Infinite Runs in Recharge Automata

Daniel Ejsing-Duun, Lisa Fontani

Department of Computer Science, Aalborg University, Denmark

Summary

In this article we introduce recharge automata, a variant of priced timed automata with only one resource variable. In this formalism, the resource level can be decreased at a given rate while delaying in locations and instantaneously increased to its maximum when taking discrete transitions.

We focus on recharge automata with only one clock and want to find out whether for a given automaton there exists an infinite time-diverging run such that the resource never goes below 0. For this purpose we present a normal form that divides the automaton into segments, in which it is possible to freely move between locations. We then abstract such automaton segments by making use of an adaptation of energy functions. These take as input the current energy when entering the automaton segment and compute the highest possible energy we can end up with when leaving the segment.

The adaptation of energy functions to this formalism results in functions that are non-decreasing and may have some points of discontinuity. We propose a representation for energy functions, describe maximum and composition operations and show how these operations can be computed in polynomial time.

We then use energy functions in the construction of an abstraction of recharge automata, called energy function automata, where each transition abstracts a segment of the recharge automaton. Energy change is represented on the transitions through the before-mentioned functions and time information is given by a boolean value.

By graph analysis we can find all reachable cycles of a certain size in the resulting automaton and find out whether one of them can be repeated infinitely while not using more than the available energy. This results in an NP-algorithm. We further show that the problem can be solved in polynomial time if we restrict to so-called flat recharge automata, where each location is only part of one cycle.

Infinite Runs in Recharge Automata

Daniel Ejsing-Duun, Lisa Fontani

Department of Computer Science, Aalborg University, Denmark

Abstract. We consider recharge automata, a variant of weighted/priced timed automata with a single, bounded cost variable that can be decreased when delaying in locations and fully recharged when taking discrete transitions. Given such an automaton with just one clock, we investigate whether there exists an infinite time-divergent run where the resource always stays above zero. Our method includes a notion of energy functions for abstracting runs by only considering the initial and final energy. By means of these, we provide a polynomial time algorithm for solving the problem for 'flat' recharge automata and prove that the general problem for any automaton is in NP.

1 Introduction

Pollution and the greenhouse effect have made energy consumption a hot topic in climate discussions in the last years [1,11,2]. Several examinations of the state of the environment over the last decades have made it clear that we, among other things, need to put great attention to how much energy our electronic equipment consumes. This has lead to research into how to reduce energy consumption in most areas, from big infrastructures to domestic appliances [17,20,18,19,13].

In the area of software verification, priced timed automata [6,5] have been proposed as a new formalism based on timed automata [4]. They are used for modelling real-time systems with clock variables monitoring time and cost variables representing resource consumption and production. Their use has made it possible to reason about and verify properties regarding resource consumption of software systems before creating them and as such make guarantees regarding for instance their energy efficiency. They can be used to optimize or compare systems fulfilling the same or similar purposes and choose between them based on how much energy they consume in the process. Examples of such requirements could be how long a given system can run based on given start conditions or what these conditions have to be to ensure the existence of some desired runs.

In the present article we introduce a variant of this formalism, called *recharge automata*. These are priced timed automata with a resource which is consumed as time passes and can be instantly recharged on transitions when performing specific actions. The intuition is to simulate the energy used by for instance electric vehicles which originates from a battery. This has a given capacity and can at any time be replaced with one that is fully charged. All actions and states consume energy at a known rate and it is not possible to obtain an energy level below zero.

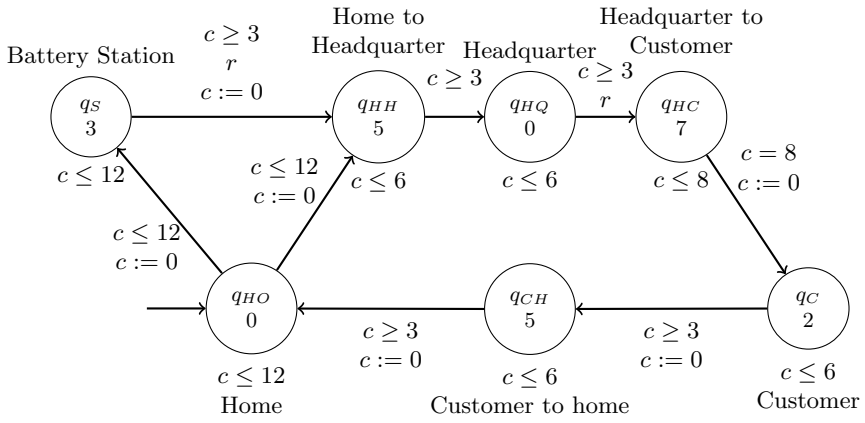


Fig. 1: Automaton modelling a business man travelling between his home, his customer and the headquarter in his electric car.

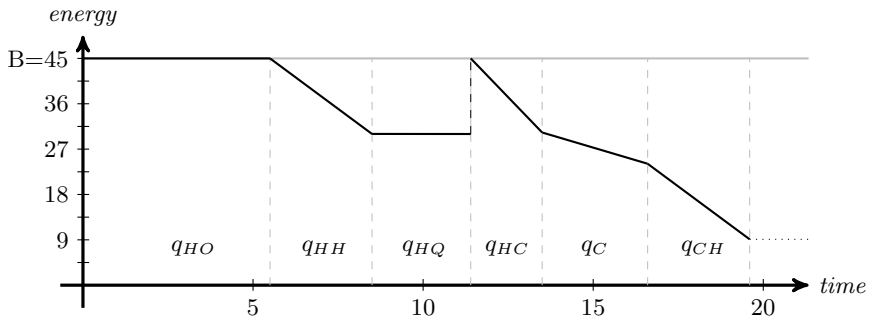


Fig. 2: Energy consumption over time during a run of the business man automaton.

We will be concerned with the problem of finding infinite time-diverging runs in recharge automata with one clock. This can for instance be relevant for manufacturers of electric cars with requirements regarding energy efficiency and an option of immediate resource refill. Examples could be the mobile phone industry when deciding on battery capacity of devices, such that they can be sure to have energy for a day or more. Other areas include the issue of determining the placement of battery replacement station for electric cars, such that they do not risk running out of energy when going for long distances.

As an example consider the recharge automaton in Fig. 1. It models a business man that each day has to travel to the headquarter of his firm (q_{HQ}). When he arrives, he gets an assignment regarding one of his customers, which he visits afterwards (q_C). After having completed his work, he finally can head back home (q_{HO}). The business man always travels by his electric car, and has the possibility of changing the car's battery both on the way to work and at the

firm's headquarters. When he decides to change the battery on the way to work he needs to travel to a nearby battery station (q_S), which takes him at least 3 time units. The different rates on the locations depend on the type of road taken and the speed limit. A rate of zero represents that the car is parked and thereby is not using any energy. Fig. 2 represents the energy level of the battery of the car while the business man moves between the different locations, assuming that the battery has a capacity of 45. The problem is now to find out whether the business man can continue this travelling pattern without ever requiring more energy than available.

We present a polynomial time algorithm solving the problem for flat recharge automata, a subclass where each location can be part of at most one cycle and requiring guards on transitions to be of a specific form. Moreover, an NP algorithm is presented for the general case. The solution is based on an adaptation of energy functions, which were originally presented in [8].

The article is structured as follows. In Section 2 we go through other work related to recharge automata. Section 3 and 4 define recharge automata as well as the problem considered. Section 5 defines energy functions, some operations on them and how to represent them. Section 6 then describes how these can be used to abstract part of a recharge automaton. This leads to the abstraction of a whole automaton, in Section 7, where energy function automata and their construction are presented. Finally, Section 8 explains how this automaton can provide us with a solution to the problem and Section 9 rounds off with a conclusion and future work.

2 Related Work

Automata with resources and instantaneous resets have been investigated before by Wang et. al. [3]. In their work, they define R-automata as finite state machines with an arbitrary amount of counters. The value of each counter can be increased, reset or left unchanged at every transition. The authors look into the universality problem in regards to determining an upper bound that permits all executions and prove that it can be done in 2-EXPSPACE. Within our formalism the corresponding problem would be to decide on an upper bound value such that all runs can be performed without ever running out of energy.

The concept of bounding the resources has been studied in-depth for weighted timed automata. When only focusing on a resource that is not allowed to have a value less than zero [10], finding infinite runs has been proved undecidable for models with four clocks, while a time-limited run makes the problem decidable in NEXPTIME. Another version of the problem where an initial energy level is to be found, such that an infinite or time-limited run can be performed, is PSPACE-complete [10].

Furthermore, different kinds of optimization problems for weighted timed automata have been considered. For example, reaching a goal location while incurring the least cost has been proven decidable [6,5], and an implementation based on the extension of zones with prices was presented in [16]. In models with

both costs and rewards an optimal strategy could imply the cost per reward to be minimized [7]. For models where the present costs are more relevant than future costs, or where there is a growing probability that the object modelled will stop running (e.g. component failure), optimal infinite runs can be calculated using discount factors [14]. The last two problems are proven to be PSPACE-complete [7,14].

When considering a constrained resource, the main problems include determining whether either one or all runs of a given automaton can be performed while keeping the resource level between a lower and either a strict or weak upper bound, meaning that energy increasing behaviour after reaching the bound only maintains the current level [9]. For one clock and a lower and weak upper bound the problem is decidable in polynomial time. Though the infinite run problem for recharge automata could be seen as a similar problem, their result cannot be used here, since recharge automata have discrete updates on transitions, meaning that optimal delays can require spending time in more than one location.

An interesting approach to path abstraction has appeared in [8], where energy functions are used for one-clock priced timed automata with both positive and negative weights on locations and transitions. Their purpose is to create a mapping between an initial resource value and the maximal resource value that can be obtained along a given path. Using them, it has been shown that the reachability and the infinite run problem can be solved in EXPTIME. We will also make use of this technique, though with a few differences regarding the properties of the functions and how to construct them. Note that our approach cannot be used for this formalism since recharge automata do not have arbitrarily weighted transitions and the locations can only have negative rates.

3 Preliminaries

A recharge automaton is a timed automaton extended with a continuous variable that models the consumption of some resource along an execution of the system under consideration. The system execution starts with a fully charged resource which can be used when delaying in locations by specifying a rate. The model may include a recharge along some transitions that brings the level of the resource back to its maximum.

