in the first step). By comparing the energy of the source state of the back-edge $e$ while taking into account its weight, one can determine whether there exists a $c$-feasible cycle containing $e$. If this is the case, then we have found




(a) Original WBA

(b) Degeneralizing SCC $\{1, 2\}$ with level 1 rooted in the original WBA. Back-edges colored red.

Fig. 5: Left: WBA (also used in Example 10); right: degeneralization of one SCC (states named $originalstate$, $level$).

a $c$-feasible lasso cycle, and by concatenating it with the prefix found in the first step, we can construct a lasso. Note that we might have to check the loop a second time (using the energy level calculated in the first iteration as initial credit), see Example 10. If the answer is negative, we continue with the next back-edge in the SCC or with the next SCC once all back-edges exhausted.

*Example 10.* Figure 5a shows an automaton where we have to compute maximal energy levels twice (lines 11-14 in Alg. 1): with $b = 30$ and $c = 0$, the prefix energy of state 1 is 30, while its optimal energy on the cycle is 20, despite it being part of a energy-positive loop. Hence we cannot conclude that we have found an accepting lasso after the first iteration, but need to run the algorithm once more with an initial credit of 20.

**Finding energy optimal paths** We now discuss how to find energy optimal paths. The problem is equivalent (but inverse) to finding shortest paths in weighted graphs. This may be done using the well-known Bellman-Ford algorithm [4, 18], which breaks with an error if it finds negative loops. In our inverted problem, we are seeking to maximize energy, so positive loops are accepted and even desired. To take into account this particularity, we modify the Bellman-Ford algorithm to invert the weight handling and to be able to handle positive loops. The modified Bellman-Ford algorithm is given in Alg. 2.

The standard algorithm computes shortest paths by relaxing the distance approximation until the solution is found. One round relaxes all edges and the algorithm makes as many rounds as there are nodes. Inverting the algorithm is easy: the relaxation is done if the new weight is higher than the old weight; additionally the new weight has to be higher than 0 and is bounded from above by the weak upper bound.

The second modification to Bellman-Ford is the handling of positive loops. This part is a bit more involved, especially if one strives for an efficient algorithm. We could run Bellman-Ford until it reaches a fixed point, however this can significantly impact performance as shown in the following example.

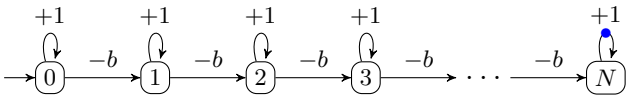



Fig. 6: WBA for Example 11

*Example 11.* Consider the automaton shown in Fig. 6. Here one round of Bellman-Ford only increases the energy level by 1 at the rightmost state already reached and possibly reaches the state to its right once the weak upper bound attained. This means that we need to run $(N+1) \cdot b$ rounds of Bellman-Ford to reach a fixed point. Ideally we would like the upper bound to have no influence on the runtime. To this end we introduce the function PumpAll, which sets the energy level of all states on positive loops detected by the last round of Bellman-Ford to the achievable maximum. This way, instead of needing $b$ rounds of Bellman-Ford to attain the maximal energy, we only need one plus a call to PumpAll.

Before continuing, we make the following observation. This stage will be called from Algorithm 1 that recognizes loops necessary to fulfill the Büchi condition. Here, we only need to check reachability. Therefore, the only reason to form a loop is to gain energy, implying that we are only interested in *simple* energy positive loops, *i.e.,* loops where every state appears at most once. If we set the optimal reachable weight in simple loops, then nested loops are updated by Bellman-Ford in the usual way afterwards.

To improve the runtime of our algorithm, we exploit that Bellman-Ford can detect positive cycles and handle these cycles specifically. Note however that contrary to a statement in [7], we cannot simply set all energy levels on a positive loop to $b$: in the example of Fig. 5a, starting in state 2 with an initial credit of 10, the energy level in state 1 will increase with every round of Bellman-Ford but never above $20 = b - 10$.

In order to have an algorithm whose complexity is independent of $b$, we instead compute the fixed point from above. We first make the following observation.

