Battery Transition Systems [*]

Udi Boker
The Interdisciplinary Center, Herzliya, Israel

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria

Arjun Radhakrishna
IST Austria, Klosterneuburg, Austria

Abstract

The analysis of the energy consumption of software is an important goal for quantitative formal methods. Current methods, using weighted transition systems or energy games, model the energy source as an ideal resource whose status is characterized by one number, namely the amount of remaining energy. Real batteries, however, exhibit behaviors that can deviate substantially from an ideal energy resource. Based on a discretization of a standard continuous battery model, we introduce *battery transition systems*. In this model, a battery is viewed as consisting of two parts – the available-charge tank and the bound-charge tank. Any charge or discharge is applied to the available-charge tank. Over time, the energy from each tank diffuses to the other tank.

Battery transition systems are infinite state systems that, being not well-structured, fall into no decidable class that is known to us. Nonetheless, we are able to prove that the ω-regular model-checking problem is decidable for battery transition systems. We also present a case study on the verification of control programs for energy-constrained semi-autonomous robots.

Categories and Subject Descriptors Theory of computation [*Logic*]: Verification by model checking

General Terms Theory, Verification

Keywords Battery, Transition systems, energy, model checking

1. Introduction

Systems with limited energy resources, such as mobile devices or electric cars, have become ubiquitous in everyday life. In accordance, there is a growing attention to the formal modeling of such systems and the analysis of their behavior. These systems are commonly modeled as weighted transition systems, where the states of the transition system represent the system configurations, the transitions represent the possible operations, and the weights on the transitions correspond to the energy consumed (negative value) or added (positive value) during the operation. In recent literature (for example, [7, 8, 19]), weighted transition systems have been analyzed with respect to various problems, such as finite-automaton emptiness problem (starting from a given initial energy, can a specific configuration be reached while keeping the energy positive in all intermediate steps?), and Büchi emptiness problem (can a specific configuration be visited repeatedly, while keeping the energy positive?).

In all these works, the energy-resource is idealized. In particular, it is assumed that its status can be completely characterized by one number, namely the amount of remaining energy. However, physical systems with energy restrictions often use batteries, which are far from an ideal-energy source. One such non-ideal behavior of a battery behavior is the "recovery effect", where the available energy at certain times is smaller than the sum of energies consumed and charged. Intuitively, the recovery effect is a result of the fact that energy is consumed from the edge of the battery, while the total charge is spread across the entire battery. When the consumption is high, additional time may be required until the charge diffuses from the inside of the battery to its edge, during which period there is no available energy, possibly failing the required operation.

The recovery effect is often noticed in our daily usage of battery-powered systems, for example mobile phones – a phone might shutdown due to an "out of power" condition, but then become live again after an idle period.

We aim to formally model such energy systems with non-ideal resources. We define a "battery transition system" (BTS), where the system is viewed as a weighted transition system, as is standard. However, the semantics of its possible traces is specified differently, to capture non-ideal behaviors. The semantics we specify for BTSs correspond to a discretization of a well-known battery model – the kinetic battery model (KiBaM) [20]. There are various battery models in the literature, admitting various accuracies and complexities, among which the KiBaM model is a good choice for the purpose of properly analyzing systems with the recovery effect [17]. We elaborate, in Section 2, on various battery models, and explain the derivation of BTS semantics from the KiBaM model.

Semantics. The status of the battery in a BTS is a pair (x, y), where x represents the *available charge* (available for immediate usage) and y the *bound charge* (internal charge in a battery that is not immediately available). During each transition, some amount of charge diffuses between x and y. The weight of the transition (say w) affects only x in the current step. The diffusion rate depends on the difference between x and y, and on two constants of the battery: a *width constant* $c \in \mathbb{R}$ with $0 < c < 1$, and a *diffusion constant* $k \in \mathbb{R}$ with $0 < k < c(1 - c)$. Formally, making a transition of weight w from a battery status (x, y), results in the battery status (x', y'), where $x' = x - k \cdot (\frac{x}{c} - \frac{y}{1-c}) + w$ and $y' = y + k \cdot (\frac{x}{c} - \frac{y}{1-c})$. The value $k \cdot (\frac{x}{c} - \frac{y}{1-c})$ represents the amount of charge diffused between x and y. The above transition is legal if the available charge x remains positive after the transition.

The mathematical properties of a BTS are shown to be inherently different from those of a simple-energy transition system

[*] This work was supported in part by the Austrian Science Fund NFN RiSE (Rigorous Systems Engineering), and by the ERC Advanced Grant QUAREM (Quantitative Reactive Modeling).

POPL '14 San Diego, CA, USA.
Copyright © 2014 ACM . . . $15.00

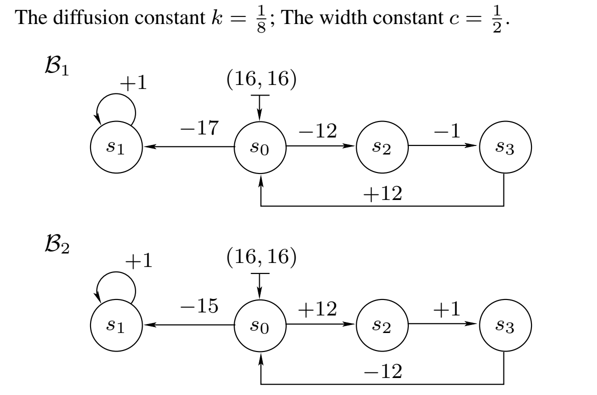

Figure 1. In the battery system $\mathcal{B}_1$, a trace reaching the state s_1 must make a cycle with total negative energy. In $\mathcal{B}_2$, an illegal trace must make different choices at different visits in s_0.

(where only the value of $x+y$ is considered), as illustrated in Fig. 1: Considering the system $\mathcal{B}_1$ as a simple-energy system, where only the total energy should remain positive, s_1 is directly reachable from s_0, and the cycle $s_0 \rightarrow s_2 \rightarrow s_3 \rightarrow s_0$ is completely useless. On the other hand, viewing $\mathcal{B}_1$ as a BTS, in order to go from state s_0 to s_1, counterintuitively, a legal trace must first take the cycle through s_2 and s_3. Though decreasing the total energy, the cycle temporarily increases the available energy, allowing the transition to s_1. Furthermore, it is known that a simple-energy system (even with multi-dimension energies) admits an illegal trace if and only if it admits a memoryless illegal trace (always making the same choice at each state) [8]. However, an illegal trace in the system $\mathcal{B}_2$ of Fig. 1 must make different choices at different visits in state s_0 (Theorem 3).

Model checking. We consider the finite-automaton, Büchi, and Streett emptiness problems for a BTS; these problems are central to the model checking of systems with no fairness constraints, weak fairness constraints, and strong fairness constraints, respectively.

As BTSs are infinite-state systems, it is natural to ask if they fall into a known, tractable, class of infinite state systems. For example, standard model-checking algorithms exist for well-structured transition systems (e.g., lossy channel systems, timed automata [2], etc), where a well-quasi ordering can be defined on the states of the infinite systems, and this ordering is compatible with the transition relation of the system. However, for BTSs, we can show that the infinite sequence $(1, 1), (\frac{3}{4}, 1\frac{1}{4}), (\frac{11}{16}, 1\frac{5}{16}), (\frac{43}{64}, 1\frac{21}{64}), \ldots$ of battery statuses is monotonically decreasing with respect to any ordering that is compatible with transitions. Intuitively, this sequence contains battery statuses that have equal total charge, but strictly decreasing available charge. This implies that model-checking algorithms from the domain of well-structured transitions systems do not apply directly to BTS.

We solve the finite-automaton emptiness problem by building a forward reachability tree, along the lines of the Karp-Miller tree for Petri nets [11]. There, using the well-structured properties of Petri-nets, the Karp-Miller tree is shown to be a finite summarization of all reachable states, despite there being infinitely-many reachable states. A BTS is not well-structured, yet we are able to generate a finite "summary tree", having all the reachability data, by proving the following key observations: 1. Once the total energy in a battery status is high-enough, the problem can be reduced to simple-energy reachability; 2. Considering some characteristic properties of battery statuses allows to define a simulation-compatible total ordering between statuses having the same total energy; and 3. Repeating a cycle whose total energy sums to 0 makes the battery status converge monotonically to a limit value independent of the initial status. Despite the fact that the above ordering is not well-founded, i.e., there may be infinite chains, the last observation lets us take limits of infinite chains while constructing the reachability tree.

In simple-energy systems, extending the finite automaton emptiness algorithm to a Büchi emptiness algorithm is straightforward – checking whether there is a reachable Büchi state that has a cycle back to itself, such that the sum of weights on the cycle is non-negative. In a BTS, such a simple solution does not work – a cycle that does not decrease the total energy might still fail the process after finitely many iterations, as the available charge can slightly decreases on every iteration. A simple modification, seeking cycles that do not decrease both the total and the available energies is too restrictive, as the available charge may still converge to a positive value. We solve the Büchi emptiness problem by showing that if there exists an accepted trace, there also exists a lasso-shaped accepted trace having one of two special forms. These forms concern the way that the available charge changes along the cycle. By a delicate analysis of the reachability tree, we then solve the question of whether the transition system allows for a trace in one of these special forms. The Streett emptiness problem is solved similarly, by using a small extension of the Büchi emptiness algorithm.

We show that our algorithms for the finite-automaton, Büchi, and Streett emptiness problems are in PSPACE with respect to the number of states in the transition system and a unary representation of the weights. If weights are represented in binary, or if the battery constants are arbitrarily small and represented in binary, the space complexity grows exponentially.

Case study: Robot control. We examine a semi-autonomous robot control in an energy-constrained environment. We present a small programming language for robot-controllers and define quantitative battery-based semantics for controllers written in that language. We solve the ω-regular model-checking problem for programs written in this language, using our results on battery transition systems. We demonstrate the inadequacy of standard quantitative verification techniques, where the battery is viewed as an ideal energy resource – they might affirm, for example, that the robot can reach some target locations, while taking into account the non-ideal behavior of its battery, it cannot.

Related work. Batteries are involved devices, exhibiting various different phenomena. Accordingly, there are many different works considering these aspects, for example scheduling the load among several batteries [4, 9, 16, 18], optimizing the lifetime of a battery with respect to the "cycle aging effect" [1], analyzing the thermal effects, and more.

To the best of our knowledge, this is the first work to formally analyze an energy system with a non-ideal energy source. Previous work has either considered ideal energy sources (for example, [8]) or provides approximations for the battery life-time with respect to various discharge scenarios (for example, [16, 17]).

Our finite-automaton emptiness algorithm follows the approach taken in the Karp-Miller tree [11] which can be used in general for well-structured transition systems [12]. However, our systems are not well-structured and a naive application of this technique does not entail termination of algorithms. We use ideas from flattable systems [3] and additional analysis of BTSs to produce a terminating version of these algorithms. In particular, we also use an intricate analysis of BTSs to get an algorithm for deciding Büchi and Streett properties. This kind of analysis is not possible for general flattable systems.

2. Battery Models

We provide a short description of how batteries are modeled in the literature, and explain how we derive our formal model of a battery.

A battery consists of one or more electrochemical cells, each of which contain a negative electrode (anode), a positive electrode (cathode), and a separator between them. During discharge and recharge, electrons move through the external circuit, while chemical reaction produces or consumes chemical energy inside the battery. For example, during discharge in lithium-ion (Li-ion) batteries, positive lithium ions move from the anode to the cathode, while the reverse occurs during recharge (see Fig. 2).

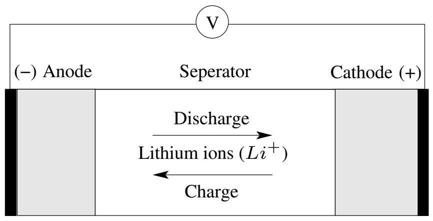

Figure 2. Schematic of a lithium-ion (Li-ion) battery.

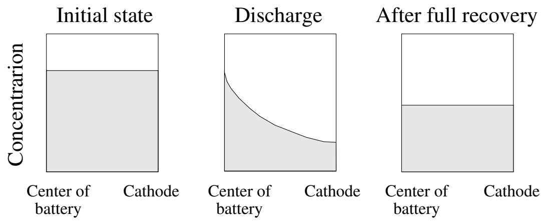

Figure 3. Concentration of electroactive species along the battery, following a discharge and a recovery phase.