In our definition of a recharge automaton we focus on energy as the resource of interest. We define the set C to be the set of clocks and $\Phi(C)$ to be the set of clock constraints ϕ defined by the grammar $\phi ::= c \bowtie k \mid \phi_1 \wedge \phi_2$, where $c \in C$, $k \in \mathbb{Z}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Moreover, let $\Psi(C)$ be the set of clock constraints ψ defined by the new grammar $\psi ::= c \preceq k \mid \psi_1 \wedge \psi_2$, where $c \in C$, $k \in \mathbb{Z}$ and $\preceq \in \{<, \leq\}$.

Definition 1 (Recharge Automaton). *A recharge automaton is a tuple* $\mathcal{A} = (Q, q_0, B, C, I, \mathit{rate}, \Delta)$*, where*

- Q *is a finite set of locations,*
- $q_0 \in Q$ *is the initial location,*
- $B \in \mathbb{N}_0$ *is a maximum energy level,*
- C *is a finite set of clocks,*
- $I : Q \to \Psi(C)$ *is a function assigning an invariant to each location,*
- $\mathit{rate} : Q \to \mathbb{N}_0$ *is a rate function and*
- $\Delta \subseteq Q \times \Phi(C) \times \{\epsilon, r\} \times 2^C \times Q$ *is a transition relation.*

The transition relation allows to move from one location to another when the clocks satisfy the guard of the transition and the invariants of the locations involved. Also, during any transition the energy may be recharged and any number of clocks reset. A transition (q, g, z, R, q') is called reset transition if $R \neq \emptyset$ and it is called recharge transition if $z = r$.

Example 1. Assuming a resource bound of 45, the recharge automaton in Fig. 1 can be formally described as $\mathcal{A} = (Q, q_{HO}, 45, \{c\}, I, \mathit{rate}, \Delta)$, where:

$Q = \{q_{HO}, q_S, q_{HH}, q_{HQ}, q_{HC}, q_C, q_{CH}\}$

$I(q_{HO}) = c \leq 12$ $\quad I(q_S) = c \leq 12$ $\quad I(q_{HH}) = c \leq 6$

$I(q_{HQ}) = c \leq 6$ $\quad I(q_{HC}) = c \leq 8$ $\quad I(q_C) = c \leq 6$

$I(q_{CH}) = c \leq 6$

$\mathit{rate}(q_{HO}) = 0$ $\quad \mathit{rate}(q_S) = 3$ $\quad \mathit{rate}(q_{HH}) = 5$

$\mathit{rate}(q_{HQ}) = 0$ $\quad \mathit{rate}(q_{HC}) = 7$ $\quad \mathit{rate}(q_C) = 2$

$\mathit{rate}(q_{CH}) = 5$

$$\begin{aligned}\Delta = \{&(q_{HO}, c \leq 12, \epsilon, \{c\}, q_S), (q_S, c \geq 3, r, \{c\}, q_{HH}),\\ &(q_{HO}, c \leq 12, \epsilon, \{c\}, q_{HH}), (q_{HH}, c \geq 3, \epsilon, \emptyset, q_{HQ}),\\ &(q_{HQ}, c \geq 3, r, \emptyset, q_{HC}), (q_{HC}, c = 8, \epsilon, \{c\}, q_C),\\ &(q_C, c \geq 3, \epsilon, \{c\}, q_{CH}), (q_{CH}, c \geq 3, \epsilon, \{c\}, q_{HO})\}\end{aligned}$$

Let $\mathcal{A} = (Q, q_0, B, C, I, \mathit{rate}, \Delta)$ be a recharge automaton. We define a *path* π in $\mathcal{A}$ to be an alternating sequence of locations and transitions

$$q_1 \xrightarrow{g_1, z_1, R_1} q_2 \xrightarrow{g_2, z_2, R_2} \dots \xrightarrow{g_{n-1}, z_{n-1}, R_{n-1}} q_n,$$

such that for all $i,\ 1 \leq i < n$, $(q_i, g_i, z_i, R_i, q_{i+1}) \in \Delta$.

We will further on consider a subclass of recharge automata called one-clock closed recharge automata.

Definition 2 (One-Clock Closed Recharge Automaton). *A one-clock closed recharge automaton* $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$ *is a recharge automaton where* $C = \{c\}$ *and where all guards and invariants are non-strict.*

See Fig. 1 for an example of such an automaton.

We define the semantics of a recharge automaton $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$ by a labelled transition system $[\![\mathcal{A}]\!] = (S, s_0, \to)$. A state $s \in S$ is a tuple (q, v, e) where $q \in Q$ is a location, $v : C \to \mathbb{R}_{\geq 0}$ is a valuation over the clocks and $e \in [0, B]$ is the remaining energy. The initial state is $s_0 = (q_0, v_0, B)$, where v_0 is the valuation mapping each clock to 0.

We write $v' = v[R]$ to denote the valuation v' where all clocks in the set R have been reset to 0 and all other clocks have the same value as in v. Finally, we denote by $v + d, d \in \mathbb{R}_{\geq 0}$, the valuation where the value of each clock has been incremented by d.

Transitions between states in the labelled transition system $[\![\mathcal{A}]\!]$ are of two kinds:

- *Delay transitions*: $(q, v, e) \xrightarrow{d} (q, v + d, e')$, where $d \in \mathbb{R}_{\geq 0}$ and it holds that $v + d \models I(q)$ and $e' = e - rate(q) \cdot d,\ e' \geq 0$.
- *Discrete transitions*: $(q, v, e) \xrightarrow{t} (q', v', e')$, where $t = (g, z, R)$ and $(q, g, z, R, q') \in \Delta$, $v \models g$, $v' = v[R]$ and $v' \models I(q')$. Furthermore, if $z = r$ then $e' = B$, else $e' = e$.

An *infinite run* Γ of the recharge automaton $\mathcal{A}$ is an infinite sequence on the form $s_1 \xrightarrow{d_1} s'_1 \xrightarrow{t_1} s_2 \xrightarrow{d_2} s'_2 \xrightarrow{t_2} s_3 \xrightarrow{d_3} \ldots$, where for all $i,\ i \geq 1,\ s_i \in S$.

We will also use the notion of a *run segment* γ to denote a finite part of an infinite run, $s_i \xrightarrow{d_i} s'_i \xrightarrow{t_i} \ldots \xrightarrow{t_{j-1}} s_j \xrightarrow{d_j} s'_j$. Similarly, a *simple* run segment is a run segment in which no location is visited more than once.

Given an infinite run Γ, let $time(\Gamma) = \sum_{i=0}^{\infty} d_i$ denote the time elapsed during the run.

Example 2. The following shows an infinite run of the automaton in Fig. 1, assuming that the bound is 45:

$$
\begin{aligned}
\Gamma = \ & (q_{HO}, [c=0], 45) \xrightarrow{5.5} (q_{HO}, [c=5.5], 45) \xrightarrow{c\leq 12, \epsilon, \{c\}} (q_{HH}, [c=0], 45) \\
& \xrightarrow{3} (q_{HH}, [c=3], 30) \xrightarrow{c\geq 3, \epsilon, \emptyset} (q_{HQ}, [c=3], 30) \\
& \xrightarrow{2.9} (q_{HQ}, [c=5.9], 30) \xrightarrow{c\geq 3, r, \emptyset} (q_{HC}, [c=5.9], 45) \\
& \xrightarrow{2.1} (q_{HC}, [c=8], 30.3) \xrightarrow{c=8, \epsilon, \{c\}} (q_{C}, [c=0], 30.3) \\
& \xrightarrow{3.1} (q_{C}, [c=3.1], 24.1) \xrightarrow{c\geq 3, \epsilon, \{c\}} (q_{CH}, [c=0], 24.1) \\
& \xrightarrow{3} (q_{CH}, [c=3], 9.1) \xrightarrow{c\geq 3, \epsilon, \{c\}} (q_{HO}, [c=0], 9.1) \\
& \xrightarrow{3} (q_{HO}, [c=3], 9.1) \xrightarrow{c\leq 12, \epsilon, \{c\}} (q_{S}, [c=0], 9.1) \\
& \xrightarrow{3} (q_{S}, [c=3], 0.1) \xrightarrow{c\geq 3, r, \{c\}} (q_{HH}, [c=0], 45) \xrightarrow{3} \dots
\end{aligned}
$$

Note that $time(\Gamma) = \infty$.

4 Infinite Run Problem

We will now define the problem to be solved:

Problem 1. Given a recharge automaton $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$, does there exist an infinite run Γ of $\mathcal{A}$ starting in (q_0, v_0, B) such that $time(\Gamma) = \infty$?

We will call this Γ a *winning* run. Similarly, we will define a winning state.

Definition 3 (Winning State). *Let $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$ be a recharge automaton. A state (q, v, e) is said to be winning if there exists an infinite run Γ of $\mathcal{A}$ starting from (q, v, e) such that $time(\Gamma) = \infty$.*

In the following, we will see how a winning state will also be winning if we increase the available energy of the state, meaning that any energy level equal to or above the energy level of the state will make it winning.

Lemma 1. *Let $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$ be a recharge automaton. If (q, v, e) is a winning state of $\mathcal{A}$ then every state (q, v, e') such that $e' \geq e$ is winning as well.*

A proof of this lemma can be found in our previous report on the subject [12].

Example 3. The run Γ from Example 2 is winning. Moreover, any state s in Γ is also winning, since we can just remove the run segment from the start state to s to obtain an infinite run starting from s. Additionally, consider the state $(q_{HQ}, [c = 3], 37)$. By Lemma 1 this is a winning state, since the state $(q_{HQ}, [c = 3], 30)$ of Γ is winning.

We will now focus on one-clock closed recharge automata. Given a clock constraint ϕ, let $ub(\phi)$ be the highest clock value that satisfies ϕ. Moreover let *true* denote the guard $c \geq 0$. Then we can define the following normal form for a recharge automaton.