**Lemma 12.** *In energy positive loops, there exists at least one state on the loop that can attain the maximal energy $b$.*

*Proof.* Since the loop is energy positive we can increase the energy level at any specific node by cycling through the loop. This can be repeated until a fixed point is reached. This fixed point is only reached when at one of the states the accumulated weight reaches $b$ (or surpasses $b$ but is restricted to $b$). As the increase of energy with every cycle is a strictly monotone operation, the fixed point will be reached and no alternation is possible. □

If we knew the precise state that attains maximal energy, we could set its energy to $b$ and loop through the cycle once while propagating the energy, causing every state on the loop to obtain its maximal energy. However, not knowing which state will effectively attain $b$, we start with any state on the loop, set

**Algorithm 2** Modified Bellman-Ford

**Shared Variables:** $E, P$

Modified Bellman-Ford algorithm

**function** MODBF(weighted graph $G$)
  **for** $n \in \{1, \ldots, |S|\}$ **do**
    **for all** $t = s \xrightarrow{w} s' \in T$ **do**
      $e' \leftarrow \min(E(s) + w, b)$
      **if** $E[s'] < e'$ and $e' \geq 0$ **then**
        $E[s'] \leftarrow e'$
        $P[s'] \leftarrow t$ // *$P\colon S \to T$, mapping states to best incoming transition*

Helper function assigning the optimal energy to all states on the energy positive loop containing state $s$

**function** PUMPLOOP(weighted graph $G$, state $s$)
  **for all** $s' \in$ LOOP(s) **do** // *LOOP returns the states on the loop of $s$ ...*
    $E[s'] \leftarrow -1$ // *Special value to detect fixed point*
  $E[P[s].src] \leftarrow b$
  **while** $\top$ **do** // *Loops at most twice*
    **for all** $s' \in$ LOOP(s) **do** // *... in forward order*
      $t \leftarrow P[s']$
      $e' \leftarrow \min(b, E[t.src] + t.w)$
      **if** $e' = E[t.dst]$ **then**
        Mark loop (and postfix) as done
        **return** // *fixed point reached*
      $E[t.dst] \leftarrow e'$

Helper function, pumping all energy positive loops induced by $P$

**function** PUMPALL(weighted graph $G$)
  **for all** states $s$ that changed their weight **do**
    $t = P[s]$
    **if** $\min(b, E[t.src] + t.w) > E[s]$ **then**
      $s' \leftarrow s$ // *$s$ can be either on the cycle or in a postfix of one*
      **repeat** // *Go through it backwards to find a state on the cycle*
        $s'.mark \leftarrow \top$
        $s' \leftarrow t.src$
      **until** $s'$ already marked
      PUMPLOOP($G, s'$) // *Pump it*

Function computing the optimal energy for each state

**function** FINDMAXE(graph $G$, start state $s_0$, initial credit $c$)
  Init($s_0$, $c$) // *initialize values in $E$ to $-\infty$ and $E(s_0) = c$*
  **while** *not $fixedpoint(E)$* **do** // *Iteratively search for loops, then pump them*
    MODBF($G$)
    PUMPALL($G$)
  **return** copyOf($E$)

its energy to $b$ and propagate the energy along the loop until a fixed point is reached. This is the case after traversing the loop at most twice. This is done by the function PumpLoop.

**Lemma 13.** PumpLoop *calculates the desired fixed point after at most two cycles through the loop.*

*Proof.* In Alg. 2, lines 9 and 10 ensure that the fixed point check in line 16 does not detect false positives. After setting an arbitrary state's energy to $b$, the algorithm cycles through the states in the loop in forward order.

Consider w.l.o.g. the positive cycle $\gamma = s_1 \xrightarrow{w_1} s_2 \xrightarrow{w_2} \cdots \xrightarrow{w_{N-1}} s_N$ with $s_1 = s_N$. By Lemma 12 we know that there exists at least one state $s_j$ with $0 \leq j < N$ whose maximal energy equals $b$. Before the first energy propagating traversal of the cycle we set the energy of $s_1$ to $b$. Two cases present themselves. If $j = 0$, then energy is correctly propagated and we reach a fixed point after one traversal. In the second case, the energy attainable by $s_1$ is strictly smaller than $b$. Propagating from this energy level will over-approximate the energies reached by the states $s_0$ through $s_{j-1}$ on the cycle, but only until state $s_j$ is reached which actually attains $b$. As energy is bounded, the energy levels of state $s_j$ and its successors $s_{j+1}, \ldots, s_N$ are correctly calculated. This means that after traversing the cycle $s_j \xrightarrow{w_j} \cdots \xrightarrow{w_{N-1}} s_N \xrightarrow{w_1} s_2 \xrightarrow{w_2} \cdots \xrightarrow{w_{j-1}} s_j$, all energy levels on the cycle are correctly calculated and this is guaranteed to happen before traversing the original cycle twice.