Batteries of all types have a "recovery effect", meaning that the chemical reaction inside the battery does not keep up with the rate of the external activity. Internally, in Li-ion batteries, the concentration of the electroactive species near the electrodes becomes smaller than their concentration in the interior of the battery. When the battery has low load for some time, the ions have enough time to diffuse to the electrodes, and charge recovery takes place (see Fig. 3). The well-known symptom of this is that a battery might be "empty" after some usage, but then becomes "charged" after an idle period.

There are many battery models modeling various aspects of a real battery. The most accurate ones model the electrochemical reactions in detail [5, 10, 13, 21]. Though highly accurate, they require configuration of many (usually around 50) parameters, making them difficult to analyze. Another approach taken is to model the electrical properties of the battery using voltage sources, resistors, and other elements [14, 15]. These approximate battery voltage behavior well, but their modeling of the available battery capacity is inaccurate. A third class consists of the analytical models that describe the battery at a high abstraction level, modeling only its major properties by means of a few equations. The dominant models of this class are the kinetic battery model [20], and the diffusion model [22]. A detailed description of the various models can be found in [16, 17].

The possibly simplest, yet useful, model that handles the recovery effect is the kinetic battery model (KiBaM) [20]. While being originally developed for Lead-Acid batteries to model both battery capacity and battery voltage, its capacity modeling was found to be a good approximation even for more modern batteries such as the Li-ion battery. In [16, 17], it was theoretically shown that KiBaM is a first order approximation of the diffusion model, which was designed for Li-ion batteries. In addition, their experimental results show that it has up to 7 percent deviation from the accurate electrochemical models.

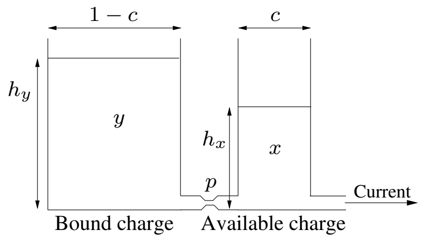

Figure 4. Kinetic battery model (KiBaM).

We concentrate on a battery's available capacity, and hence, adopt the KiBaM model. In this model, the battery charge is distributed over two tanks: the available-charge tank, denoted x, of width $c \in (0,1)$, and the bound-charge tank, denoted y, of width $1-c$ (see Fig. 4). The external current gets electrons only from the available-charge tank, whereas electrons from the bound-charge tank flow to the available-charge tank. When recharging, the reverse process occurs, electrons are added directly to the available-charge tank, from there they flows to the bound-charge tank. The charge flows between the tanks through a "valve" with a fixed conductance p. The parameter p has the dimension 1/time and influences the rate at which the charge can flow between the two tanks. This rate is also proportional to the height difference between the two tanks. If the heights are given by $h_x = x/c$ and $h_y = y/(1-c)$, and the current load by $w(t)$, the charge in the tank over time behaves according to the following system of differential equations [20]:

$$\frac{dx}{dt} = -w(t) - p(h_x - h_y); \qquad \frac{dy}{dt} = p(h_x - h_y) \tag{1}$$

with initial conditions $x(0) = c \cdot C$ and $y(0) = (1-c) \cdot C$, where C is the total battery capacity. The battery cannot supply charge when there is no charge left in the available-charge tank.

We are interested in calculating the battery status along a discrete-time transition system, thus consider the equations (1) for fixed time steps. We get the following equations:

$$x_{i+1} = x_i - w_i - k(h_{x_i} - h_{y_i}); \qquad y_{i+1} = y_i + k(h_{x_i} - h_{y_i}) \tag{2}$$

where x_i and y_i are the values of x and y before the time step i, respectively, w_i is the total load on the battery at time step i, and $k = p \times$ (length of a time step). The smaller the time steps are, the smaller k is, and the more accurate the discretization is.

We further need to ensure that the discretization does not introduce undesirable behaviours. In Eq. 1 if $h_x > h_y$ and $w(t)$ is 0, the relation $h_x > h_y$ keeps holding. We should ensure this in the discrete model, i.e., if $h_{x_i} > h_{y_i}$ and $w_i = 0$, then it cannot be that $h_{x_{i+1}} \leq h_{y_{i+1}}$.

Formalizing the above requirement, we have

$$h_{x_{i+1}} = \frac{x_{i+1}}{c} = \frac{x_i - 0 - k(h_{x_i} - h_{y_i})}{c} = h_{x_i} - \frac{k(h_{x_i} - h_{y_i})}{c};$$

$$h_{y_{i+1}} = \frac{y_{i+1}}{1-c} = \frac{y_i + k(h_{x_i} - h_{y_i})}{1-c} = h_{y_i} + \frac{k(h_{x_i} - h_{y_i})}{1-c}.$$

Hence, $h_{x_{i+1}} - h_{y_{i+1}} = (h_{x_i} - h_{y_i})(1 - k(\frac{1}{c(1-c)}))$.

Therefore, the parameter k is acceptable if $k(\frac{1}{c(1-c)}) < 1$, leading to the conclusion that

$$k < c(1-c) \tag{3}$$

3. Battery Transition Systems

We incorporate the discrete battery model from Equation 2 into a weighted transition system. The system consists of finitely many control-states and weighted transitions between them, where the weights denote the amount of energy recharged/consumed at each operation.

3.1 Weighted Transition Systems and Battery Semantics

A *transition system* is a tuple $\langle S, \Delta, s_\iota \rangle$ where S is a (possibly infinite) set of control states, $s_\iota \in S$ is the initial control state, and $\Delta \subseteq S \times S$ is a set of transitions. A *weighted transition system* (WTS) is a tuple $\mathcal{S} = \langle S, \Delta, s_\iota, \nu \rangle$ where $\langle S, \Delta, s_\iota \rangle$ is a transition system, and $\nu : \Delta \to \mathbb{Z}$ is a weight function labeling transitions with integer weights[1].

A *battery transition system* (BTS or *battery system*, for short) is a tuple $\mathcal{B} = \langle \langle S, \Delta, s_\iota, \nu \rangle, c, k \rangle$ where $\langle S, \Delta, s_\iota, \nu \rangle$ is a WTS with a finite number of control states (i.e., $|S| < \infty$), $c \in \mathbb{R}$ is a *width constant* with $0 < c < 1$, and $k \in \mathbb{R}$ is a *diffusion constant* with $0 < k < c(1-c)$.

Semantics. Given a BTS $\mathcal{B} = \langle \mathcal{S}, c, k \rangle$, a *battery status* $(x, y) \in \mathbb{R}^2_{>0}$ represents the current configuration of the battery. Intuitively, the values x and y represent the charge in the available-charge tank and the bound-charge tank of the battery, respectively (see Figure 4).

If the current battery status is (x, y), on a transition of weight w, we define the change in the battery status using a function $\mathtt{Post} : \mathbb{R}^2 \times \mathbb{Z} \to \mathbb{R}^2$, letting the battery status after the transition $(x', y') = \mathtt{Post}((x, y), w)$. Here, we follow the standard convention of energy transition systems and consider a positive (resp. negative) weight as adding (resp. drawing) a charge to (resp. from) the battery. In matrix notation, we have the following.

$$\mathtt{Post}((x,y),w) = \left[\mathcal{A}_\mathcal{B} \cdot \begin{pmatrix} x \\ y \end{pmatrix}\right]^\top + \begin{pmatrix} w \\ 0 \end{pmatrix}^\top, \text{ where}$$

$$\mathcal{A}_\mathcal{B} = \begin{pmatrix} 1 - \frac{k}{c} & \frac{k}{1-c} \\ \frac{k}{c} & 1 - \frac{k}{1-c} \end{pmatrix}$$

The matrix $\mathcal{A}_\mathcal{B}$ is called the *diffusion matrix*. Note that the $\mathtt{Post}$ function indeed follows Equation 2 except for the change in the sign of w. The values $\frac{k}{c} \cdot x$ and $\frac{k}{1-c} \cdot y$ denote the heights h_x and h_y of the two tanks, and hence, we get that $x' = x - k \cdot (h_x - h_y) + w$ and $y' = y + k \cdot (h_x - h_y)$. We abuse notation by defining $\mathtt{Post}(\mathbf{t}, w_1 w_2 \ldots w_m)$ inductively as $\mathtt{Post}(\mathtt{Post}(\mathbf{t}, w_1), w_2 \ldots w_m)$ where each $w_i \in \mathbb{Z}$.

Given an *initial battery status* $\mathbf{t}_\iota \in \mathbb{R}^2_{>0}$, the *semantics* of a BTS $\mathcal{B} = \langle \langle S, \Delta, s_\iota, \nu \rangle, c, k \rangle$ is given by a (possibly infinite) transition system $\langle E, \to, e_\iota \rangle$ where $E = S \times \mathbb{R}^2_{>0}$ is the set of states, $\to \subseteq E \times E$ is the transition relation, and $e_\iota = (s_\iota, \mathbf{t}_\iota)$ is the initial state.

- We call each $(s, \mathbf{t}) \in E$ an *extended state* with $s \in S$ being its control state, and $\mathbf{t} \in \mathbb{R}^2_{>0}$ being its battery status [2].
- We have $((s, \mathbf{t}), (s', \mathbf{t}')) \in \to$ if and only if $\mathtt{Post}(\mathbf{t}, w) = \mathbf{t}'$ and $(s, s') \in \Delta \wedge \nu((s, s')) = w$. We write $(s, \mathbf{t}) \to (s', \mathbf{t}')$ instead of $((s, \mathbf{t}), (s', \mathbf{t}')) \in \to$.

A weight w (and by extension, a transition with weight w) is *feasible* from battery status $\mathbf{t}$ if $\mathtt{Post}(\mathbf{t}, w) \in \mathbb{R}^2_{>0}$, namely if $\mathtt{Post}(\mathbf{t}, w)$ is a valid battery status. Similarly, a sequence of weights $w_0 w_1 \ldots w_n$ is feasible from $\mathbf{t}$ iff w_0 is feasible from $\mathbf{t}$ and each w_i is feasible from $\mathtt{Post}(\mathbf{t}, w_0 \ldots w_{i-1})$. Extending the nomenclature, we call every $\mathbf{t} \in \mathbb{R}^2 \setminus \mathbb{R}^2_{>0}$ *infeasible*.

The *traces* of a BTS $\mathcal{B}$, denoted $\Pi(\mathcal{B})$, are given by (infinite or finite) paths of the form $\pi = (s_0, \mathbf{t_0})(s_1, \mathbf{t_1}) \ldots$ where $s_0 = s_\iota$ and $\mathbf{t_0} = \mathbf{t}_\iota$ and for every $i \geq 1$, we have $(s_{i-1}, \mathbf{t_{i-1}}) \to (s_i, \mathbf{t_i})$. The corresponding *control trace* is given by $\theta = \mathtt{control}(\pi) = s_0 s_1 \ldots$, and the set of control traces by $\Theta(\mathcal{B}) = \{\mathtt{control}(\pi) \mid \pi \in \Pi(\mathcal{B})\}$. We say that a (finite or infinite) sequence of control states $s_0 s_1 \ldots$ is feasible from a battery status $\mathbf{t}$ iff the weight sequence $w_0 w_1 \ldots$ is feasible from $\mathbf{t}$, where $(s_i, s_{i+1}) \in \Delta \wedge \nu((s_i, s_{i+1})) = w_i$.

Energy feasibility. We define an alternate set of semantics corresponding to the classical notion of ideal-energy systems. We say that $(x, y) \in \mathbb{R}^2$ is *energy-feasible* if $x + y > 0$. As for the term "feasible", we further extend the notion of energy-feasible as follows: 1. A sequence of weights $w_0 \ldots w_n$ is energy-feasible from (x, y) iff $\mathtt{Post}((x, y), w_0 \ldots w_i)$ is energy-feasible for all $0 \leq i \leq n$; and 2. A sequence of control states $s_0 s_1 \ldots$ is energy-feasible from (x, y) iff $w_0 w_1 \ldots$ is energy-feasible from (x, y) where $w_i = \nu(s_i, s_{i+1})$.