Definition 4 (Recharge-Free Cycle Normal Form). *A one-clock closed recharge automaton $\mathcal{A} = (Q, q_0, B, \{c\}, I, rate, \Delta)$ is in recharge-free cycle normal form (rfcNF), if*

- *there does not exist any path $q_1 \xrightarrow{g_1,z_1,R_1} q_2 \xrightarrow{g_2,z_2,R_2} \dots \xrightarrow{g_n,z_n,R_n} q_1$ where $q_i \neq q_j$ whenever $i \neq j$, and for all $i,\ 1 \leq i \leq n,\ R_i = \emptyset$ and*
- *for all transitions $(q, g, z, R, q') \in \Delta$, we have that either*
  - *$R = \emptyset,\ I(q) = I(q')$ and $g = true$ or*
  - *$R = \{c\}$ and either $g =$ "$c = ub(I(q))$" or $g = true$.*

Since only transitions with a reset can require the clock to have a specific value, this normal form allows a run to freely move between locations that are connected by transitions without a clock reset. Moreover all cycles in the automaton have at least one clock reset, allowing us to order the transitions that can be taken on a path between two reset transitions.

These properties enable us to abstract all paths between two reset transitions, by means of functions that return the highest energy that can be achieved at the end of any of these paths. These functions will be introduced in the next section.

Theorem 1. *Given a one-clock closed recharge automaton $\mathcal{A}$ it is possible to construct in polynomial time a one-clock closed recharge automaton in rfcNF $\mathcal{A}'$ such that there exists a winning run in $\mathcal{A}'$ iff there exists a winning run in $\mathcal{A}$.*

Once again the proof can be found in our previous work [12].

Example 4. In Fig. 3 the path $q_{HH} \xrightarrow{c\geq 3,\epsilon,\emptyset} q_{HQ} \xrightarrow{c\geq 3,r,\emptyset} q_{HC} \xrightarrow{c=8,\epsilon,\{c\}} q_C$ of the recharge automaton in Fig. 1 is shown in rfcNF. Since the automaton grows quite a lot by this conversion, we have chosen only to show this part of the automaton.

In the rest of the article, we will only consider one-clock closed recharge automata in rfcNF.

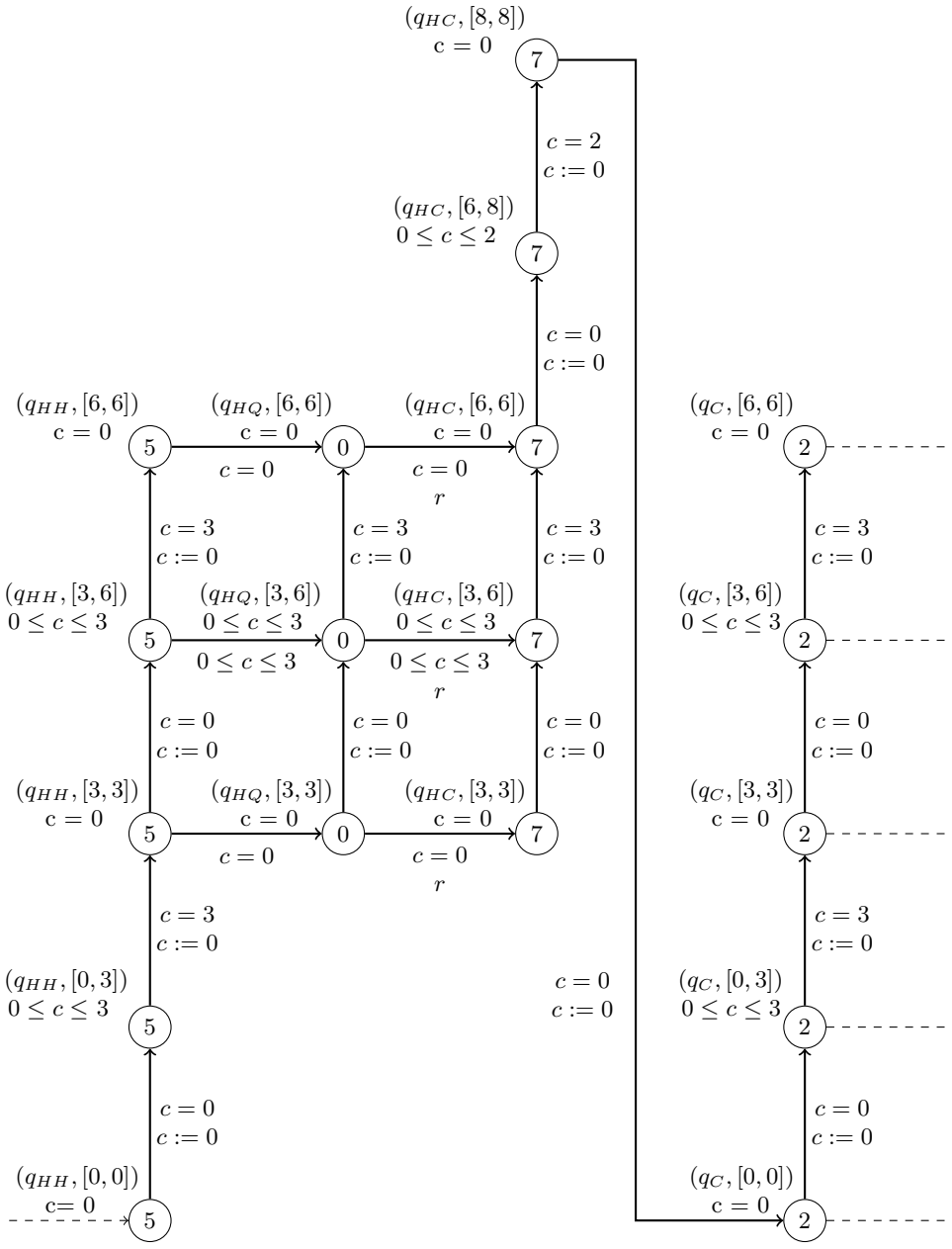


Fig. 3: Part of the automaton from Fig. 1 in rfcNF.

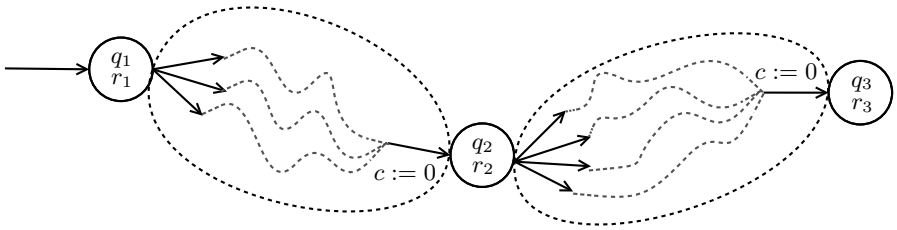


Fig. 4: Paths abstracted by energy functions.

5 Energy Functions

In this section we will define energy functions. Their purpose is to abstract a set of paths between two locations with a clock reset only on the last transition, as shown in Fig. 4. Given the energy level at the source location, the functions return the highest energy level that can be obtained at the target location by any run along one of the abstracted paths.

Recall that all locations along the paths, except the last one, have the same invariant and all transitions, except the last one, have true guards. By choosing as source a location with an incoming reset transition, we then know that all paths will have to delay the same amount of time as the value of the clock will be zero at the start of the run and will have to reach the value required to satisfy the guard on the last transition.

Because of the equal invariants and true guards, it is possible to distribute the delay among any of the locations on the paths. In this way we can then abstract all paths in an automaton by making an energy function for each pair of locations with an incoming reset transition. The illustration in Fig. 4 shows which paths will be abstracted by energy functions.

An example of an energy function can be seen on Fig. 5. As shown, the domain of the functions is [0,B], while the codomain includes also the element $\perp$ (dashed line in the graph), meaning that the given energy is not enough to reach the target location along any path. The function itself is non-decreasing

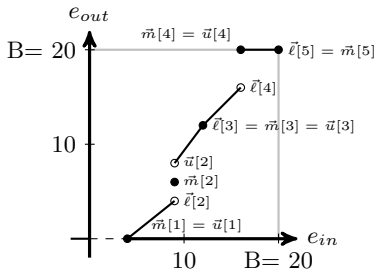


Fig. 5: An energy function.

and can be discontinuous in some points, while in the intervals between these points the function is continuous and linear.

To represent energy functions, we can therefore divide the domain of the function into intervals in which the function is linear. Observe that each interval and point can be seen as a coarse region. Following this idea, we can observe each interval as a region and each endpoint of an interval as a region of only one value.

Because of this partitioning, each endpoint of an interval will delimit three regions (or two in case of the first and last interval). We will therefore have three possibly different function values for each endpoint: the lowest value delimiting the region before the point, the middle value for the region at the point, and the highest for the region after the point.

We use this in our representation of the energy functions by having one vector $\vec{x}$ for the domain values and three vectors $\vec{\ell}$, $\vec{m}$ and $\vec{u}$ representing the corresponding lower, middle and upper codomain values for a given domain value.

Definition 5 (Energy Function - Representation and Semantics). *Given a bound $B \in \mathbb{N}_0$, an energy function is represented by a tuple $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ where*

- *$\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ is an increasing finite sequence of numbers such that for all $i, 1 \leq i \leq n-1$, $x_i \in \mathbb{Q}, 0 \leq x_i \leq B$ and $x_n = B$,*
- *$\vec{m} = \langle m_1, m_2, \ldots, m_n \rangle$, $\vec{u} = \langle u_1, u_2, \ldots, u_{n-1} \rangle$, $\vec{\ell} = \langle \ell_2, \ell_3, \ldots, \ell_n \rangle$ are finite sequences of numbers such that for all $i, 1 \leq i \leq n-1$, $m_i, u_i, \ell_{i+1}, m_n \in \mathbb{Q}, 0 \leq m_i, u_i, \ell_i, m_n \leq B$, $m_i \leq u_i \leq \ell_{i+1} \leq m_{i+1}$.*

The representation $\mathcal{R}$ defines an energy function $f_{\mathcal{R}} : [0, B] \to [0, B] \cup \{\bot\}$ such that:

$$f_{\mathcal{R}}(x) = \begin{cases} \bot & \text{if } x < x_1 \\ m_i & \text{if } x = x_i,\ 1 \leq i \leq n \\ u_i + (\ell_{i+1} - u_i)\frac{x - x_i}{x_{i+1} - x_i} & \text{if } x_i < x < x_{i+1},\ 1 \leq i \leq n-1 \end{cases}$$

Note that $\vec{\ell}$ starts with ℓ_2, since there is no region before x_1 and as such the first region including more than one value ends in x_2.

We use $\mathcal{F}_{\mathcal{R}}$ to denote the set of all representations of energy functions and $\mathcal{F}$ to denote the set of all energy functions.