The corresponding fixed point condition is detected by line 16 which will stop the iteration. Note that we actually need to check for *changes* in the energy level on line 16, and not whether some state attained energy $b$, as we at this point cannot know whether this energy was reached due to over-approximation. □

Note that the pseudocode shown here is a simplification, as our implementation contains some further optimizations. Namely, we implement an early exit in modBF if we detect that a fixed point is reached, and we keep track of states which have seen an update to their energy, as this allows to perform certain operations selectively.

**Algorithm Complexity** We are now able to conclude our discussion from Sect. 2 and show that energy Büchi problems for finite WBA are decidable in polynomial time.

*Proof (of Theorem 5).* For our decision procedure, the search for strongly connected components can be done in polynomial time. Our modified Bellman-Ford algorithm also has polynomial complexity. It is called once at the beginning and once for every back-edge of every strongly connected component. Given that the number of such back-edges is bounded by the number of edges, we conclude that our overall algorithm has polynomial complexity. □

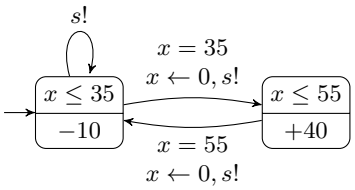



Fig. 7: Base circuit

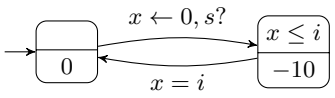



Fig. 8: Work module $\#i$

| #mod | #states | to cpa [s] | sol [s] |
|---|---|---|---|
| 1 | 25 | 0.01 | 0.00 |
| 3 | 90 | 0.03 | 0.02 |
| 5 | 293 | 0.06 | 0.24 |
| 7 | 1012 | 0.19 | 3.24 |
| 9 | 3759 | 0.89 | 59.52 |
| 10 | 7377 | 1.87 | 261.38 |
| 11 | 14582 | 4.37 | 1194.81 |

Table 1: Benchmark results. From left to right: Number of work modules, Number of states in cpa, time needed to compute cpa, time needed to solve energy Büchi problem. Benchmarks done on an ASUS G14, Ryzen 4800H CPU with 16Gb RAM.

## 5 Benchmarks

We employ our running example to build a scalable benchmark case. For modeling convenience we use products of WTBAs as introduced above extended with standard sender/receiver synchronization via channels. The additional labels $s!$ and $s?$ are used for synchronization. Edges with $s!$ can always be taken and emit the signal $s$; edges with $s?$ can only be taken if a signal $s$ is currently emitted. This modeling allows multiple work modules to start working at the same time.

As before, we use a base circuit with two states, see Fig. 7. Work module $\#i$, see Fig. 8, uses 10 energy units while working and spends exactly $i$ time units in the work state. We then combine these models with the specification that time must pass and that every work module is activated infinitely often. All the presented instances are schedulable. Table 1 presents the results of our benchmark, showing that the presented approach scales fairly well. We note that most of the time for solving the energy Büchi problem (last column) is spent in our Python implementation of our modified Bellman-Ford algorithm. In fact the total runtime is (at least for $\#\text{mod} \geq 5$) directly proportional to the number of times lines 4 to 7 of ModBF in Alg. 2 are executed. Therefore, the implementation could greatly benefit from a direct integration into Spot and using its C++ engine.

## 6 Conclusion

We have shown how to efficiently solve energy Büchi problems, both in finite weighted (transition-based generalized) Büchi automata and in one-clock weighted timed Büchi automata. We have implemented all our algorithms in a tool based on TChecker and Spot. Solving the latter problem is done by using the corner-point abstraction to translate the weighted timed Büchi automaton to a finite weighted

Büchi automaton; the former problem is handled by interleaving a modified version of the Bellman-Ford algorithm with Couvreur's algorithm.

Our tool is able to handle some interesting examples, but the restriction to one-clock weighted timed Büchi automata without weights on edges does impose some constraints on modeling. We believe that trying to lift the one-clock restriction is unrealistic; but weighted edges (without Büchi conditions) have been treated in [6], and we suspect that their approach should also be viable here. (See [10] for a related approach.) In passing we should like to argue that, as shown by our running example, the modeling constraints imposed by only having one clock may be somewhat circumvented by careful modeling.

Also adopting our approach to the unlimited energy problem, without weak upper bound, should not pose any problems. In fact, setting $b = \infty$ will facilitate the algorithm, as maximal energy levels of nodes on positive loops can directly be set to $\infty$ (making PumpLoop obsolete), and also the second iteration in Alg. 1 can be dropped.