Characteristic functions. For mathematical simplicity, we follow the approach taken in [20] and use an alternate representation for battery statuses. We represent (x, y) using two other numbers, denoting it $[\mathtt{e}; \mathtt{d}]$, where $\mathtt{e}$ and $\mathtt{d}$ are defined by the following *energy* and *deviation* functions.

- $\mathtt{e} = \mathtt{energy}((x, y)) = x + y$; and
- $\mathtt{d} = \mathtt{deviation}((x, y)) = x - y \cdot \dfrac{c}{1-c}$.

Intuitively, $\mathtt{e}$ is the total energy in the battery and $\mathtt{d}$ is the difference between the heights of the two tanks multiplied by the factor c. The mathematical simplicity in using *energy* and *deviation* stems from the fact that they correspond to the eigenvectors of the diffusion matrix. Given $\mathtt{e} = \mathtt{energy}(\mathbf{t})$ and $\mathtt{d} = \mathtt{deviation}(\mathbf{t})$, the battery status $\mathbf{t} = (x, y)$ is uniquely determined: $x = c\mathtt{e} + (1 - c)\mathtt{d}$ and $y = \mathtt{e} - x$. Hence, we use the notations $[\mathtt{e}; \mathtt{d}]$ and (x, y) interchangeably.

Proposition 1. *For any battery status* $[\mathtt{e}; \mathtt{d}]$ *and* $w \in \mathbb{N}$, *we have that* $\mathtt{Post}([\mathtt{e}; \mathtt{d}], w) = [\mathtt{e} + w; \lambda \cdot \mathtt{d} + w]$ *where* $\lambda = 1 - \frac{k}{1-c} - \frac{k}{c}$.

The above proposition is a translation of the $\mathtt{Post}$ function to the $[\mathtt{e}; \mathtt{d}]$ notation. Intuitively, the energy is increased by the weight w as expected, while the difference in the tank heights is first reduced by a constant factor of λ and then increased due to the charge w added to the first column. The factor λ turns out to be the central parameter of the battery, playing a key role in how BTSs behaves. The following lemma formalizes the intuition that the bound-charge tank (y) cannot get empty before the available-charge tank (x) does.

Lemma 2. *Suppose battery status* (x, y) *is feasible, and let* $\mathtt{Post}((x, y), w) = (x', y')$. *We have* $[\mathtt{e}'; \mathtt{d}'] = (x', y')$ *is feasible iff* $x' > 0$, *and, equivalently, if and only if* $c\mathtt{e}' + (1 - c)\mathtt{d}' > 0$.

Proof. The *only if* implication is obvious. As for the *if*, we have $x' = (1 - \frac{k}{c})x + \frac{k}{1-c}y + w$ and $y' = \frac{k}{c}x + (1 - \frac{k}{1-c})y$. Assuming $x > 0$ and $y > 0$, it easily follows that $y' > 0$. Hence, (x', y') is infeasible if and only if $x' \leq 0$ or equivalently, if $x' = c\mathtt{e}' + (1 - c)\mathtt{d}' \leq 0$ □

Model checking. The problems we consider ask for the existence of a control trace θ in the semantics of a BTS $\mathcal{B}$ with control states S given an initial battery status $\mathbf{t}$, such that $\theta \in \Phi$ for some given *objective set* $\Phi \subseteq S^* \cup S^\omega$. Specifically, we consider the following objective sets Φ:

- **Finite-automaton emptiness.** Asking if there exists a feasible trace to a set of target control states. Formally, given target control states $T \subseteq S$, we have $\Phi = \mathtt{Reach}(T) = \{s_0 s_1 \ldots \mid$

[1] The weights in a weighted transition system are often rational numbers rather than integers. All of our results equally hold for rational weights, by simply multiplying all weights by the product of all denominators.

[2] All of our results equally hold for the case that the element values should be non-negative, rather than strictly positive, but the proofs are marginally simpler in the strictly positive case.

$\exists i : s_i \in T\}$, i.e., Φ is the set of control traces which visit T at least once.

- **Büchi emptiness.** Asking if there exists a feasible trace which visits a set of target Büchi states infinitely often. Formally, given Büchi control states $L \subseteq S$, we have $\Phi = \texttt{Büchi}(L) = \{s_0 s_1 \ldots \mid \forall j \exists i > j : s_i \in L\}$.
- **Streett emptiness.** The objective is specified by a set of request-grant pairs $\langle R_i, G_i \rangle$ (where each pair consists of a set $R_i \subseteq S$ of *request* control states and a set G_i of *grant* control states). The objective asks if there exists a feasible trace in the system such that for every request-grant pair, either G_i is visited infinitely often or R_i is visited finitely often. Formally, given a set of Streett pairs $P = \{\langle R_0, G_0 \rangle, \langle R_1, G_1 \rangle, \ldots, \langle R_m, G_m \rangle\}$, we have $\Phi = \texttt{Streett}(P) = \{s_0 s_1 \ldots \mid \forall 0 \leq i \leq m : [(\forall p \exists q > p : s_q \in G_i) \vee (\exists p \forall q > p : s_q \notin R_i)]\}$.

We call the traces which satisfy the finite-automaton, Büchi, and Streett conditions, accepting, Büchi, and Streett traces, respectively.

3.2 (Battery VS. Ideal-Energy) Transition Systems

Due to the recovery effect, BTSs behave qualitatively differently from a simple-energy transition systems. Nevertheless, in the domain where the energy in the battery is high, they do behave similarly. This lets us solve problems related to unlimited initial credit (referred to as "unknown initial credit") by reducing them to the simple-energy system problems.

Different Behavior. The BTS $\mathcal{B}_1$ of Fig. 1 demonstrates a key difference from a simple-energy system – the total energy in the initial state is 32, while a transition of weight (-17) cannot be taken, since the available energy is only 16. Yet, taking the cycle through states s_2 and s_3 reduces the total energy, but allows the (-17)-transition. After the cycle, the battery status is $(19\frac{3}{4}, 11\frac{1}{4})$, which becomes $(\frac{5}{8}, 13\frac{3}{8})$ following a (-17)-transition.

We formalize the difference in the theorem below. It is known that if an ideal-energy system contains an infeasible trace, it contains a memoryless infeasible trace [8]. A memoryless trace is one where a control state is always followed by the same control state. However, an infeasible trace in the system $\mathcal{B}_2$ of Fig. 1 must make different choices at different visits in state s_0.

Theorem 3. *A battery transition system may have feasible (resp. infeasible) traces without having any memoryless feasible (resp. infeasible) traces.*

Proof. Consider the BTS $\mathcal{B}_1$ in Figure 1. There is a trace for reaching s_1, as well as an infinite trace, however both traces make non-uniform choices at different visits in s_0. Analogously, an infeasible trace in the system $\mathcal{B}_2$ of Figure 1 must make different choices at s_0. We prove below the claim for $\mathcal{B}_2$. Analogous arguments and calculations can be made with respect to $\mathcal{B}_1$.

The only nondeterminism in $\mathcal{B}_2$ is in state s_0, allowing transitions to states s_1 and s_2. A trace that first chooses s_1 is legal, since the (-15)-transition is feasible from the initial status, after which there are only positive-weight transitions. The other memoryless option, of always choosing s_2, is also legal: Let (x, y) be the battery status when first reaching s_2. It can be shown that the first few cycles $s_2 \to s_3 \to s_0 \to s_2$ are feasible, after which the battery status will be (x', y'), such that $x' > x$ and $y' > y$. By the monotonicity of the $\texttt{Post}$ function, it follows that the cycle can be repeated forever.

On the other hand, first choosing s_2 and then choosing s_1 makes an illegal trace: The battery status when returning to s_0 is $(12\frac{3}{4}, 20\frac{1}{4})$, which changes to $(-\frac{5}{8}, 18\frac{3}{8})$ after the (-15)-transition. □

High energy domain and unknown initial credit problems. In energy systems, one often considers "unknown-initial-credit problems", asking if there is some initial energy that allows accomplishing a task. It is clear that every control state of a battery system can be reached, at least once, if there is a path leading to it and enough initial energy to start with. This is formalized in the following lemma (which will also serve us in Theorem 5 and in Section 4).

Lemma 4. *Consider a BTS $\mathcal{B} = \langle\langle S, \Delta, s_\iota, \nu \rangle, k, c\rangle$. There exist constants $\texttt{HighEnergyConstant}(\mathcal{B}, i)$ for every $i \in \mathbb{N}$ such that for every feasible extended state $(s_0, [\texttt{e}; \texttt{d}])$ with $\texttt{e} > \texttt{HighEnergyConstant}(\mathcal{B}, i)$, every weight sequence $w_0 w_1 \ldots w_{i-1}$ of length i is feasible from $(s_0, [\texttt{e}; \texttt{d}])$.*

Proof. We define the constants inductively as follows:
(a) $\texttt{HighEnergyConstant}(\mathcal{B}, 0) = 0$ if $i = 0$; and
(b) $\texttt{HighEnergyConstant}(\mathcal{B}, i) =$
$\max\left(\texttt{HighEnergyConstant}(\mathcal{B}, i-1) + W, \frac{W}{c(1-\lambda)}\right)$ otherwise. Here, $W = \max_{(s,s') \in \Delta} |\nu((s, s'))|$.

We prove the theorem by induction. When $i = 0$, the weight sequence is empty and there is nothing to prove.

For the induction case, assume that we have shown the result up to $i - 1$. Let $\texttt{Post}([\texttt{e}; \texttt{d}], w) = [\texttt{e}'; \texttt{d}']$. From Proposition 1, we get that $\texttt{e}' = \texttt{e} + w$ and $\texttt{d}' = \lambda \texttt{d} + w$. If $[\texttt{e}'; \texttt{d}']$ is feasible and $\texttt{e}' \geq \texttt{HighEnergyConstant}(\mathcal{B}, i-1)$, we can apply the induction hypothesis on the weight sequence $w_1 \ldots w_{i-1}$ to prove the result. We show these facts below.

- We have $\texttt{e}' = \texttt{e} + w > \texttt{HighEnergyConstant}(\mathcal{B}, i) + w \geq \texttt{HighEnergyConstant}(\mathcal{B}, i-1) + W + w$. As $w \geq -W$, we have that $\texttt{e}' > \texttt{HighEnergyConstant}(\mathcal{B}, i-1) \geq 0$.
- Further, we have that $c\texttt{e}' + (1-c)\texttt{d}' = c\texttt{e} + cw + (1-c)\lambda\texttt{d} + (1-c)w$ or, equivalently, $c\texttt{e}' + (1-c)\texttt{d}' = (1-\lambda)c\texttt{e} + w + \lambda(c\texttt{e} + (1-c)\texttt{d})$. As $[\texttt{e}; \texttt{d}]$ is feasible, by Proposition 1, $c\texttt{e} + (1-c)\texttt{d} > 0$. Further, $\texttt{e} > \texttt{HighEnergyConstant}(\mathcal{B}, i) \geq \frac{W}{c(1-\lambda)}$. Using these, we get $c\texttt{e}' + (1-c)\texttt{d}' > -W + w \geq 0$. By Lemma 2, $[\texttt{e}'; \texttt{d}']$ is feasible, completing the proof.

□

The following theorem states that the emptiness problems for battery systems reduce to the corresponding problems for energy systems if the initial energy is large enough.

Theorem 5. *Let $\mathcal{B} = \langle\langle S, \Delta, s_\iota, \nu \rangle, k, c\rangle$ be a BTS, $W = \max_{(s,s') \in \Delta} |\nu((s, s'))|$, and T, L and $\{\langle R_0, G_0 \rangle, \ldots, \langle R_n, G_n \rangle\}$ be a set of target states, a set of Büchi states, and a set of Streett pairs, respectively. There exist constants $M_R = \texttt{HighEnergyConstant}(\mathcal{B}, |S|)$, $M_B = \texttt{HighEnergyConstant}(\mathcal{B}, 3|S| + 2W|S|^2)$, and $M_S = \texttt{HighEnergyConstant}(\mathcal{B}, |S| + |S|^2 + W|S|^2 + W|S|^3)$ such that for any extended state $(s, [\texttt{e}; \texttt{d}])$: if $\texttt{e} > M_R$ (resp. $\texttt{e} > M_B$ and $\texttt{e} > M_S$), a feasible accepting (resp. Büchi and Streett) trace starting from $(s, [\texttt{e}; \texttt{d}])$ exists iff an energy-feasible accepting (resp. Büchi and Streett) trace exists.*

Proof of theorem 5. In all three parts, the existence of an energy feasible trace is directly implied by the existence of a feasible trace. Therefore, we only deal with showing that the existence of an energy feasible trace implies the existence of a feasible trace.