Example 5. Given $B = 20$, we can construct a representation $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ for the energy function in Fig. 5, where:

$$\vec{x} = \langle 4, 9, 12, 16, 20 \rangle \qquad \vec{m} = \langle 0, 6, 12, 20, 20 \rangle$$
$$\vec{u} = \langle 0, 8, 12, 20 \rangle \qquad \vec{\ell} = \langle 4, 12, 16, 20 \rangle$$

Since energy functions abstract a set of paths, to be able to solve the infinite run problem it will be necessary to choose the most profitable between different paths leading to the same location and to concatenate a path with another to make a longer path. Therefore, we will introduce maximum and composition operators and show that energy functions are closed under these operators.

5.1 The Maximum Operator

The operator $\mathcal{M}\boldsymbol{ax} : \mathcal{F} \times \mathcal{F} \to \mathcal{F}$ takes two energy functions with the domain $[0, B]$, for some $B \in \mathbb{N}_0$, and returns the pointwise maximum of them. Assume that $\bot < n$ for all $n \in \mathcal{R}$. Then $\mathcal{M}\boldsymbol{ax}$ is defined as follows:

$$\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = \max\{f_{\mathcal{R}_1}(x), f_{\mathcal{R}_2}(x)\} \qquad \text{for all } x \in [0, B]$$

Construction

Given two representations of energy functions, $\mathcal{R}_1 = (\vec{x}_1, \vec{m}_1, \vec{u}_1, \vec{\ell}_1)$ and $\mathcal{R}_2 = (\vec{x}_2, \vec{m}_2, \vec{u}_2, \vec{\ell}_2)$, it is possible to construct a representation $\mathcal{R}$ s.t. $f_{\mathcal{R}} = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})$.

First, let $elem(\vec{a})$ denote the set containing all elements of a vector $\vec{a}$.

$$elem(\vec{a}) = \{\vec{a}[i] \mid 1 \leq i \leq |\vec{a}|\}$$

When taking the pointwise maximum, note that when one function is giving rise to maximum values, the other function can only take over either when we move to a new region with a new linear function (for example point $\vec{m}[3]$ in Fig. 6(e)), or within the same region if the two functions have only one point of intersection in the region (for example point $\vec{m}[2]$ in Fig. 6(c)). Let S_{inter} be the set of these points of intersection and let S_{max} be the set containing all relevant points:

$$S_{inter} = \Big\{ x \mid f_{\mathcal{R}_1}(x) = f_{\mathcal{R}_2}(x) \text{ and } \exists \delta > 0 \text{ such that } \forall \epsilon,\ \delta \geq \epsilon > 0,$$
$$f_{\mathcal{R}_1}(x+\epsilon) \neq f_{\mathcal{R}_2}(x+\epsilon) \text{ and } f_{\mathcal{R}_1}(x-\epsilon) \neq f_{\mathcal{R}_2}(x-\epsilon) \Big\}$$
$$S_{max} = S_{inter} \cup elem(\vec{x}_1) \cup elem(\vec{x}_2)$$

Then we can construct $\mathcal{R}$ by calling the function *Maximum* shown in Algorithm 1.

The idea behind the construction is always to compare the points of the two function representations and overtaking them directly if we are at a domain value defined in $\vec{x}_1$ or $\vec{x}_2$. If not, the value is found by interpolation. Note that we only look at the relevant vector indices for each vector. In the case of $\vec{\ell}$ we define the values $\vec{\ell}_1[1]$ and $\vec{\ell}_2[1]$ to be $\bot$, to avoid special cases when considering the first value in one of the vectors. This is not relevant in the same way for $\vec{u}$, since both functions end in the same point by definition, and thus the undefined vector values will never be accessed by the algorithm.

Algorithm 1: The algorithm calculates the energy function representation of the pointwise maximum between the functions represented by $\mathcal{R}_1$ and $\mathcal{R}_2$.

Maximum($\mathcal{R}_1$, $\mathcal{R}_2$)
Input - $\mathcal{R}_1 = (\vec{x}_1, \vec{m}_1, \vec{u}_1, \vec{\ell}_1)$, $\mathcal{R}_2 = (\vec{x}_2, \vec{m}_2, \vec{u}_2, \vec{\ell}_2)$
Output - $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ such that $f_{\mathcal{R}} = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})$

- Let $\vec{x}$ be the increasing sequence containing all elements from the set S_{max}.
- For all $i,\ 1 \leq i \leq |\vec{x}|,\ \vec{m}$ is defined as follows:

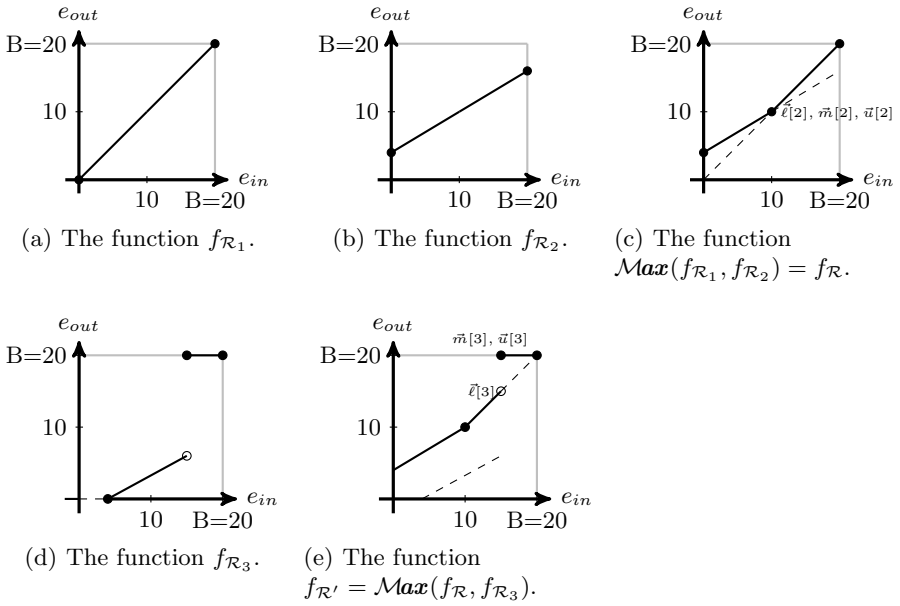
$$\vec{m}[i] = \max\{f_{\mathcal{R}_1}(\vec{x}[i]), f_{\mathcal{R}_2}(\vec{x}[i])\} \tag{1}$$

- For all $i,\ 1 \leq i \leq |\vec{x}| - 1,\ \vec{u}$ is defined as follows, where $1 \leq j \leq |\vec{x}_1| - 1$ and $1 \leq k \leq |\vec{x}_2| - 1$:

$$\vec{u}[i] = \begin{cases} \max\{\vec{u}_1[j], f_{\mathcal{R}_2}(\vec{x}[i])\} & \text{if } \vec{x}[i] = \vec{x}_1[j] \text{ and } \vec{x}[i] \notin \mathit{elem}(\vec{x}_2) \quad (2a) \\ \max\{f_{\mathcal{R}_1}(\vec{x}[i]), \vec{u}_2[k]\} & \text{if } \vec{x}[i] = \vec{x}_2[k] \text{ and } \vec{x}[i] \notin \mathit{elem}(\vec{x}_1) \quad (2b) \\ \max\{\vec{u}_1[j], \vec{u}_2[k]\} & \text{if } \vec{x}[i] = \vec{x}_1[j] = \vec{x}_2[k] \quad (2c) \\ f_{\mathcal{R}_1}(\vec{x}[i]) & \text{otherwise (it is an intersection)} \quad (2d) \end{cases}$$

- Finally, let $\vec{\ell}_1[1] = \vec{\ell}_2[1] = \perp$. Then, for all $i,\ 2 \leq i \leq |\vec{x}|,\ \vec{\ell}$ is defined similarly, where $1 \leq j \leq |\vec{x}_1|$ and $1 \leq k \leq |\vec{x}_2|$:

$$\vec{\ell}[i] = \begin{cases} \max\{\vec{\ell}_1[j], f_{\mathcal{R}_2}(\vec{x}[i])\} & \text{if } \vec{x}[i] = \vec{x}_1[j] \text{ and } \vec{x}[i] \notin \mathit{elem}(\vec{x}_2) \quad (3a) \\ \max\{f_{\mathcal{R}_1}(\vec{x}[i]), \vec{\ell}_2[k]\} & \text{if } \vec{x}[i] = \vec{x}_2[k] \text{ and } \vec{x}[i] \notin \mathit{elem}(\vec{x}_1) \quad (3b) \\ \max\{\vec{\ell}_1[j], \vec{\ell}_2[k]\} & \text{if } \vec{x}[i] = \vec{x}_1[j] = \vec{x}_2[k] \quad (3c) \\ f_{\mathcal{R}_1}(\vec{x}[i]) & \text{otherwise (it is an intersection)} \quad (3d) \end{cases}$$



(a) The function $f_{\mathcal{R}_1}$. (b) The function $f_{\mathcal{R}_2}$. (c) The function $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2}) = f_{\mathcal{R}}$.

(d) The function $f_{\mathcal{R}_3}$. (e) The function $f_{\mathcal{R}'} = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}}, f_{\mathcal{R}_3})$.

Fig. 6: Example of the maximum between energy functions.

Example 6. Consider the two energy functions $f_{\mathcal{R}_1}$ and $f_{\mathcal{R}_2}$ in Fig. 6(a) and 6(b). The function $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})$ is shown in Fig. 6(c). Its representation is given by $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$. Furthermore, we have the representation $\mathcal{R}_3 = (\vec{x}_3, \vec{m}_3, \vec{u}_3, \vec{\ell}_3)$ shown in Fig. 6(d).

$$\vec{x} = \langle 0, 10, 20 \rangle \qquad \vec{x}_3 = \langle 4, 15, 20 \rangle$$
$$\vec{m} = \langle 4, 10, 20 \rangle \qquad \vec{m}_3 = \langle 0, 20, 20 \rangle$$
$$\vec{u} = \langle 4, 10 \rangle \qquad \vec{u}_3 = \langle 4, 20 \rangle$$
$$\vec{\ell} = \langle 10, 20 \rangle \qquad \vec{\ell}_3 = \langle 6, 20 \rangle$$

The representation $\mathcal{R}'$ of the energy function $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}}, f_{\mathcal{R}_3})$, shown in Fig. 6(e), is defined as follows:

$$\vec{x} = \langle 0, 10, 15, 20 \rangle \qquad \vec{m} = \langle 4, 10, 20, 20 \rangle$$
$$\vec{u} = \langle 4, 10, 20 \rangle \qquad \vec{\ell} = \langle 10, 15, 20 \rangle$$

Let the size of the representation of an energy function, $|\mathcal{R}|$, be equal to the number of entries in the vector $\vec{x}$, since this is related to the number of entries in the other vectors. By doing this we abstract from the actual size of the values used within the vectors and assume that they have a constant size. We can then state the following theorem.