Further, we strongly believe that our idea of investigating whether a back-edge can be embedded in an energy positive cycle is not restricted to (generalized) Büchi acceptance. In fact, the same methods should be applicable to, for example, parity acceptance conditions without losing the polynomial runtime.

As a last remark, it is known that multiple clocks, multiple weight dimensions, and even turning the weak upper bound into a strict one which may not be exceeded, rapidly leads to undecidability results, see [7, 8, 16, 26], and we are wondering whether some of these may be sharpened when using Büchi conditions.

**Acknowledgments** We are grateful to Alexandre Duret-Lutz, Nicolas Markey and Ocan Sankur for fruitful discussions on the subjects of this paper.

## References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Pierre-Alain Reynier. Optimal and robust controller synthesis using energy timed automata with uncertainty. *Formal Aspects Comput.*, 33(1):3–25, 2021.
3. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *HSCC*, volume 2034 of *Lect. Notes Comput. Sci.*, pages 147–161. Springer, 2001.
4. Richard Bellman. On a routing problem. *Quart. Appl. Math.*, 16(1):87–90, 1958.
5. Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krčál, Gilles Nies, and Marvin Stenger. Battery-aware scheduling in low orbit: The GomX-3 case. In John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou, editors, *FM*, volume 9995 of *Lect. Notes Comput. Sci.*, pages 559–576. Springer, 2016.

6. Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 61–70. ACM, 2010.
7. Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lect. Notes Comput. Sci.*, pages 33–47. Springer, 2008.
8. Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Lower-bound-constrained runs in weighted timed automata. *Perform. Eval.*, 73:91–109, 2014.
9. J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966.
10. David Cachera, Uli Fahrenberg, and Axel Legay. An $\omega$-algebra for real-time energy problems. *Logical Meth. Comput. Sci.*, 15(2), 2019.
11. Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012.
12. Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 505–516, 2010.
13. Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and *omega*-automata manipulation. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *ATVA*, volume 9938 of *Lect. Notes Comput. Sci.*, pages 122–129. Springer, 2016.
14. Zoltán Ésik, Uli Fahrenberg, Axel Legay, and Karin Quaas. An algebraic approach to energy problems I: $^*$-Continuous Kleene $\omega$-algebras. *Acta Cyb.*, 23(1):203–228, 2017.
15. Zoltán Ésik, Uli Fahrenberg, Axel Legay, and Karin Quaas. An algebraic approach to energy problems II: The algebra of energy functions. *Acta Cyb.*, 23(1):229–268, 2017.
16. Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jiří Srba. Energy games in multiweighted automata. In Antonio Cerone and Pekka Pihlajasaari, editors, *ICTAC*, volume 6916 of *Lect. Notes Comput. Sci.*, pages 95–115. Springer, 2011.
17. Heiko Falk, Kevin Hammond, Kim G. Larsen, Björn Lisper, and Stefan M. Petters. Code-level timing analysis of embedded software. In Ahmed Jerraya, Luca P. Carloni, Florence Maraninchi, and John Regehr, editors, *EMSOFT*, pages 163–164. ACM, 2012.
18. Lester R. Ford. *Network Flow Theory.* RAND Corporation, Santa Monica, CA, 1956.
19. Goran Frehse, Kim G. Larsen, Marius Mikučionis, and Brian Nielsen. Monitoring dynamical signals while testing timed aspects of a system. In Burkhart Wolff and Fatiha Zaïdi, editors, *ICTSS*, volume 7019 of *Lect. Notes Comput. Sci.*, pages 115–130. Springer, 2011.
20. Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lect. Notes Comput. Sci.*, pages 53–65. Springer, 2001.
21. Frédéric Herbreteau and B. Srivathsan. Coarse abstractions make Zeno behaviours difficult to detect. *Log. Methods Comput. Sci.*, 9(1), 2011.
22. Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Inf. Comput.*, 251:67–90, 2016.

23. François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lect. Notes Comput. Sci.*, pages 387–401. Springer, 2004.
24. Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *Softw. Tools Techn. Trans.*, 1(1-2):134–152, 1997.
25. Marius Mikučionis, Kim G. Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using Uppaal: Herschel-Planck case study. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (2)*, volume 6416 of *Lect. Notes Comput. Sci.*, pages 175–190. Springer, 2010.
26. Karin Quaas. On the interval-bound problem for weighted timed automata. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *Lect. Notes Comput. Sci.*, pages 452–464. Springer, 2011.
27. Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015.
28. Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *FOCS*, pages 185–194. IEEE Computer Society, 1983.