The first part (finite-automaton emptiness) follows directly from Lemma 4. If there exists a path from s to the target set T, there exists a path of length at most $|S|$. Taking $M_R = \texttt{HighEnergyConstant}(\mathcal{B}, |S|)$ is sufficient to give us the result.

For the second part (Büchi emptiness), the existence of an energy feasible trace implies that there exists a reachable cycle which visits a Büchi state and has non-negative total weight. It can be shown that the length of such a cycle can be bounded by $2|S| + 2W|S|^2$. Now, if $\mathtt{e} > \mathtt{HighEnergyConstant}(\mathcal{B}, |S| + 2|S| + 2W|S|^2)$, we can show, using Lemma 4, that there is a feasible path from $(s, [\mathtt{e};\mathtt{d}])$ to some $(s', [\mathtt{e}';\mathtt{d}'])$ where s' is on the cycle and $\mathtt{e}' > \mathtt{HighEnergyConstant}(\mathcal{B}, 2|S| + 2W|S|^2)$. Now, the cycle is feasible from $(s', [\mathtt{e}';\mathtt{d}'])$ and further, on returning to s', the energy is at least $\mathtt{e}'$ (as the total weight of the cycle is non-negative). Using such reasoning, it is easy to see that the cycle is repeatedly feasible from $(s', [\mathtt{e}';\mathtt{d}'])$. Hence, a value of $M_B = \mathtt{HighEnergyConstant}(\mathcal{B}, |S| + 2|S| + 2W|S|^2)$ is sufficient.

The proof for the third part is very similar to the proof of the second part except for the bound on the length of the cycle.

□

A straightforward consequence of the above theorem is that the unknown initial credit problems for BTSs are decidable. Furthermore, using the above theorem, it is easy to show that these BTS problems are equivalent to the corresponding energy-systems problems, which can be solved in polynomial time [6].

Corollary 6. *Given a BTS $\mathcal{B}$ and a finite-automaton, Büchi, or Streett condition, the problem of whether there is an initial battery status* $[\mathtt{e};\mathtt{d}]$*, such that there exists a feasible trace in $\mathcal{B}$ satisfying the condition is decidable in polynomial time.*

4. The Bounded-Energy Reachability Tree

Our algorithms for solving the emptiness problems are based on representing the infinite tree of all the possible traces in the BTS in a finite tree that summarizes all required information. The construction of the tree uses a "high-energy constant" – exploration from states whose energy is above the constant is stopped, as they can be further handled by a reduction to a simple-energy system using Theorem 5. Hence, the tree summarizes bounded-energy reachability, and we denote it BERT. As in Theorem 5, the value of the high-energy constant depends on the problem to be solved. We describe how we construct the tree, taking the high-energy constant as a parameter. We start with some basic lemmata about a total order among battery statuses of equal energy. Then, we present the 0-cycle saturation lemma, which helps summarize unbounded iterations of cycles in a finite manner.

Feasibility order for battery statuses. The following lemma shows that there exists a total order on the set of battery statuses with the same energy such that every weight sequence feasible from a lower battery status is also feasible from a higher battery status.

Lemma 7. *Given two battery statuses* $[\mathtt{e};\mathtt{d}]$ *and* $[\mathtt{e};\mathtt{d}']$ *with* $\mathtt{d} > \mathtt{d}'$*, every weight sequence* $w_0 w_1 \ldots w_{n-1}$ *feasible from* $[\mathtt{e};\mathtt{d}']$ *is also feasible from* $[\mathtt{e};\mathtt{d}]$*. Furthermore, if* $\mathtt{Post}([\mathtt{e};\mathtt{d}], w_0 \ldots w_{n-1}) = [\mathtt{e}'';\mathtt{d}'']$ *and* $\mathtt{Post}([\mathtt{e};\mathtt{d}'], w_0 \ldots w_{n-1}) = [\mathtt{e}'';\mathtt{d}''']$*, we have* $\mathtt{d}'' > \mathtt{d}'''$.

Proof. We prove the result by induction. For the base case, we let the weight sequence be of length 1, i.e., w_0. From Proposition 1, we know that $\mathtt{Post}([\mathtt{e},\mathtt{d}], w_0) = [\mathtt{e} + w_0, \lambda\mathtt{d} + w_0]$ and $\mathtt{Post}([\mathtt{e},\mathtt{d}'], w_0) = [\mathtt{e} + w_0, \lambda\mathtt{d}' + w_0]$. Hence, $\mathtt{d}'' = \lambda\mathtt{d} + w_0$ and $\mathtt{d}''' = \lambda\mathtt{d}' + w_0$. As $\mathtt{d} > \mathtt{d}'$, we get that $\lambda\mathtt{d} + w_0 > \lambda\mathtt{d}' + w_0$ and $\mathtt{d}'' > \mathtt{d}'''$.

Now, assume that $[\mathtt{e} + w_0; \lambda\mathtt{d}' + w_0]$ is feasible. By Lemma 2, we get that $c(\mathtt{e}+w_0) + (1-c)(\lambda\mathtt{d}' + w_0) > 0$. As $d > d'$, we have $c(\mathtt{e}+w_0) + (1-c)(\lambda\mathtt{d} + w_0) > 0$ and $c(\mathtt{e}+w_0) + (1-c)\mathtt{d}'' > 0$. This gives us that $[\mathtt{e} + w_0; \mathtt{d}'']$ is feasible, hence completing the proof for the base case.

Now, assume that the required result holds for every weight sequence of length $n-1$. Applying the induction hypothesis on the battery statuses $[\mathtt{e} + w_0; \lambda\mathtt{d} + w_0]$ and $[\mathtt{e} + w_0; \lambda\mathtt{d}' + w_0]$ and the weight sequence $w_1 \ldots w_{n-1}$ gives us the required result. □

Guided by Lemma 7 above, we define a partial order on the set of battery statuses as follows: $[\mathtt{e}';\mathtt{d}'] \sqsubseteq [\mathtt{e};\mathtt{d}]$ if $\mathtt{e} = \mathtt{e}'$ and $\mathtt{d}' \leq \mathtt{d}$, in which case we say that $[\mathtt{e};\mathtt{d}]$ *subsumes* $[\mathtt{e}';\mathtt{d}']$. We extend the partial order $\sqsubseteq$ to extended states (with both control states and battery statuses) by letting $(s', [\mathtt{e}';\mathtt{d}']) \sqsubseteq (s, [\mathtt{e};\mathtt{d}])$ if $s = s'$ and $[\mathtt{e}';\mathtt{d}'] \sqsubseteq [\mathtt{e};\mathtt{d}]$. Lemma 7 can now be restated as follows: *If* $(s', [\mathtt{e}';\mathtt{d}']) \sqsubseteq (s, [\mathtt{e};\mathtt{d}])$*, every control path feasible from* $(s', [\mathtt{e}';\mathtt{d}'])$ *is also feasible from* $(s, [\mathtt{e};\mathtt{d}])$.

Zero-cycle saturation. We formalize below the key observation that 0-energy cycles can be finitely summarized: an infinite run along such a cycle monotonically converges to a fixed battery status. Moreover, the deviation in the limit is independent of the initial status.

Lemma 8 (Zero-cycle saturation)**.** *Let* $w_0 \ldots w_{n-1}$ *be a sequence of weights such that* $\sum_{i=0}^{n-1} w_i = 0$ *and let* $\mathbf{t_0}, \mathbf{t_1}, \ldots$ *be a sequence of tuples in* $\mathbb{R}^2$*, such that* $\mathbf{t_{i+1}} = \mathtt{Post}(\mathbf{t_i}, \mathbf{w_0} \ldots \mathbf{w_{n-1}})$*. We have the following:*

1. *The sequence* $\mathbf{t_0}, \mathbf{t_1}, \ldots$ *converges (say to* $\mathbf{t}^* = [\mathtt{e}^*;\mathtt{d}^*]$*). In other words,* $\forall \epsilon \geq 0. \exists m \in \mathbb{N} : |\mathbf{t}^* - \mathbf{t_m}|_1 \leq \epsilon$ *where* $|.|_1$ *denotes the maximum absolute component in a vector.*
2. *We have* $\forall i \in \mathbb{N} : \mathbf{t}^* \sqsubseteq \mathbf{t_i} \sqsubseteq \mathbf{t_0}$ *or* $\forall i \in \mathbb{N} : \mathbf{t_0} \sqsubseteq \mathbf{t_i} \sqsubseteq \mathbf{t}^*$*. In the latter case, if* $w_0 \ldots w_{n-1}$ *is feasible from* $\mathbf{t_0}$ *it is feasible from each* $\mathbf{t_i}$.

Proof. Let $\mathbf{t_i} = [\mathtt{e}_i;\mathtt{d}_i]$. Obviously, $\forall i. \mathtt{e}_i = \mathtt{e}^*$. By repeated application of Proposition 1 on the weights $w_0 w_1 \ldots w_{n-1}$, starting with $\mathbf{t_i}$, we have $\mathtt{d}_{i+1} = \mathtt{d}_i \cdot \lambda^n + \sum_{p=0}^{n-1} w_p \cdot \lambda^{n-1-p}$. Hence, for all $i \in \mathbb{N}$, $\mathtt{d}_i = \mathtt{d}_0 \cdot \lambda^{i \cdot n} + \sum_{q=0}^{i-1} L \cdot \lambda^{n \cdot q}$, where $L = \sum_{p=0}^{n-1} w_p \cdot \lambda^{n-1-p}$. From this, it follows that the sequence $\mathbf{d_i}$ converges to $\mathbf{d}^* = \sum_{q=0}^{\infty} L \cdot \lambda^{q \cdot n} = L \cdot \frac{1}{1-\lambda^n}$. Further,

$$\begin{aligned}
\mathtt{d}_{(i+1)} &= \mathtt{d}_0 \cdot \lambda^{i \cdot n + n} + L \cdot \sum_{q=0}^{i} \lambda^{n \cdot q} \\
&= \mathtt{d}_0 \cdot \lambda^{i \cdot n} + L \cdot \sum_{q=0}^{i-1} \lambda^{n \cdot q} + \mathtt{d}_0 \cdot [\lambda^{i \cdot n + n} - \lambda^{i \cdot n}] + L \cdot \lambda^{i \cdot n} \\
&= \mathtt{d}_i + \mathtt{d}_0 \cdot [\lambda^{i \cdot n + n} - \lambda^{i \cdot n}] + L \cdot \lambda^{i \cdot n} \\
&= \mathtt{d}_i + \lambda^{i \cdot n}(1 - \lambda^n) \cdot [\frac{L}{1 - \lambda^n} - \mathtt{d}_0] \\
&= \mathtt{d}_i + \lambda^{i \cdot n}(1 - \lambda^n) \cdot [d^* - \mathtt{d}_0]
\end{aligned}$$

Since $\forall i. \lambda^{i \cdot n}(1 - \lambda^n)$ is positive, it follows that d_{i+1} is bigger, or not, than d_i based on whether $d^* < \mathtt{d}_0$ or not. If $d^* < \mathtt{d}_0$, we get $\mathtt{d}_0 > \mathtt{d}_1 > \ldots > d^*$, or, equivalently, $\mathbf{t_0} \sqsupseteq \mathbf{t_1} \sqsupseteq \ldots \sqsupseteq \mathbf{t}^*$. Similarly, if $d^* \geq \mathtt{d}_0$, we get $\mathtt{d}_0 \leq \mathtt{d}_1 \leq \ldots \leq d^*$, or, equivalently, $\mathbf{t_0} \sqsubseteq \mathbf{t_1} \sqsubseteq \ldots \sqsubseteq \mathbf{t}^*$. The feasibility of $w_0 \ldots w_n$ from each $\mathbf{t_i}$ follows from Lemma 7 and $\mathbf{t_i} \sqsupseteq \mathbf{t_0}$. □

We denote the limit $\mathtt{deviation}$ $\mathbf{d}^*$ as $\mathtt{Saturate}(w_0 w_1 \ldots w_{n-1})$, i.e., $\mathtt{Saturate}(w_0 w_1 \ldots w_{n-1}) = \frac{1}{1-\lambda^n} \cdot \left(\sum_{p=0}^{n-1} w_p \cdot \lambda^{n-1-p}\right)$. Note that this deviation does not depend on the initial battery status $\mathbf{t_0}$. Accordingly, we extend the definition of the function $\mathtt{Saturate}$ to battery statuses as $\mathtt{Saturate}([\mathtt{e};\mathtt{d}], w_0 w_1 \ldots w_{n-1}) = [\mathtt{e}; \mathtt{Saturate}(w_0 w_1 \ldots w_n)]$.