Theorem 2. *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be representations of energy functions and let $\mathcal{R} = Maximum(\mathcal{R}_1, \mathcal{R}_2)$. Then $f_{\mathcal{R}} = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})$ and $\mathcal{R}$ can be constructed in time $O(|\mathcal{R}_1| + |\mathcal{R}_2|)$.*

The proof is shown in Appendix A on page 47.

5.2 The Inverse of an Energy Function

When defining the composition operator, we will need to be able to get the inverse of an energy function, since the domain of one function will be directly related to the codomain of another. Given a monotone function $f : \mathbb{R} \to \mathbb{R}$ and a number $x \in \mathbb{R}$, the inverse of f is defined as a partial function $f^{-1}(y) = x$ if $f(x) = y$ and there does not exist $x' \neq x$ where $f(x) = f(x')$.

In our case, we need to consider that the domains are limited, meaning that there are codomain values to which there are no corresponding domain value. Therefore, the inverse of an energy function $f_{\mathcal{R}}$, where $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ is a function $f_{\mathcal{R}}^{-1} : ([0, B] \cup \{\perp\}) \to ([0, B] \cup \{\perp\})$ defined as:

$$f_{\mathcal{R}}^{-1}(y) = \begin{cases} \vec{x}[1] & \text{if } y = \vec{m}[1] \\ \vec{x}[i] & \text{if } y = \vec{m}[i] \text{ and } y \neq \vec{u}[i-1], \\ & \quad 1 < i \leq |\vec{x}| \\ \vec{x}[i] + (\vec{x}[i+1] - \vec{x}[i]) \frac{y - \vec{u}[i]}{\vec{\ell}[i+1] - \vec{u}[i]} & \text{if } \vec{u}[i] < y < \vec{\ell}[i+1], \\ & \quad 1 \leq i \leq |\vec{x}| - 1 \\ \perp & \text{otherwise} \end{cases}$$

Naturally, we can only define the inverse energy function for injective function intervals. That is, if the slope of $f_{\mathcal{R}}$ is zero in an interval, the value of $f_{\mathcal{R}}^{-1}$ will be $\perp$ for the values of the interval. An example of the inverse of an energy function can be seen in Fig. 7(a) and Fig. 7(b).

Proposition 1. *Let $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ be the representation of an energy function. Then for all $x \in [\vec{x}[1], B]$ such that there does not exist $x' \neq x$ where $f_{\mathcal{R}}(x) = f_{\mathcal{R}}(x')$, it holds that $f_{\mathcal{R}}^{-1}(f_{\mathcal{R}}(x)) = x$. Otherwise, $f_{\mathcal{R}}^{-1}(f_{\mathcal{R}}(x)) = \perp$.*

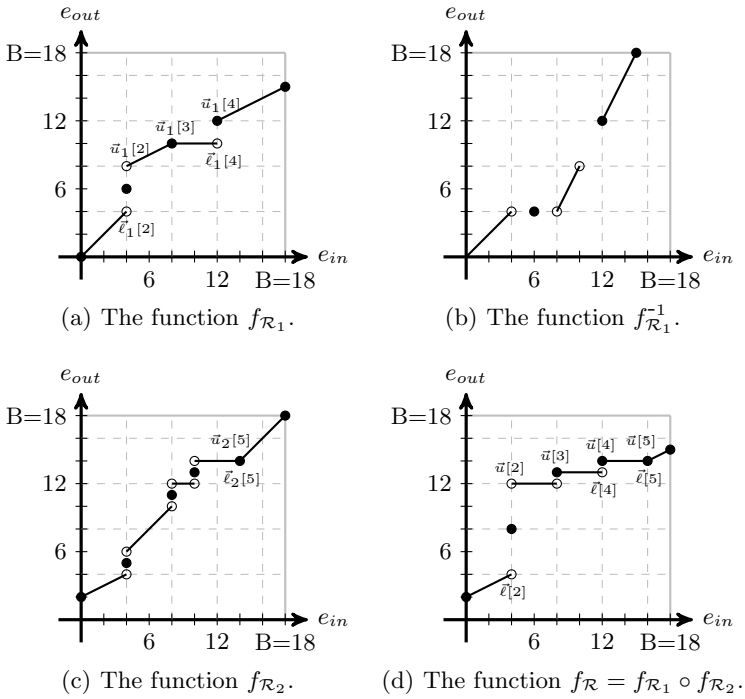
5.3 The Composition Operator

We define the composition operator $\circ : \mathcal{F} \times \mathcal{F} \to \mathcal{F}$ as:

$$(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(x)) \qquad \text{for all } x \in [0, B]$$

where the domain of the two functions is $[0, B]$, for some $B \in \mathbb{N}_0$.

We will start by taking a look at the algorithm and then afterwards explain it by a case analysis.



(a) The function $f_{\mathcal{R}_1}$.

(b) The function $f^{-1}_{\mathcal{R}_1}$.

(c) The function $f_{\mathcal{R}_2}$.

(d) The function $f_{\mathcal{R}} = f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$.

Fig. 7: The inverse of an energy function is shown in 7(a) and 7(b). Function composition of the functions in 7(a) and 7(c) is shown in 7(d)

Construction

Given two energy function representations $\mathcal{R}_1$ and $\mathcal{R}_2$, we can construct a representation $\mathcal{R}$ s.t. $f_{\mathcal{R}} = f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$.

Let $\mathcal{R}_1 = (\vec{x}_1, \vec{m}_1, \vec{u}_1, \vec{\ell}_1)$ and $\mathcal{R}_2 = (\vec{x}_2, \vec{m}_2, \vec{u}_2, \vec{\ell}_2)$. Moreover, let S_{comp} be the set of all points from $\vec{x}_1$, where the corresponding value in $\vec{m}_1$ is higher than or equal to the first value in $\vec{x}_2$ (values lower than $\vec{x}_2[1]$ would lead to undefined values). Furthermore, we add to S_{comp} all points obtained by applying the inverse of $f_{\mathcal{R}_1}$ to the values in $\vec{x}_2$, since $f_{\mathcal{R}_2}$ is applied last when calculating $f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$.

$$S_{comp} = \{x \mid x = \vec{x}_1[i],\ \vec{m}_1[i] \geq \vec{x}_2[1],\ 1 \leq i \leq |\vec{x}_1|\} \cup$$
$$\{x \mid x = f^{-1}_{\mathcal{R}_1}(x_2) \neq \bot, x_2 \in elem(\vec{x}_2)\}$$

Example 7. The set S_{comp} for the composition of $f_{\mathcal{R}_1}$ and $f_{\mathcal{R}_2}$ looks as shown below. The first set contains all values of $\vec{x}_1$, while the second set contains the values from $f_{\mathcal{R}_1}^{-1}$.

$$S_{comp} = \{0, 4, 8, 12, 18\} \cup \{0, 16\} = \{0, 4, 8, 12, 16, 18\}$$

As can be seen on Fig. 7(d), these are exactly the values of $\vec{x}$.

At all points in S_{comp} there is a change in the linear functions used, and thereby there is a change in the composed function. Based on this set we construct the composed function $f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$ as done in Algorithm 2.

Algorithm 2: The algorithm calculates the representation of the energy function $f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$.

Composition($\mathcal{R}_1$, $\mathcal{R}_2$)
Input - $\mathcal{R}_1 = (\vec{x}_1, \vec{m}_1, \vec{u}_1, \vec{\ell}_1)$, $\mathcal{R}_2 = (\vec{x}_2, \vec{m}_2, \vec{u}_2, \vec{\ell}_2)$
Output - $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ such that $f_{\mathcal{R}} = f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$

- The vector $\vec{x}$ is the increasing sequence containing all elements in S_{comp}.
- For all $i,\ 1 \leq i \leq |S_{comp}|$, we have:

$$\vec{m}[i] = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(\vec{x}[i])) \tag{4}$$

- For all $i,\ 1 \leq i \leq |S_{comp}| - 1$, $\vec{u}$ is defined as follows, where $1 \leq j < |\vec{x}_1| - 1$ and $1 \leq k < |\vec{x}_2| - 1$:

$$\vec{u}[i] = \begin{cases} f_{\mathcal{R}_2}(\vec{u}_1[j]) & \text{if } \vec{x}[i] = \vec{x}_1[j] \text{ and } \vec{u}_1[j] \notin \mathit{elem}(\vec{x}_2) \quad (5a) \\ \vec{m}_2[k] & \text{if } \vec{x}[i] = \vec{x}_1[j] \\ & \quad \text{and } \vec{u}_1[j] = \vec{x}_2[k], \vec{u}_1[j] = \vec{\ell}_1[j+1] \quad (5b) \\ \vec{u}_2[k] & \text{if } \vec{x}[i] = \vec{x}_1[j] \\ & \quad \vec{u}_1[j] = \vec{x}_2[k], \vec{\ell}_1[j+1] \neq \vec{u}_1[j] \quad (5c) \\ \vec{u}_2[k] & \text{if } \vec{x}[i] \notin \mathit{elem}(\vec{x}_1),\ \vec{x}[i] = f_{\mathcal{R}_1}^{-1}(\vec{x}_2[k]) \quad (5d) \end{cases}$$

- Likewise, for all $i,\ 2 \leq i \leq n$, $\vec{\ell}$ is defined as follows, where $1 < j \leq |\vec{x}_1|$ and $1 < k \leq |\vec{x}_2|$:

$$\vec{\ell}[i] = \begin{cases} f_{\mathcal{R}_2}(\vec{\ell}_1[j]) & \text{if } \vec{x}[i] = \vec{x}_1[j] \text{ and } \vec{\ell}_1[j] \notin \mathit{elem}(\vec{x}_2) \quad (6a) \\ \vec{m}_2[k] & \text{if } \vec{x}[i] = \vec{x}_1[j], \\ & \quad \text{and } \vec{\ell}_1[j] = \vec{x}_2[k], \vec{u}_1[j-1] = \vec{\ell}_1[j] \quad (6b) \\ \vec{\ell}_2[k] & \text{if } \vec{x}[i] = \vec{x}_1[j], \\ & \quad \text{and } \vec{\ell}_1[j] = \vec{x}_2[k], \vec{u}_1[j-1] \neq \vec{\ell}_1[j] \quad (6c) \\ \vec{\ell}_2[k] & \text{if } \vec{x}[i] \notin \mathit{elem}(\vec{x}_1),\ \vec{x}[i] = f_{\mathcal{R}_1}^{-1}(\vec{x}_2[k]) \quad (6d) \end{cases}$$

To better understand the construction of $f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}$, we will need to consider how the functions behave. If we take a look at Fig. 7, we can observe several of these cases by observing selected points, which we will use to give an intuition of the construction shown in Algorithm 2.