Constructing the tree. For generating a finite tree with all the relevant bounded energy reachability information, we explore the feasible states and transitions starting from the initial state. However,

1. Extended states with high-enough energies are not explored further, and
2. If an extended state $\mathbf{q}$ that has an ancestor $\mathbf{q}'$ with the same control state and the same energy (but possibly a different deviation) is reached, we check the feasibility order, i.e., if $\mathbf{q} \sqsubseteq \mathbf{q}'$ or $\mathbf{q}' \sqsubseteq \mathbf{q}$. If $\mathbf{q} \sqsubseteq \mathbf{q}'$ we stop exploration from $\mathbf{q}$; otherwise, we saturate this 0-energy cycle from $\mathbf{q}'$ to $\mathbf{q}$, i.e., calculate the fixed battery status $\mathbf{t}^*$ to which an infinite run on that cycle will monotonically converge to (see Lemma 8). Then, we replace the battery status in $\mathbf{q}'$ with the maximum between battery status in $\mathbf{q}$ and $\mathbf{t}^*$.

Algorithm 1 `ComputeBERT`: Computing the bounded-energy reachability tree

```
Require: Battery system $\mathcal{B} = \langle\langle S, \Delta, s_\iota, \nu\rangle, k, c\rangle$, initial battery
    status $\mathbf{t}$, energy bound $M$
 1: BERT ← EmptyTree
 2: BERT.$root.label$ ← $(s_\iota, \mathbf{t})$;  BERT.$root.star$ ← false
 3: OpenNodes ← {BERT.$root$}
 4: while OpenNodes ≠ ∅ do
 5:   Pick and remove currNode from OpenNodes
 6:   $(s_0, [\mathtt{e};\mathtt{d}])$ ← currNode.$label$
 7:   // If we found a good cycle, saturate that cycle
 8:   if currNode has an ancestor ancestor with label $(s_0, [\mathtt{e};\mathtt{d}'])$ then
 9:     if $\mathtt{d} > \mathtt{d}'$ then
10:       $s_0 \dots s_n s_0$ ← control state sequence in node labels from ancestor to currNode
11:       $w_0 \dots w_n \leftarrow \nu((s_0,s_1))\nu((s_1,s_2))\dots\nu((s_n,s_0))$
12:       if $w_0 w_1 \dots w_n$ is feasible from $[e;d']$ and Saturate($w_0 \dots w_n$) > d then
13:         ancestor.$label$ ←$(s_0$, [e; Saturate($w_0 \dots w_n$)]);
14:         ancestor.$star$ ← true
15:       else {There was another cycle saturation between ancestor and currNode}
16:         $ancestor.label$ ← [e; d]
17:       BERT.$delete$(all descendants of ancestor)
18:       OpenNodes ← OpenNodes ∪ {ancestor} \ all descendants of ancestor
19:     continue; // If d < d′ there is no further exploration from the current node
20:   else {Explore one step further}
21:     for all $(s_0, s') \in \Delta$ do
22:       if Post([e; d], $\nu(s_0, s')$) is feasible then
23:         newNode ← new child of currNode
24:         newNode.$label$ ← $(s'$, Post([e; d], $\nu(s_0, s')$));
25:         newNode.$star$ ← false
26:         if energy(Post([e; d], $\nu(s_0, s')$)) ≤ $M$ then
27:           OpenNodes ← OpenNodes ∪ newNode
28: return BERT
```

The procedure `ComputeBERT` (Algorithm 1) computes the bounded-energy reachability tree `BERT`, given a BTM $\mathcal{B}$, an initial battery status $\mathbf{t}$, and an energy bound M. It is a rooted tree where each node is labelled with an extended state. During the procedure's execution, each node in the tree is either open (in `OpenNodes`) or closed, and exploration will only continue from open nodes. In addition, each node contains a Boolean field $star$, marking whether its label is a result of saturation. Initially, the root of `BERT` is labelled with the initial extended state $(s_\iota, \mathbf{t})$ and the root node is added to the set of `OpenNodes`.

In each step, one open node (`currNode`) is picked and removed from the set of `OpenNodes`. Let `currNode`.$label = (s, [\mathtt{e};\mathtt{d}])$. By default, we append to `currNode` children labelled by all feasible successors of $(s, [\mathtt{e};\mathtt{d}])$ (we call this an exploration step). If one of the following holds, we do not perform the exploration step.

- In case the energy (i.e., $\mathtt{e}$) of `currNode` is greater than the given bound M, we stop exploration from it.
- In case an ancestor `ancestor` (with label $(s, [\mathtt{e};\mathtt{d}'])$) of `currNode` has the same control state and energy as `currNode`:
  - If $[\mathtt{e};\mathtt{d}] \sqsubseteq [\mathtt{e};\mathtt{d}']$ we stop exploration from `currNode`.
  - If $[\mathtt{e};\mathtt{d}'] \sqsubset [\mathtt{e};\mathtt{d}]$ we i) delete all the descendants of `ancestor`; and ii) replace the battery status in the label of `ancestor` with the $\sqsubset$-maximum between $[\mathtt{e};\mathtt{d}]$ and the zero-cycle saturation of $[\mathtt{e};\mathtt{d}']$, where $\mathtt{d}'$ is the 0-cycle saturation of the the weight sequence from `ancestor` to `currNode`. Note that if the weight sequence from `ancestor` to `currNode` is infeasible from $[\mathtt{e};\mathtt{d}']$ (which can happen if there is a saturation of another cycle between the `ancestor` and `currNode`), we replace $[\mathtt{e};\mathtt{d}']$ by $[\mathtt{e};\mathtt{d}]$ and not the maximum.

When no open nodes are left, the procedure stops and returns `BERT`. We prove in a series of lemmata the properties of the procedure `ComputeBERT` and of the returned tree.

Termination. We prove that Algorithm 1 terminates for every input, by showing a bound on both the number of possible nodes in `BERT` and the number of node deletions in an execution. The latter bound follows from (i) every deletion event strictly increases a deviation value; and (ii) the number of possible values that a deviation can get in a deletion event is bounded. Note that this is in contradiction to the unbounded number of deviations that may occur in a trace of the BTS $\mathcal{B}$.

Lemma 9. *Algorithm 1 terminates on all inputs.*

Proof. Termination follows from a bound on both the number of possible nodes in `BERT` and the number of node deletions in an execution.

The number of nodes in a tree depends on the fanout of the nodes and the length of the branches. The fanout of every node in `BERT` is bounded by the number of states in the BTS $\mathcal{B}$. As for the branches, the control state and energy of each node in a path is unique, except for the leaf, giving a bound of $|S| \times M + 1$ to the length of a path.

The bound on the number of deletions follows from (i) every deletion event strictly increases a deviation value; and (ii) the number of possible values that a deviation can get in a deletion event is bounded. Note that this is in contradiction to the unbounded number of deviations that may occur in a trace of the BTS $\mathcal{B}$.

Consider a node-deletion event in the execution of Algorithm 1, and a new deviation value set to `ancestor`.$label$.
(i) The new value is either `Saturate`$(w_0 \dots w_n)$ or $\mathtt{d}$, set in Line 14 or Line 16, which are in the scope of "if `Saturate`$(w_0 \dots w_n) > \mathtt{d} > \mathtt{d}'$" or "if $\mathtt{d} > \mathtt{d}'$", respectively, while the old value is $\mathtt{d}'$.
(ii) The new value is uniquely determined by the following:

- The value of `Saturate`$(w_0 \dots w_n)$, where $w_0 \dots w_n$ is the sequence of weights from `ancestor` to `currNode`; or
- The label of the last saturated (i.e., starred) node between `ancestor` and `currNode`, and the suffix $w_i \dots w_n$ of $w_0 \dots w_n$ corresponding to the segment from the last starred node to `currNode`, otherwise.

In the first case, resulting from Line 14, the new value only depends on $w_0 \dots w_n$, which in turn is determined by $s_0 s_1 \dots s_n s_0$. In the second case, resulting from Line 16, let $(s_i, [\mathtt{e}^{(i)};\mathtt{d}^{(i)}])$ be the label of the last starred node. By Lemma 8, $[\mathtt{e}^{(i)};\mathtt{d}^{(i)}]$ only depends on the sequence of weights in a simple cycle of the BTS $\mathcal{B}$, thus may take a bounded number of possible values. The new value is then calculated by `Post`$([\mathtt{e}^{(i)};\mathtt{d}^{(i)}], w_i \dots w_n)$, which only depends on $[\mathtt{e}^{(i)};\mathtt{d}^{(i)}]$ and the sequence $w_i \dots w_n$. □

Correctness. We now prove that BERT is a summarization of all extended states reachable through states of low-energy. Let $\texttt{Reach}(M)$ be the set of extended states reachable from the initial state of the BTS $\mathcal{B}$ through paths containing only extended states of energy less than M. In the lemmata below, we prove the following:

- *Soundness.* For every node node in BERT and for all $\epsilon > 0$, there is an extended state $\mathbf{q} \in \texttt{Reach}(M)$ such that $\mathbf{q} \sqsubseteq \texttt{node}.label$ and the difference between the deviations of $\mathbf{q}$ and $\texttt{node}.label$ is smaller than ϵ.
- *Completeness.* For every extended state $\mathbf{q} \in \texttt{Reach}(M)$, there exists a node node in BERT such that $\mathbf{q} \sqsubseteq \texttt{node}.label$.

Lemma 10 (Soundness)**.** *For every node* node *with label* $(s, [\mathtt{e};\mathtt{d}])$ *encountered in an execution of Algorithm 1 and* $\epsilon > 0$*, there exists an extended state* $(s, [\mathtt{e};\mathtt{d}-\delta])$ *reachable from* $(s_\iota, \mathbf{t})$ *with* $0 \leq \delta < \epsilon$.

Proof. The claim can be proved by looking at all the points in the algorithm where a new label is created (lines 10, 14, and 25). The claim is trivially true for the initial label of the root. Assume as induction hypothesis that the claim holds for every label encountered upto the current point of the execution. Fix $\epsilon \geq 0$.
Saturation. Let the new label be created during a deletion event, in line 10 or line 14. In the case the label of currNode is copied to the label of ancestor, the proof follows immediately as we are just copying an existing label. Otherwise, we are taking the Saturate value, and the proof is based on Lemma 8. By iterating the path from ancestor to currNode a sufficient number of times, we can get as close as necessary (i.e., within ϵ) to the limit of the zero-weight cycle saturation. Lemma 8 also gives us that every iteration is feasible.
Exploration. Let $(s', [\mathtt{e}';\mathtt{d}'])$ be a new node label created in line 25. Choosing $\epsilon' < \min(\frac{\epsilon}{\lambda}, c\mathtt{e}' + (1-c)\mathtt{d}')$, by the induction hypothesis, there is a feasible path from $(s_\iota, \mathbf{t})$ to $(s, [\mathtt{e};\mathtt{d}-\delta'])$ with $0 \leq \delta' < \epsilon'$.