Let $\mathcal{R}_1 = (\vec{x}_1, \vec{m}_1, \vec{u}_1, \vec{\ell}_1)$, $\mathcal{R}_2 = (\vec{x}_2, \vec{m}_2, \vec{u}_2, \vec{\ell}_2)$ and let the representation of the composition of $f_{\mathcal{R}_1}$ and $f_{\mathcal{R}_2}$ be $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$. Then consider the following cases of the algorithm:

- **Case 5a and 6a**
  Consider the point $\vec{u}[4]$ in Figure 7(d). We are considering a value of $\vec{x}$ that was taken from $\vec{x}_1$, but where the corresponding value in $\vec{u}_1$ does not match a value in $\vec{x}_2$. This means that $f_{\mathcal{R}_2}$ is continuous in the point $\vec{u}_1$ and as such the value of $\vec{u}$ can be found by interpolation. This point is handled by case 5a in Algorithm 2 and similarly we have case 6a for vector $\vec{\ell}$.

- **Case 5b and 6b**
  The point $\vec{u}[3]$ is an example of this case. Here the slope of $f_{\mathcal{R}_1}$ is zero in the region $]\vec{x}_1[3], \vec{x}_1[4][$. As such the function reaches the value $\vec{u}_1[3]$, which corresponds to a domain value of $f_{\mathcal{R}_2}$ in $\vec{x}_2$. Thus the function value of $f_{\mathcal{R}}$ is equal to the value of $\vec{m}_2$ at that domain value. This corresponds to case 5b in the algorithm and similarly we have case 6b for vector $\vec{\ell}$.

- **Case 5c and 6c**
  A third example of function behaviour is shown in the computation of point $\vec{u}[2]$. Notice that the slope of $f_{\mathcal{R}_1}$ is not zero in the region after $\vec{x}[2]$, $]\vec{x}_1[2], \vec{x}_1[3][$. This means that the value $\vec{u}_1[2]$ is never actually obtained in the region. Since $\vec{u}_1[2]$ corresponds to a value in $\vec{x}_2$, we need the corresponding value from $\vec{u}_2$ to delimit $f_{\mathcal{R}}$ in the interval. As such, at $\vec{x}[2]$ we have $\vec{u}[2] = 12$. This is what is taken care of in case 5c in the algorithm and similarly we have case 6c for vector $\vec{\ell}$.

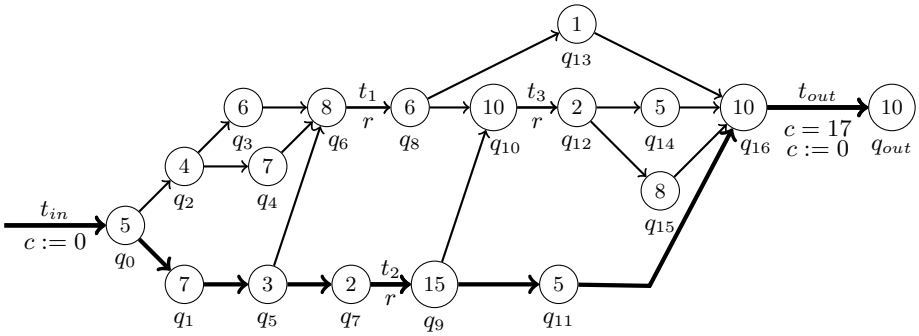
- **Case 5d and 6d**
  Finally, we have point $\vec{u}[5]$ in Fig. 7(d). Here, $\vec{x}[5]$ was not present in $\vec{x}_1$, but comes from a value in $\vec{x}_2$, namely $\vec{x}_2[5]$. Thus, we use the value $\vec{u}_2[5]$ for $\vec{u}[5]$. Case 5d in the algorithm covers this circumstance and similarly we have case 6d for vector $\vec{\ell}$.

Theorem 3. *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be energy functions. We can in time $O(|\mathcal{R}_1|+|\mathcal{R}_2|)$ construct $\mathcal{R} = Composition(\mathcal{R}_1, \mathcal{R}_2)$, s.t. $f_{\mathcal{R}} = (f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})$.*

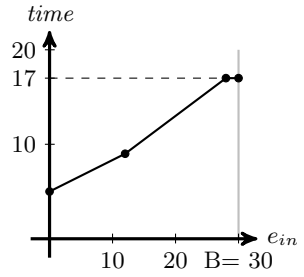
The proof is shown in Appendix B on page 51.

Theorem 4. *The class of energy functions is closed under $\mathcal{M}\boldsymbol{ax}$ and $\circ$.*

Proof. Follows from Theorem 2 and Theorem 3. □



(a) Automaton segment. Recharge transitions are marked with an r and an example of a path is highlighted with thick transitions. Here, $B = 30$ and all invariants are $c \leq 17$.



(b) The time function for the recharge transition t_3. Here $t_{in} = (g_{in}, z_{in}, c := 0)$ and $t_{out} = (c = 17, z_{out}, c := 0)$.

Fig. 8: Example of a time function based on an automaton segment.

6 Construction of an Energy Function Representation

We will in this section only consider automata segments of the form shown in Fig. 8(a), where we have exactly two reset transitions, t_{in} and t_{out}, and all locations are on some path from t_{in} to t_{out}. The intention is to use energy functions to abstract these paths and in the end create an abstraction of a whole recharge automaton, simplifying the task of finding infinite runs where the energy does not go below zero at any time.

Formally, we call such an automaton segment a *snippet* from t_{in} to t_{out}. For convenience, given a transition t, we will denote its source location by ${}^\bullet t$ and its target location by $t^\bullet$. Moreover, let $\mathit{Paths}(t_{in}, t_{out})$ denote the set of all paths going from $t_{in}^\bullet$ to $t_{out}^\bullet$, where t_{out} is the last transition.

$$\mathit{Paths}(t_{in}, t_{out}) = \{\pi \mid \pi = t_{in}^\bullet \xrightarrow{true, z_1, \emptyset} \dots \xrightarrow{t_{out}} t_{out}^\bullet\}$$

Additionally, let $\mathit{loc}(\pi)$ denote all locations and $\mathit{trans}(\pi)$ denote all transitions along the path π. We can then define a snippet of a recharge automaton as follows.

Definition 6 (Snippet). *Let $\mathcal{A} = (Q, q_0, B, C, I, \mathit{rate}, \Delta)$ be a one-clock closed recharge automaton in rfcNF. A snippet $\mathcal{S}$ of $\mathcal{A}$ is a tuple $(t_{in}, t_{out}, Q', B', C', k, \mathit{rate}', \Delta')$ such that*

- *$t_{in} \in \Delta$ is a reset transition,*
- *$t_{out} \in \Delta$ is a reset transition such that $\mathit{Paths}(t_{in}, t_{out}) \neq \emptyset$,*
- *$Q' \subseteq Q$ such that $Q' = \{q \mid q \in \mathit{loc}(\pi)$ for some $\pi \in \mathit{Paths}(t_{in}, t_{out})\}$.*
- *$B' = B$,*
- *$C' = C$,*
- *$k \in \mathbb{N}_0$, where $I(q) =$ "$c \leq k$" for all $q \in Q' \setminus \{t_{out}^\bullet\}$,*
- *$\mathit{rate}' = \mathit{rate}$ and*
- *$\Delta' \subseteq \Delta$ such that $\Delta' = \{t \mid t \in \mathit{trans}(\pi)$ for some $\pi \in \mathit{Paths}(t_{in}, t_{out})\}$.*

In the rest of this section we will only consider a fixed snippet $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$.

Let $\mathit{Runs}(\mathcal{S})$ denote the set of all simple run segments starting in $t_{in}^\bullet$ with valuation $c = 0$ and ending in some $q \in Q$.

$$\mathit{Runs}(\mathcal{S}) = \{\gamma \mid \gamma = (t_{in}^\bullet, [c = 0], e_{in}) \to \dots \to (q, v, e) \text{ is a simple run segment in } \mathcal{S}\}$$

The energy function for a snippet determines the maximal energy level that can be obtained in $t_{out}^\bullet$ given the energy level in $t_{in}^\bullet$ along a run in $\mathit{Runs}(\mathcal{S})$.

As we proved in Lemma 1, having a higher energy level in a given state allows at least to take the same transitions as a similar state with lower energy. A higher energy level may even allow to take new paths. For example, in Fig. 8(a) assume that we only have the highlighted path from q_{in} to q_{out} and an initial energy of 0. Then we would only be able to spend 6 time units in q_{11} on the path to q_{16},

which would not be enough to satisfy the guard of the transition t_{out}. On the other hand, having an initial energy level of 22 or above would allow to spend 17 time units in total and reach q_{out}, allowing this run to continue from there.

Each snippet is potentially followed by another one, so when we leave the current snippet, we prefer to do this with as high an energy level as possible. In this way, the initial energy of the next snippet is increased and so is the likelihood of being able to satisfy its guard. It is possible to maximize the energy level by analysing the set of runs in the snippet and selecting the ones spending most time while conserving energy. We will use a notion of time functions for this purpose, to be defined in the next section.

6.1 Time Functions

In this section we consider a fixed snippet $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$.