Now, we have $[\mathtt{e}';\mathtt{d}'] = \texttt{Post}([\mathtt{e};\mathtt{d}], w) = [\mathtt{e}+w; \lambda\mathtt{d}+w]$ and $\texttt{Post}([\mathtt{e};\mathtt{d}-\delta'], w) = [\mathtt{e}';\mathtt{d}' - \lambda\cdot\delta']$. Letting $\delta = \lambda\delta'$, we get $0 \leq \delta = \lambda\delta' < \lambda\epsilon' \leq \epsilon$. If we prove that $[\mathtt{e}';\mathtt{d}'-\delta]$ is feasible, we are done as we have shown that the path from $(s_\iota, \mathbf{t})$ to $(s, [\mathtt{e};\mathtt{d}-\delta'])$ followed by the feasible transition from $(s, [\mathtt{e};\mathtt{d}-\delta'])$ to $(s', [\mathtt{e}';\mathtt{d}'-\delta])$ is a path from $(s_\iota, \mathbf{t})$ to $(s', [\mathtt{e}';\mathtt{d}'-\delta])$.

As $[\mathtt{e}';\mathtt{d}']$ is feasible, we get that $c\mathtt{e}' + (1-c)\mathtt{d}' > 0$. As we chose that $\epsilon' < c\mathtt{e}' + (1-c)\mathtt{d}'$, we get that $c\mathtt{e}' + (1-c)\mathtt{d}' - \delta = c\mathtt{e}' + (1-c)\mathtt{d}' - \lambda\delta' > c\mathtt{e}' + (1-c)\mathtt{d}' - \lambda\epsilon' > c\mathtt{e}' + (1-c)\mathtt{d}' - \lambda(c\mathtt{e}' + \mathtt{d}') > (1-\lambda)(c\mathtt{e}' + (1-c)\mathtt{d}') > 0$. This completes the proof for this case.

□

Lemma 11 (Completeness)**.** *Let* $(s, [\mathtt{e};\mathtt{d}])$ *be an extended state with* $\mathtt{e} < M$ *that is reachable from* $(s_\iota, \mathbf{t})$ *through extended states with energy less than* M*. Then, there exists a node with label* $(s, [\mathtt{e};\mathtt{d}'])$ *in* BERT *with* $\mathtt{d}' \geq \mathtt{d}$.

Proof. We prove the lemma by induction on the length of the path from $(s_\iota, \mathbf{t})$ to $(s, [\mathtt{e};\mathtt{d}])$. For paths of length 0, it is trivial as $(s_\iota, \mathbf{t})$ is the initial label of the root. Suppose we have proven the claim for paths upto the length $n-1$.

It is easy to induct to length n. Suppose the path of length $n-1$ ending with $(s, [\mathtt{e};\mathtt{d}])$ is extended to length n by adding $(s^\sharp, [\mathtt{e}^\sharp;\mathtt{d}^\sharp])$. The proof has two cases:

- If the node labelled with $(s, [\mathtt{e};\mathtt{d}'])$ is a non-leaf node, then it has a successor labelled $(s^\sharp, [\mathtt{e}^\sharp;\mathtt{d}^{\sharp'}])$, with $\mathtt{d}^{\sharp'} > \mathtt{d}^\sharp$, which is the required node. This follows from lines 21–27 of Algo. 1.
- If the BERT node labelled with $(s, [\mathtt{e};\mathtt{d}'])$ is a leaf node, it will have an ancestor labelled $(s, [\mathtt{e};\mathtt{d}''])$ with $\mathtt{d}'' \geq \mathtt{d}$. This follows from line 19 of Algorithm 1. Since the latter node is not a leaf, we comply with the previous case, and we are done.

□

5. Model Checking

We are now ready to tackle the finite-automaton, Büchi, and Streett emptiness problems for BTSs. We show that the problems are decidable and give suitable algorithms. The algorithms are based on Theorem 5 and analysis of the bounded-energy reachability tree, as constructed in Section 4.

5.1 Finite-Automaton Emptiness

Combining the results from the previous section on bounded energy reachability tree and Theorem 5, we can obtain a complete algorithm for the finite-automaton emptiness problem in a battery system. Given a BTS $\mathcal{B}$ with states S, an initial battery status $\mathbf{t}$, the algorithm works as follows:

- Build a bounded-energy reachability tree
  $\texttt{BERT} = \texttt{ComputeBERT}(\mathcal{B}, (s_\iota, \mathbf{t}), M)$, where
  $M = \texttt{HighEnergyConstant}(\mathcal{B}, |S|))$;
- If there is a node label $(s, [\mathtt{e};\mathtt{d}])$ in BERT where s is in the target set T, return true;
- If there is a node label $(s, [\mathtt{e};\mathtt{d}])$ in BERT where $\mathtt{e} > M$, and some node in the target set is reachable from $(s, [\mathtt{e};\mathtt{d}])$ through an energy-feasible path, return true;
- Otherwise, return false.

The correctness proof of the algorithm follows from Lemma 4 and the soundness and completeness of the bounded-energy reachability tree (Lemmas 10–11). The following theorem states that this algorithm can be implemented in polynomial space in the inputs.

Theorem 12. *The finite-automaton emptiness problem for BTSs is decidable in polynomial space with respect to the number of control states in the BTS and a unary encoding of weights.*

Proof. The major part of the algorithm is the construction of the bounded-energy reachability tree. For a given energy bound M, this tree can contain an exponential number of nodes in M. However, using standard on-the-fly techniques, we can reduce the space complexity, only storing the current branch of the tree being explored. The corresponding space is the product of the number of nodes in each branch and the bits required for storing a node's label.

By the proof of Lemma 9, the length of each branch in the tree is bounded by $|S| \times M + 1$, where S are the states of the given BTS $\mathcal{B}$. For the finite-automaton emptiness algorithm, we use $M = \texttt{HighEnergyConstant}(\mathcal{B}, |S|)$ and by the proof of Lemma 4, we have $M \leq |S|W + \frac{W}{c(1-\lambda)}$, where W is the maximal negative weight in the BTS. With a unary encoding of the constants, M is polynomial in the size of the input.

To complete the proof, we need to show that all the labels created in BERT can be represented in polynomial space in the energy bound M.

A label contains a control state, an energy, and a deviation. There are $|S| < M$ control states and up to M different energies.

As for the deviations, they are generated by a sequence of operations, involving two functions: Post (defined in Section 3) and Saturate (defined in Lemma 8). By Lemma 8, the value of Saturate is independent of the deviation value before saturation. Hence, the deviation at each node in BERT is a result of the last Saturate operation in the branch of BERT leading to the node, followed by some Post operations. By the proof of Lemma 9, the length of each branch in the tree is polynomial in M, implying up to M applications of Post. Hence, it is left to show that the Saturate function generates a deviation that can be stored in space polynomial in M, and that each application of the Post function adds up to b bits, where b is polynomial in M.

By Lemma 8, given a sequence $w_0 w_1 \dots w_{n-1}$ of weights, $\texttt{Saturate}(w_0 w_1 \dots w_{n-1}) = \frac{1}{1-\lambda^n} \cdot \left(\sum_{p=0}^{n-1} w_p \cdot \lambda^{n-1-p} \right)$. The space required to store this value is polynomial in the constant λ and n, where $n \leq |S| < M$. In each application of Post on a battery status $[\mathtt{e}; \mathtt{d}]$ and weight w, we have, by Proposition 1, that $\texttt{Post}([\mathtt{e}; \mathtt{d}], w) = [\mathtt{e}'; \mathtt{d}']$, with $d' = \lambda \cdot \mathtt{d} + w$. Hence, storing $\mathtt{d}'$ requires up to b bits more than storing $\mathtt{d}$, where b is polynomial in the constant λ and $|w| < M$.

□

5.2 Büchi and Streett Emptiness

Suppose we are given a BTS $\mathcal{B} = \langle \langle S, \Delta, s_\iota, \nu \rangle, k, c \rangle$ and a Büchi condition given by a set of Büchi states $B \subseteq S$. Our approach to Büchi emptiness consists of two major parts. If there exists a Büchi trace containing an extended state with energy more than $M_B = 3|S| + 2W|S|^2$, the problem can be reduced to the Büchi problem for simple-energy systems (Theorem 5). Therefore, we concentrate on the case where the energy of states is bounded by M_B. Here, the key idea is that if the energies of the extended states are bounded, then a BTS has a Büchi trace if and only if it has a Büchi trace of a special form.

First, we define the notion of an energy-unique path: we call a control trace $s_0 s_1 \dots s_n$ *energy-unique* if we have $\sum_{i=0}^{p} \nu((s_i, s_{i+1})) \neq \sum_{i=0}^{q} \nu((s_i, s_{i+1})) \vee s_p \neq s_q$ for $p \neq q$. Intuitively, $s_0 s_1 \dots s_n$ is energy-unique if no trace whose corresponding control trace is $s_0 \dots s_n$ has two extended states with the same control state and equal energy. Similarly, $s_0 s_1 \dots s_n$ is an *energy-unique* 0*-energy cycle* if $s_0 s_1 \dots s_n$ is energy-unique and $\sum_{i=0}^{n-1} \nu((s_i, s_{i+1})) + \nu((s_n, s_0)) = 0$.

The following theorem intuitively states that if there exists a bounded-energy Büchi trace in $\mathcal{B}$, then there exists a lasso-shaped bounded-energy Büchi trace where the first state of the cycle in the lasso is a Büchi state and the cycle in the lasso has one of the two following forms:

- the cycle is an energy-unique 0-energy cycle $s_0^l \dots s_n^l s_0^l$, such that the sequence $s_0^l s_1^l \dots s_n^l s_0^l$ is feasible from s_0^l with the battery status $\texttt{Saturate}(w_0 w_1 \dots w_n)$, where $w_i = \nu((s_i^l, s_{i+1}^l))$ for $i < n$ and $w_n = \nu((s_n^l, s_0^l))$; or
- the cycle is an energy-unique 0-energy cycle composed of an alternating sequence of energy-unique paths and energy-unique 0-energy cycles. Here, every energy-unique 0-energy cycle in the sequence is unique.

Theorem 13. *Suppose a BTS $\mathcal{B}$ has a Büchi trace such that every extended state has energy less than some constant M. Then, $\mathcal{B}$ has a Büchi trace π such that the corresponding control trace θ has one of the following two forms:*

Form 1 $\theta = \theta_h (s_0^l s_1^l \dots s_n^l)^\omega$ *where s_0^l is a Büchi state, and $s_0^l s_1^l \dots s_n^l s_0^l$ is an energy-unique* 0*-energy cycle.*

Form 2 $\theta = \theta_h (\theta_l)^\omega$, *where $\theta_h, \theta_l \in S^*$ and $\theta_l = (s_0^0 \dots s_{k_0}^0)(\theta_{l0})^{r_0} \dots (s_0^n \dots s_{k_1}^n) \dots (\theta_{ln})^{r_n} (s_0^{n+1} \dots s_{k_{n+1}}^{n+1})$ and (a) each θ_{li} is a distinct energy-unique* 0*-energy cycle; (b) each $s_0^i \dots s_{k_i}^i$ is a energy-unique path; (c) θ_l is a* 0*-energy cycle; and (d) s_0^0 is a Büchi state.*

We omit the proof of Theorem 13 due to lack of space. The proof proceeds by taking a witness Büchi trace and reducing it to one of the two forms by deleting parts of the trace where the initial and final energies and control states are the same, while the final deviation is less than the initial deviation.

The algorithm. The Büchi-emptiness algorithm intuitively consists of two separate parts: (a) searching for high energy Büchi traces (where some extended state has energy more than $M_B = \texttt{HighEnergyConstant}(\mathcal{B}, 3|S| + 2W|S|^2)$); and (b) searching for low energy fair traces (where every extended state has energy less than M_B). The algorithm first constructs $\texttt{BERT} = \texttt{ComputeBERT}(\mathcal{B}, (s_\iota, \mathbf{t}), M_B)$.

High energy. For every node label $(s, [\mathtt{e}; \mathtt{d}])$ in the BERT where $\mathtt{e} > M_B$, we check (using techniques of [6]) whether there exists an energy-feasible fair trace from it.

Form 1 low energy. For every node label $(s, [\mathtt{e}; \mathtt{d}])$ in BERT with $\mathtt{e} \leq M_B$ and $s \in B$, we check if there exists a fair trace of Form 1 starting from $(s, [\mathtt{e}; \mathtt{d}])$. Performing this check entails constructing energy-unique 0-energy cycles θ_l starting from s and examining if θ_l is feasible from $\texttt{Saturate}((s, [\mathtt{e}; \mathtt{d}]), \theta_l)$.

Form 2 low energy. For every $s \in B$ and $\mathtt{e} < M_B$, we run Algorithm 2 with initial state s and initial battery status $[\mathtt{e}; \mathtt{d}]$ to check if there exists a fair trace of Form 2. Here, $\mathtt{d}$ is the maximum deviation of a node label which has control state s and energy $\mathtt{e}$.

Algorithm 2 Finding form 2 low energy fair traces

```
Require: Battery system $\mathcal{B} = \langle\langle S, \Delta, s_\iota, \nu\rangle, k, c\rangle$, Energy bound
    $M \in \mathbb{N}$, Control state $s$, Initial battery status $[\mathtt{e}; \mathtt{d}]$
 1: $d^* \leftarrow \mathtt{d}$
 2: while true do
 3:     BERT ← ComputeBERT($\langle\langle S, \Delta, s, \nu\rangle, k, c\rangle$, $[\mathtt{e}; \mathtt{d}^*]$, $M$)
 4:     $P \leftarrow \{$leaf.label | BERT leaf leaf has label $(s, [\mathtt{e}; \mathtt{d}']) \wedge$
            leaf has a starred ancestor $\}$
 5:     if $P = \emptyset$ then
 6:         return false
 7:     else if $d^* = \max\{d' \mid (s, [\mathtt{e}; \mathtt{d}']) \in P\}$ then
 8:         return true
 9:     else
10:         $d^* \leftarrow \max\{d' \mid (s, [\mathtt{e}; \mathtt{d}']) \in P\}$
```

Lemma 14. *Algorithm 2 returns* true *if $\mathcal{B}$ contains a Büchi trace of Form 2, and* false *otherwise.*

Intuitively, Algorithm 2 works by finding some deviation $\mathtt{d}^*$ such that $(s, [\mathtt{e}; \mathtt{d}^*])$ is feasibly reachable from itself through some number of 0-energy cycle saturations (represented by starred nodes). In every iteration of the while-loop, it decreases the possible value for $\mathtt{d}^*$ to the largest deviation for control-state s and energy $\mathtt{e}$ reachable from the previous value of $\mathtt{d}^*$ through some starred nodes. If $\mathtt{d}^*$ becomes so low that we are not able to saturate any 0-energy cycle starting from $(s, [\mathtt{e}; \mathtt{d}^*])$, then we return false.

Theorem 15. *Büchi emptiness for battery transition systems is decidable in polynomial space with respect to the number of states and a unary encoding of weights and constants.*

Equipped with Theorem 13, the proof of Theorem 15 follows in a similar fashion as in Theorem 12.

Using similar techniques, we can construct an algorithm for Streett emptiness. In this case, also keeping track of the set of states visited along each branch of the reachability tree.

Theorem 16. *Street emptiness for battery transition systems is decidable in polynomial space with respect to the number of states and a unary encoding of weights and constants.*

5.3 ω-Regular Model Checking

Equipped with a procedure for checking Büchi emptiness (Theorem 15), one can check whether a given BTS $\mathcal{B}$ satisfies any ω-regular constraint φ that is defined with respect to $\mathcal{B}$'s states. Such a constraint can be formalized, for example, by a linear temporal logic (LTL) formula, whose atomic propositions are the names of the states in $\mathcal{B}$. Indeed, any ω-regular constraint φ can be translated to a Büchi automaton $\mathcal{A}$, such that $\mathcal{A}$'s language is equivalent to the language of φ (or to the language of its negation, as is the common practice in the case of an LTL formula) [24]. Now, one can take the product of $\mathcal{A}$ and $\mathcal{B}$, defined in the usual way, getting a BTS $\mathcal{C}$ with a Büchi emptiness problem.

As Streett emptiness is a special case of ω-regular model checking, one may wonder why we bothered to have Theorem 16. The reason lies in the complexity – In Theorem 16, we show that Streett emptiness can be solved in the same complexity class as the one for Büchi emptiness, while translating a Streett automaton into a Büchi automaton might involve an exponential state blowup [23].

```
program := statements

statements := statement | statement; statements
statement := (label : tick) | action_var = expr
        | internal_var = expr | skip
        | if (expr == 0) statements else statements
        | while (expr == 0) statements
expr := sensor_var | input_var | internal_var
        | expr + expr | constant | expr * expr
        | (expr == 0) ? expr : expr
```

Figure 5. Syntax of the robot-control language

6. Case Study

We conclude this paper with a case study relating to controlling an energy-constrained robot. We first define a toy language for programming the robot controller, inspired by various real languages for programming robots, and define how the different constructs interact with the environment.

The setting. We model a semi-autonomous robot that operates in an *arena* D_L where each $l \in D_L$ is a possible location of the robot. For example, a location can be an (x, y) vector, providing the position of the robot in a plane of $1,000 \times 1,000$ squares.

We model the *environment* of the robot as a function that gives attributes to each location in the arena. Formally, the environment is $\mathcal{E} : D_L \to \langle D_{E_1}, D_{E_2}, \dots, D_{E_m} \rangle$ where each D_{E_i} is a finite domain of some property. For example, the environment may define the terrain of each location and whether it lies in the sun or in the shade, in which case $\mathcal{E}(3,5) = \langle$"smooth terrain", "sun"$\rangle$ means that the location $(3,5)$ is a sunny place with a smooth terrain. Note that, in this case study, the environment is time invariant.

The actions of the robot are governed by its *control program*. In each time step, denoted by a 'tick', the control program computes *output actions* based on some *external inputs*, *sensor values*, and the values of the robot's *internal variables*.

The external input is given by *input variables* $\langle I_1, \dots, I_k \rangle$, each over a finite domain D_{I_i}, and it comes from an external independent agent. The sensor values, given by *sensor variables* $\langle S_1, \dots, S_r \rangle$, over the finite domains $D_{S_1}, \dots, D_{S_r}$, are computed automatically based on the environment of the robot and its current location. Formally, for each sensor variable S_i there is a function $\xi_i : \mathcal{E} \times D_L \to D_{S_i}$. The robot also has some *internal variables*, $\langle N_1, \dots, N_g \rangle$, over the finite domains $D_{N_1}, \dots, D_{N_g}$, used for putting a logic in its behavior. The output actions are given by *output variables* $\langle A_1, \dots, A_l \rangle$, over the finite domains $D_{A_1}, \dots, D_{A_l}$. Upon performing the actions, the current location, given in the variable L, is automatically computed based on the previous location and the actions; formally, by a function $\eta : D_L \times D_{A_1} \times \dots \times D_{A_l} \to D_L$.

The *state* of the robot, $\mathcal{V}$, encapsulates the values of all the above variables. There is a *cost function* Energy which gives the energy gain (positive) or consumption (negative) of actions in the given environment, i.e., Energy is of type $D_{E_1} \times \dots \times D_{E_m} \times D_{A_1} \times \dots \times D_{A_l} \to \mathbb{Z}$. For the functions η, ξ_i, and Energy, we use the short-hand of applying the function to the whole state instead of the relevant variables. For example, instead of writing "$\xi_i(l) = v$ and value of L in state σ is l", we write "$\xi_i(\sigma) = v$".

The controller language. The language of the robot-control program is defined by the syntax shown in Figure 5. Most of the constructs in this language are standard, and will not be explained in detail. Note that the program cannot directly write to the location variables and sensor variables, but can only write to the internal variables and action variables. The most interesting construct in the syntax is the `tick` statement. Intuitively, the `tick` statement performs the actions described by the output variables (i.e., changes the location using the η function) and reads new values into the sensor variables (based on the environment and the current state, using the ξ_i functions) and into the input variables (nondeterministically). The formal semantics of the `tick` statement is described in the next paragraph.

We provide in Example 17 a simple setting of an environment, a control program, and the finite domains of the various variables.

Example 17.

The environment (arena).

$_x\diagdown^y$	*1*	*2*	*3*	*4*
1	-	▨ -	×	-
2	×	▨ /	×	-
3	▨ -	/	▨ /	▨ /
4	▨ -	▨ ×	/	▨ /

Legends.
□ : *Sun* ; ▨ : *Shade*
- : *hard* ; / : *soft* ; × : *obstacle*

The robot variables.

Location.	$D_L = \{(1,1),(1,2),\dots,(4,4)\}$
Inputs.	$D_{I_1} = \{Move, None\}$
	$D_{I_2} = \{Front, Back, Left, Right\}$
Sensors.	$D_{S_1} = \{InTheSun, InTheShade\}$
	$D_{S_2} = \{SunOnFront, NoSunOnFront\}$
	. . . Sensors for sun and obstacles all around
	$D_{S_9} = \{ObstacleOnRight, NoObstacleOnRight\}$
Actions.	$D_{A_1} = \{Move, None\}$
	$D_{A_2} = \{Front, Back, Left, Right\}$
Internal.	$D_{N_1} = \{InTheSun, InTheShade\}$
	$D_{N_2} = \{WasInTheSun, WasInTheShade\}$
	$D_{N_3} = \{Was^2InTheSun, Was^2InTheShade\}$

The cost function. *(The direction does not matter.)*
$\mathsf{Energy}(Sun, Hard/Soft, None) = +12$
$\mathsf{Energy}(Sun, Hard, Move) = +1$
$\mathsf{Energy}(Sun, Soft, Move) = -1$
$\mathsf{Energy}(Shade, Hard/Soft, None) = -5$
$\mathsf{Energy}(Shade, Hard, Move) = -12$
$\mathsf{Energy}(Shade, Soft, Move) = -15$

The robot-control program.
The program, intuitively, defines the following behavior.

- *Obey the external input, whenever it is legal. Otherwise, do nothing, if legal, or else check for a legal action.*
- *The constraints for a legal action:*
  - *Do not go into an obstacle. (A location out of the arena is considered as an obstacle.)*
  - *Do not stay in the sun for more than two consecutive steps.*

- *Whenever staying for two consecutive steps in the sun, avoid the sun for at least two consecutive steps.*

The code is straightforward; we give below some of its fragments.

```
while(1) {
// Check if the input is legal
  // Moving into an obstacle?
  if (I₁ = Move &&
       (   I₂ = Front && S₆ = ObstacleOnFront
        || I₂ = Back && S₇ = ObstacleOnBack
        || I₂ = Left && S₈ = ObstacleOnLeft
        || I₂ = Right && S₉ = ObstacleOnRight )
     )
      A₁ := None
  // Too much in the sun?
  if (N₂ = WasInTheSun
      && (N₁ = InTheSun || N₃ = Was²InTheSun) && ...
     )
     // Choose a legal action
     if (N₁ = InTheShadow)
        A₁ := None
     else if (S₂=NoSunOnFront && S₆=NoObstacleOnFront)
        A₁ := Move; A₂ := Front
      ...
  label1 : tick;
  N₃ := (N₂=WasInTheSun)? Was²InTheSun:Was²InTheShade
  N₂ := (N₁=InTheSun)? WasInTheSun : WasInTheShade
  N₁ := S₁
}
```

Semantics. Consider a robot-control program P, and fix diffusion constant k and a width constant c for a battery. We define the semantics of P in the standard small-step operational style. We summarize the state of the program as $(\sigma, \mathbf{t})$ where σ is a valuation of the variables, and $\mathbf{t}$ is a battery status. Therefore, the small-step semantics is given by a relation $\Rightarrow$ where intuitively, $(P, (\sigma, \mathbf{t})) \Rightarrow (P', (\sigma', \mathbf{t}'))$ holds if executing the first step from the program fragment P at state $(\sigma, \mathbf{t})$ leads to $(\sigma', \mathbf{t}')$ and the remaining program fragment is P'.

We assume that all the constructs except `tick` are executed instantaneously, and without any consumption of power; hence, the only construct that updates the battery status in the summary is the `tick`. Therefore, for all the other constructs, we do not explicitly present the semantics, but point out that the semantics are similar to a standard while-language. For the `tick` construct, we define the semantics using the proof rules from Figure 6.

Intuitively, on executing a `tick`, the effects of the output actions are performed, the sensor variables are updated based on the new location and environment, the next valuation of the input variables is given, and then the battery status is updated based on the cost of the actions in the current environment.