A time function f for a transition $t \in \Delta$ takes as input an energy level $e_{in} \in [0, B]$ and returns a value in $\mathbb{R}_{\geq 0}$, determining how much time can at most be spent on a run starting in state $(t_{in}^{\bullet}, [c = 0], e_{in})$ and having t as last transition.

$$f_t(e_{in}) = \max \left\{ \sum_{0<i\leq j} d_i \,\middle|\, (t_{in}^{\bullet}, [c=0], e_{in}) = (q_1, v_1, e_1) \xrightarrow{d_1} (q_1, v_1', e_1') \xrightarrow{t_1} \ldots \xrightarrow{d_j} (q_j, v_j', e_j') \xrightarrow{t} (q_{j+1}, v_{j+1}, e_{j+1}) \right\}$$

We are interested in maximizing the cumulative delay before the transition t_{out} along some path in the snippet to find out whether it is possible to satisfy the guard of t_{out}. If we can even obtain a cumulative delay that would satisfy this guard before reaching a recharge transition, it means that there is a run that reaches $t_{out}^{\bullet}$ with energy level B.

Observe as an example Fig. 8(a), where we can find out whether there exists a run that reaches the location q_{out} by means of a path in $\mathit{Paths}(t_{in}, t_{out})$ by strategically determining where delays are performed along the path. For example, arriving at location q_7 with energy higher than zero means that we might as well delay some time in q_7 (or an earlier location) before taking the transition t_2, since it has a recharge. In general, we want to spend as much time as we can to be able to satisfy the guard of t_{out}, and since the energy at q_9 will be B independent of the energy level before taking the transition t_2, we can spend more time by delaying until the energy is zero. An example of this can be seen on Fig. 9(a) and 9(b). Here, the evolution of the energy level of two runs, γ_1 and

γ_2, along the highlighted path of the snippet in Fig. 8(a) is shown.

$$\begin{aligned}\gamma_1 = (q_0, [c=0], 30) &\xrightarrow{\mathbf{1}} (q_0, [c=1], 25) \xrightarrow{true,\epsilon,\emptyset} (q_1, [c=1], 25)\\ &\xrightarrow{0} (q_1, [c=1], 25) \xrightarrow{true,\epsilon,\emptyset} (q_5, [c=1], 25) \xrightarrow{0} (q_5, [c=1], 25)\\ &\xrightarrow{true,\epsilon,\emptyset} (q_7, [c=1], 25) \xrightarrow{\mathbf{7}} (q_7, [c=8], 11) \xrightarrow{true,\mathbf{r},\emptyset} (q_9, [c=8], 30)\\ &\xrightarrow{0} (q_9, [c=8], 30) \xrightarrow{true,\epsilon,\emptyset} (q_{11}, [c=8], 30) \xrightarrow{\mathbf{3}} (q_{11}, [c=11], 15)\\ &\xrightarrow{true,\epsilon,\emptyset} (q_{16}, [c=11], 15) \xrightarrow{\mathbf{1.5}} (q_{16}, [c=12.5], 0)\end{aligned}$$

$$\begin{aligned}\gamma_2 = (q_0, [c=0], 30) &\xrightarrow{\mathbf{1}} (q_0, [c=1], 25) \xrightarrow{true,\epsilon,\emptyset} (q_1, [c=1], 25)\\ &\xrightarrow{0} (q_1, [c=1], 25) \xrightarrow{true,\epsilon,\emptyset} (q_5, [c=1], 25) \xrightarrow{0} (q_5, [c=1], 25)\\ &\xrightarrow{true,\epsilon,\emptyset} (q_7, [c=1], 25) \xrightarrow{\mathbf{12.5}} (q_7, [c=13.5], 0) \xrightarrow{true,\mathbf{r},\emptyset} (q_9, [c=13.5], 30)\\ &\xrightarrow{0} (q_9, [c=13.5], 30) \xrightarrow{true,\epsilon,\emptyset} (q_{11}, [c=13.5], 30) \xrightarrow{\mathbf{3}} (q_{11}, [c=16.5], 15)\\ &\xrightarrow{true,\epsilon,\emptyset} (q_{16}, [c=16.5], 15) \xrightarrow{\mathbf{0.5}} (q_{16}, [c=17], 10)\\ &\xrightarrow{\mathbf{c=\ 17},\epsilon,\mathbf{\{c\}}} (q_{out}, [c=0], 10)\end{aligned}$$

During the first run, the recharge transition t_2 is taken after 8 time units, while the energy is above zero, and the run is not able to reach the 17 time units required to satisfy the guard of t_{out}. However, if we delay before taking the transition t_2 until we reach zero energy, as done in the second run, it is possible to obtain a total delay of 17 time units and even have some energy left when taking t_{out}.

The representation of time functions is similar to the representation of energy functions. Though, since time functions are continuous, we need only two vectors.

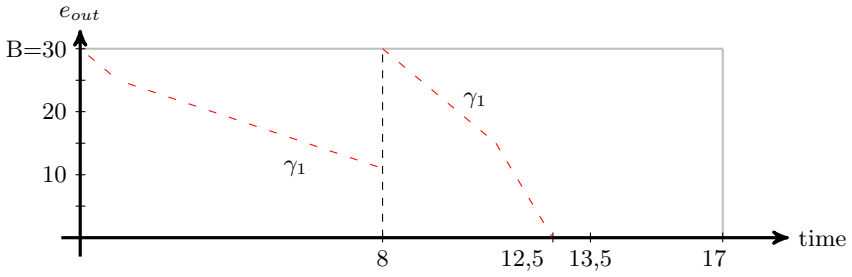
Definition 7 (Time Function - Representation and Semantics). *Given a maximal energy level $B \in \mathbb{N}_0$ and a value $k \in \mathbb{N}_0$, a time function is represented by a tuple $\mathcal{T} = (\vec{x}, \vec{y})$, where*

- *$\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ is an increasing sequence of values s.t. for all i, $1 \leq i \leq n$, $x_i \in \mathbb{Q}$, $0 \leq x_i \leq B$, $x_1 = 0$, $x_n = B$,*
- *$\vec{y} = \langle y_1, y_2, \ldots, y_n \rangle$ is a sequence of values s.t. for all i, $1 \leq i \leq n-1$, $y_i, y_{i+1} \in \mathbb{Q}$, $0 \leq y_i, y_{i+1} \leq k$ and $y_i < y_{i+1}$ if $y_i < k$, else $y_i = y_{i+1}$.*

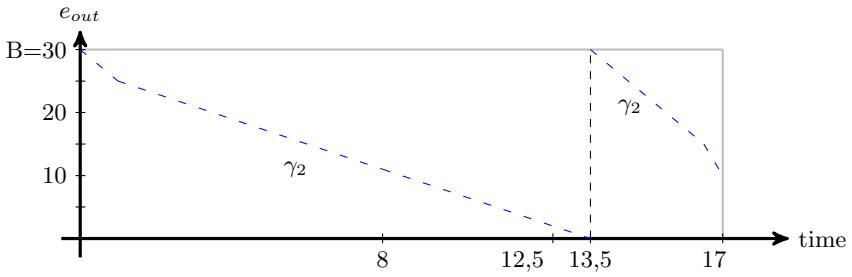
The representation $\mathcal{T}$ defines a time function $f_{\mathcal{T}} : [0, B] \to [0, k]$ such that:

$$f_{\mathcal{T}}(x) = y_i + (y_{i+1} - y_i)\frac{x - x_i}{x_{i+1} - x_i} \qquad \text{where } x_i \leq x \leq x_{i+1},\ 1 \leq i \leq n-1$$

As an example, the time function abstracting paths of Fig. 8(a) going from t_{in} to t_3 is shown in Fig. 8(b). We can observe on the graph that e_{in} has to be at least 28 to satisfy the guard of t_{out}. Each of the linear segments of the function



(a) The run γ_1.



(b) The run γ_2.

Fig. 9: Two runs of the snippet in Fig. 8(a). Using all energy before taking a recharge transition is desirable as it allows spending more time.

represents a path which is optimal for the given values of e_{in}. We will explain later how this function is actually constructed.

Similarly to the inverse of energy functions, we define the inverse of a time function $f_{\mathcal{T}}$ as a function $f_{\mathcal{T}}^{-1} : [0, k] \to [0, B] \cup \{\bot\}$:

$$f_{\mathcal{T}}^{-1}(y) = \begin{cases} x_1 & \text{if } y \leq y_1 \\ x_i & \text{if } y = y_i \neq y_{i-1},\ 2 \leq i \leq |\vec{x}| \\ x_i + (x_{i+1} - x_i)\frac{y - y_i}{y_{i+1} - y_i} & \text{if } y_i < y < y_{i+1},\ 1 \leq i \leq |\vec{x}| - 1 \\ \bot & \text{otherwise} \end{cases}$$

Note that the semantics of the inverse of a time function is different from energy functions with respect to undefined values. For example, the inverse of a time function is defined for all values between 0 and the first defined function value y_1, while the inverse of an energy function returns $\bot$ for those values.

Proposition 2. *Let $\mathcal{T} = (\vec{x}, \vec{m})$ be a time function representation with domain $[0, B]$ for some $B \in \mathbb{N}_0$. Then, for all $x \in [0, B]$ such that there does not exist $x' \neq x$ where $f_{\mathcal{T}}(x') = f_{\mathcal{T}}(x)$, it holds that $f_{\mathcal{T}}^{-1}(f_{\mathcal{T}}(x)) = x$. Otherwise, $f_{\mathcal{T}}^{-1}(f_{\mathcal{T}}(x)) = \min\{x' \mid f_{\mathcal{T}}(x') = f_{\mathcal{T}}(x)\}$.*

Proposition 3. *Given a time function representation $\mathcal{T} = (\vec{x}, \vec{m})$, we have that the time function $f_{\mathcal{T}}$ is non-decreasing, continuous and piecewise affine.*

Similarly to energy functions, we need to be able to take the maximum between two time functions, $f_{\mathcal{T}_1}$ and $f_{\mathcal{T}_2}$ with the same domain $[0, B]$ for some $B \in \mathbb{N}_0$.

$$\mathcal{M}\boldsymbol{ax}_{\mathcal{T}}(f_{\mathcal{T}_1}, f_{\mathcal{T}_2})(x) = \max\{f_{\mathcal{T}_1}(x), f_{\mathcal{T}_2}(x)\} \qquad \text{for all } x \in [0, B]$$

The representation for the time function $\mathcal{M}\boldsymbol{ax}_{\mathcal{T}}(f_{\mathcal{T}_1}, f_{\mathcal{T}_2})$ can be constructed in a similar way to constructing the maximum between energy functions. We will call the algorithm constructing this representation $\mathit{Maximum}_{\mathcal{T}}$.

6.2 Energy Function

In this section we will only consider a fixed snippet $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$.