Problem statement. We consider model-checking problems; that is, asking whether a given model satisfies a given specification. The model, in our case, is a robot and its environment; namely, a robot-control program, battery constants, an initial battery status, an environment with locations D_L, and an initial location. The specification is a regular or ω-regular language over the (finite or infinite) sequences of locations in D_L. The model-checking problem is affirmatively answered if the robot has a path, in the given setting, such that the sequence of locations along the path belongs to the language of the specification.

Consider, for example, the setting of Example 17 together with an initial location $(1,1)$, a battery width constant $\frac{1}{2}$, a battery diffusion constant $\frac{1}{8}$, and an initial battery status $(16,16)$. A regular specification can ask, for instance, whether the robot has a finite path reaching the location $(3,2)$. An ω-regular specification can ask, say, whether the robot has an infinite path visiting the location $(1,4)$ infinitely often, while avoiding the locations $(3,1)$ and $(4,4)$.

Model-checking algorithm. Given a control-program P, an environment $\mathcal{E}$, a battery width constant c, a battery diffusion constant k, an initial battery status $\mathbf{t}_\iota$, and an initial variable valuation σ_ι, we define the equivalent battery transition system $\mathsf{BTS}[\![P, \mathcal{E}, c, k, \mathbf{t}_\iota, \sigma_\iota]\!] = \langle\langle S, \Delta, s_\iota, \nu\rangle, c, k\rangle$ as follows.

Let L be the set of labels of the `tick` statements in the program.

- A state in the BTS is a pair (l, σ) where $l \in L$ is a label, and σ is a valuation of all the variables in the program.
- There exists a transition from (l_1, σ_1) to (l_2, σ_2) on weight w if for some program fragments P_1 and P_2:
  - there exist battery statuses $\mathbf{t_1}$ and $\mathbf{t_2}$ and a proof that $((\mathtt{l_1} : \mathtt{tick}); P_1, (\mathbf{t_1}, \sigma_1)) \Rightarrow ((\mathtt{l_2} : \mathtt{tick}); P_2, (\mathbf{t_2}, \sigma_2))$ where the $Tick$ rule is applied exactly once; and
  - there exist battery statuses $\mathbf{t_1}$ and $\mathbf{t_2}$, such that there is a proof $(P, (\sigma_\iota, \mathbf{t_1})) \Rightarrow ((\mathtt{l_1} : \mathtt{tick}); P_1, (\sigma_1, \mathbf{t_2}))$.
- The cost of a transition from (l, σ) is w if applying the *cost* function on the valuation of the environment and action variables in σ is w.
- The initial state s_ι is given by (l, σ) such that there exists a program fragment P_1 and a battery status $\mathbf{t}$ such that there is a proof of $(P, (\sigma_\iota, \mathbf{t})) \Rightarrow (((\mathtt{l} : \mathtt{tick}); P_1), (\sigma, \mathbf{t}))$ containing no applications of the $Tick$ rule. Due to the determinism of our language, it is guaranteed that there exists only one such (l, σ).

A part of the BTS corresponding to Example 17 is given in Figure 7.

The battery diffusion constant $k = \frac{1}{8}$ and its width constant $c = \frac{1}{2}$.

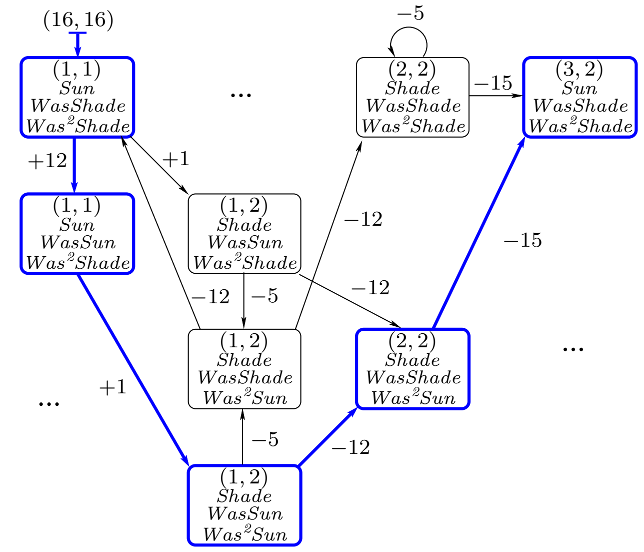
(16, 16)
(1, 1) Sun WasShade Was²Shade
(2, 2) Shade WasShade Was²Shade
(3, 2) Sun WasShade Was²Shade
(1, 1) Sun WasSun Was²Shade
(1, 2) Shade WasSun Was²Shade
(1, 2) Shade WasShade Was²Sun
(2, 2) Shade WasShade Was²Sun
(1, 2) Shade WasSun Was²Sun
−5
−15
+12
+1
−12
−12
−5
−12
−15
+1
−5
−12
...

Figure 7. A part of the BTS that corresponds to the robot and its environment, as described in Example 17. The best path to location $(3,2)$ appears in boldface blue. This path is infeasible by the BTS semantics, while feasible by models that are based on an ideal-energy resource.

We have the following theorem.

Theorem 18. *Consider a robot model-checking problem consisting of a control-program P, an environment $\mathcal{E}$ with locations D_L, a battery width constant c, a battery diffusion constant k, an initial battery status $\mathbf{t}_\iota$, an initial variable valuation σ_ι, and a regular or ω-regular language ϕ over the sequences of locations in D_L.*

$$\frac{}{(\texttt{label}:\texttt{tick},(\sigma,\mathbf{t}))\Rightarrow(\texttt{effects};\texttt{sensors};\texttt{inputs};\texttt{battery},(\sigma,\mathbf{t}))}\ \text{Tick}$$

$$\frac{cost(\sigma)=w \quad \texttt{Post}(\mathbf{t},w)=\mathbf{t}'}{(\texttt{battery},(\sigma,\mathbf{t}))\Rightarrow(\texttt{skip},(\sigma,\mathbf{t}'))}\ \text{Battery} \qquad \frac{v_1\in D_{I_1} \quad \dots \quad v_l\in D_{I_l}}{(\texttt{inputs},(\sigma,\mathbf{t}))\Rightarrow(\texttt{skip},(\sigma[\forall k: I_k:=v_k],\mathbf{t}'))}\ \text{Inputs}$$

$$\frac{v_1=\xi_1(\sigma) \quad \dots \quad v_r=\xi_r(\sigma)}{(\texttt{sensors},(\sigma,\mathbf{t}))\Rightarrow(\texttt{skip},(\sigma[\forall k: S_k:=v_k],\mathbf{t}'))}\ \text{Sensors} \qquad \frac{v=\eta(\sigma)}{(\texttt{effects},(\sigma,\mathbf{t}))\Rightarrow(\texttt{skip},(\sigma[\forall k: L:=v],\mathbf{t}'))}\ \text{Effects}$$

Figure 6. Semantics of `tick`

Let $\mathcal{B} = \mathsf{BTS}[\![P,\mathcal{E},c,k,\mathbf{t}_\iota,\sigma_\iota]\!]$. For a control state $b\in\mathcal{B}$, let $\mathsf{Location}(b)$ be the valuation of the robot location variable in b. Let ϕ' be a regular or ω-regular language over sequences of control locations in $\mathcal{B}$, such that a sequence $b_0, b_1, \ldots \in \Phi'$ iff $\mathsf{Location}(b_0), \mathsf{Location}(b_1), \ldots \in \Phi$.

Then, the robot model-checking problem is equivalent to the BTS model-checking of $\mathcal{B}$ and ϕ'.

Battery vs. ideal energy. Model-checking the robot behavior, taking into account the non-ideal aspects of the energy resource, is inherently different from considering the battery as an ideal energy resource, as demonstrated in Example 17. There, the robot cannot go with an initial battery status of $(16, 16)$ from location $(1, 1)$ to $(3, 2)$ (cf., Theorem 3.) On the other hand, it is possible to go from location $(4, 4)$ to $(3, 4)$, starting with the same initial battery status. Note that such a situation is impossible with a model that is based on an ideal-energy resource, as the energy loss going from location $(4, 4)$ to $(3, 4)$ is 15, while from location $(1, 1)$ to $(3, 2)$ it is only 14! (The reason, as elaborated on in Sections 2–3, lies in the influence of the energy changes along the path on the available charge of the battery.)

7. Conclusions and Future Work

We presented the first discrete formal model of battery systems and showed that the standard automaton emptiness problems for this model are decidable. Further, these battery transition systems do not fall into the large class of well-structured transition systems.

In terms of future work, a natural direction is to explore standard program analysis and program synthesis questions for systems that use batteries. For example, to begin with, one could define an extension to standard imperative languages to allow programs to branch based on the status of the battery. For programs written in such a language, one could attempt to compute invariants about the combined program- and battery-state through abstract interpretation. Also, one could attempt battery-aware partial-program synthesis for such a language. This would be a generalization of the battery-aware scheduling problem studied in [18]. Another direction to explore is the possibility of solving two-player games for battery transition system, leading to battery-aware algorithms for synthesis of reactive systems.

References

[1] R. Adany and T. Tamir. Online algorithm for battery utilization in electric vehicles. In *FedCSIS*, pages 349–356, 2012.

[2] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425, 1990.

[3] S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA*, volume 3707 of *LNCS*, pages 474–488, 2005.

[4] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. Extending lifetime of portable systems by battery scheduling. In *DATE*, pages 197–203, 2001.

[5] G. G. Botte, V. R. Subramanian, and R. White. Mathematical modeling of secondary lithium batteries. *Electrochimica Acta*, 45(15):2595–2609, 2000.

[6] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT*, volume 2855 of *LNCS*, pages 117–133, 2003.

[7] K. Chatterjee and L. Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012.

[8] K. Chatterjee, L. Doyen, T. A. Henzinger, and J. F. Raskin. Generalized mean-payoff and energy games. In *FSTTCS*, volume 8 of *LIPIcs*, pages 505–516, 2010.

[9] C. Chiasserini and R. Rao. Energy efficient battery management. *J. on Selected Areas in Communications*, 19(7):1235–1245, 2001.

[10] M. Doyle, T. F. Fuller, and J. Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *J. of the Electrochemical Society*, 140(6):1993, 1526–1533.

[11] A. Finkel and P. Schnoebelen. Parallel program schemata. *J. of Computer and System Sciences*, 3(2):147–195, 1969.

[12] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1–2):63–92, 2001.

[13] T. F. Fuller, M. Doyle, and J. Newman. Relaxation phenomena in lithiumion-insertion cells. *J. of the Electrochemical Society*, 141(4): 982–990, 1994.

[14] S. Gold. A PSPICE macromodel for lithium-ion batteries. In *Annual Battery Conference on Applications and Advances*, pages 215–222, 1997.

[15] S. C. Hageman. Simple PSPICE models let you simulate common battery types. *Electronic Design News*, 38(22):117–129, 1993.

[16] M. R. Jongerden. *Model-based energy analysis of battery powered systems*. PhD thesis, University of Twente, 2010.

[17] M. R. Jongerden and B. R. Haverkort. Which battery model to use? *IET Software*, 3(6):445–457, 2009.

[18] M. R. Jongerden, A. Mereacre, H. C. Bohnenkamp, B. R. Haverkort, and J.-P. Katoen. Computing optimal schedules for battery usage in embedded systems. *IEEE Trans. Industrial Informatics*, 6(3):276–286, 2010.

[19] K. G. Larsen, S. Laursen, and J. Srba. Action investment energy games. In *MEMICS*, volume 7721 of *LNCS*, pages 155–167, 2012.

[20] J. Manwell and J. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399–405, 1993.

[21] E. Podlaha and H. Cheh. Modeling of cylindrical alkaline cells. *J. of the Electrochemical Society*, 14(1):15–27, 1994.

[22] D. Rakhmatov and S. Vrudhula. An analytical high-level battery model for use in energy management of portable electronic systems. In *International Conference on Computer Aided Design (ICCAD)*, pages 488–493, 2001.

[23] S. Safra and M. Vardi. On ω-automata and temporal logic. In *STOC*, pages 127–137, 1989.

[24] M. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266, 1996.