All the runs in $\mathit{Runs}(\mathcal{S})$ that include the transition t_{out} start in a state where the clock valuation is zero and are required to reach a value satisfying the guard of t_{out}. It is possible to calculate an energy function representing, for an energy level $e_{in} \in [0, B]$, the maximal energy that can be obtained when satisfying the guard and reaching $t^{\bullet}_{out}$ on any of the runs in $\mathit{Runs}(\mathcal{S})$ starting in state $(t^{\bullet}_{in}, [c = 0], e_{in})$. Since there may be infinitely many runs in this set, we will use time functions to find the ones that lead to the highest energy level at $t^{\bullet}_{out}$.

Let $RT(\mathcal{S})$ be a function that returns a set of all recharge transitions in $\mathcal{S}$. We will construct a partial order of the recharge transitions included along any path in $\mathit{Paths}(t_{in}, t_{out})$. This is possible since the snippet is a part of a recharge automaton in rfcNF, where all cycles include a clock reset.

Definition 8 (Recharge Ordering). *Let $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$ be a snippet. We define a recharge ordering $\lhd \subseteq (RT(\mathcal{S}) \cup \{t_{in}, t_{out}\}) \times (RT(\mathcal{S}) \cup \{t_{in}, t_{out}\})$ s.t. $t \lhd t'$, iff there exists a path $\pi = t^{\bullet} \xrightarrow{true,\epsilon,\emptyset} \ldots \xrightarrow{true,\epsilon,\emptyset} {}^{\bullet}t'$ with no recharge transitions in the snippet $\mathcal{S}$.*

Note that $\lhd$ is not transitive.

We will use the recharge ordering to calculate the energy function of a snippet $\mathcal{S}$. First of all, we calculate the time function representations abstracting paths from t_{in} to each of the recharge transitions t in the snippet. For simplicity, we make use of an addition operator that adds a rational number $n \in \mathbb{Q}_{\geq 0}$ to a time function $f_{\mathcal{T}}$.

$$(f_{\mathcal{T}} + n)(x) = f_{\mathcal{T}}(x) + n$$

Additionally, we introduce the operator $\mathit{ceil}(f_{\mathcal{T}}, k)$ that ensures that the time function only returns a value that is at most k, for some constant $k \in \mathbb{N}_0$. We will use this operation to make sure that all values satisfy the invariant $c \leq k$.

$$\mathit{ceil}(f_{\mathcal{T}}, k)(x) = \begin{cases} f_{\mathcal{T}}(x) & \text{if } f_{\mathcal{T}}(x) < k \\ k & \text{otherwise} \end{cases}$$

For details on how both of the operators can be implemented, refer to Appendix C on page 55.

Before we show how to construct the time functions that are relevant for the snippet $\mathcal{S}$, we define the function $\mu(t, t')$ which for a snippet returns the minimal rate of all locations on paths between t and t' where no other transitions with recharges are used:

$$\mu(t, t') = \min\{\mathit{rate}(q) \mid \exists \pi = {}^\bullet t \xrightarrow{t} \ldots \xrightarrow{\mathit{true},\epsilon,\emptyset} q \xrightarrow{\mathit{true},\epsilon,\emptyset} \ldots \xrightarrow{t'} t'^\bullet\}$$

Algorithm 3: The algorithm calculates the time function representations each abstracting paths from t_{in} to a recharge transition t in the snippet $\mathcal{S}$ and returns them in the array A.

TimeFunction($\mathcal{S}$)
Input - $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$
Output - A, such that for all $t \in RT(\mathcal{S})$ and for all $e_{in} \in [0, B]$,

$$f_{A[t]}(e_{in}) = \max\left\{ \sum_{0<i\leq j} d_i \,\middle|\, \exists\ \gamma \in \mathit{Runs}(\mathcal{S}) \text{ s.t. } \gamma = (t_{in}{}^\bullet, v_{in}, e_{in}) \xrightarrow{d_1} (t_{in}{}^\bullet, v'_{in}, e'_{in}) \xrightarrow{t_1} \ldots \xrightarrow{d_j} ({}^\bullet t, v'_j, e'_j) \right\}$$

```
1  begin
2      S = RT(S)
3      while S ≠ ∅ do
4          if S \ {t_out} ≠ ∅ then
5              select a minimum element t ∈ S \ {t_out}
6          else
7              t = t_out
8          S = S \ {t}
9          if t_in ◁ t then
10             if μ(t_in, t) > 0 then
11                 A[t] = ceil((⟨0, B⟩, ⟨0, B/μ(t_in,t)⟩), k)
12             else
13                 A[t] = (⟨0, B⟩, ⟨k, k⟩)
14         else
15             A[t] = (⟨0, B⟩, ⟨0, 0⟩)
16         forall the t' ∈ RT(S) \ {t_out} where t' ◁ t do
17             if μ(t', t) > 0 then
18                 temp = ceil(A[t'] + B/μ(t',t), k)
19                 A[t] = Maximum_T(temp, A[t])
20             else
21                 A[t] = (⟨0, B⟩, ⟨k, k⟩)
22     return A
```

In Algorithm 3 *TimeFunction* takes a snippet $\mathcal{S}$ as input and returns an array A that records a time function for each recharge transition in the snippet.

The algorithm initially finds all recharge transitions in the snippet and then goes through them one by one following their recharge ordering. Note that special caution is taken if $t_{in} = t_{out}$ and $t_{out} \in RT(\mathcal{S})$, since there would not be a minimal element in S. If there exists a path without recharge transitions between t_{in} and a transition with a recharge t, then a representation of a time function returning the maximal delay that can be achieved on a run following such a path is constructed. This is done based on the location with the minimal rate on one of the considered paths.

All recharge transitions preceding t in the recharge ordering are then considered and, based on those, the maximal time that can be spent along paths between t_{in} and t is calculated. In the special case where there exists a minimal rate of zero along one of the paths, a time function representation that returns the maximal value allowed by the invariant for the entire domain is given. Finally, the algorithm returns an array of time function representations A abstracting paths between t_{in} and all recharge transitions in the snippet.

Example 8. An example of the computations performed by the algorithm on the automaton segment in Fig. 8(a) are shown in Fig. 10.

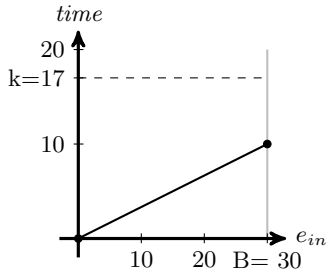
Initially, the time function abstracting the paths without recharge transitions between t_{in} and t_1 is calculated. Since the minimal rate is $rate(q_5) = 3$ and the bound is 30, the time function will be a straight line between $f(0) = 0$ and $f(30) = 10$, as seen in Fig. 10(a). Likewise, the function between t_{in} and t_2 is shown in Fig. 10(b). Since there is no path without recharges going from t_{in} to t_3, the initial time function for t_3 is undefined.

We can now construct the time function for paths going to t_3 through t_1. The resulting function in Fig. 10(c) is constructed by taking the one from Fig. 10(a) and raising its values by 5, since the lowest rate between t_1 and t_3 is 6 and $\frac{30}{6} = 5$. Similarly, the lowest rate between t_2 and t_3 is 10, giving the function shown in Fig. 10(d). When taking the maximum between the functions of Fig. 10(c) and 10(d), we obtain the time function abstracting the delay on paths from t_{in} to t_3. This is shown in Fig. 10(e).

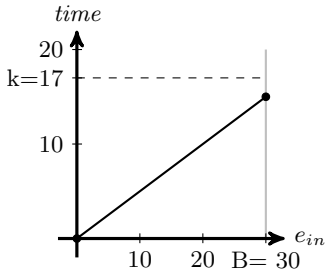
Theorem 5. *Let $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$ be a snippet. Then the algorithm TimeFunction($\mathcal{S}$) terminates and returns an array A, such that for all $t \in RT(\mathcal{S})$, and for all $e_{in} \in [0, B]$*

$$f_{A[t]}(e_{in}) = \max \left\{ \sum_{0<i\leq j} d_i \,\middle|\, (t_{in}{}^\bullet, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} (t_{in}{}^\bullet, v'_{in}, e'_{in}) \xrightarrow{t_1} \dots \xrightarrow{d_j} ({}^\bullet t, v'_j, e'_j) \right\}.$$

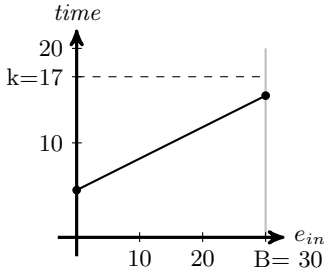
The proof is based on a loop invariant and is shown in Appendix D on page 57.



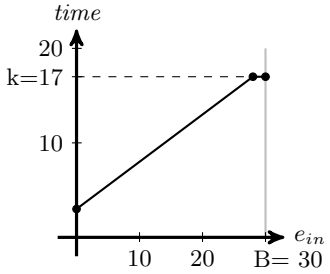
(a) Time function for t_1.



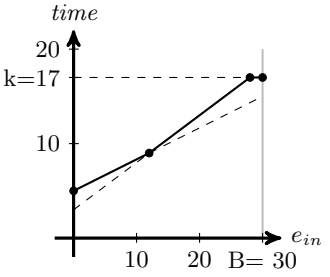
(b) Time function for t_2.



(c) Time function for t_3 through t_1.



(d) Time function for t_3 through t_2.



(e) The final time function for t_3.

Fig. 10: How to calculate a time function.

Now we can use these time functions to calculate the energy function abstracting the paths in $\mathit{Paths}(t_{in}, t_{out})$. When the guard of t_{out} requires that some time is spent in the snippet, we can use time functions to find out whether it is possible to delay enough time to satisfy the guard. Using all of the available energy after the last recharge further increases the likelihood of satisfying this guard. Therefore, given A as the result of $\mathit{TimeFunction}(\mathcal{S})$ and a recharge transition $t \neq t_{out}$ s.t. $t \lhd t_{out}$ and $\mu(t, t_{out}) > 0$, we can construct the following time function:

$$\mathcal{T} = \mathit{ceil}\left(A[t] + \frac{B}{\mu(t, t_{out})}, k\right)$$

Then the smallest energy level for which it is possible to satisfy the guard of t_{out} will be $\mathit{first}(t) = f_{\mathcal{T}}^{-1}(k)$.

If it is possible to delay long enough to satisfy the guard before the last recharge transition on a path to $t_{out}^{\bullet}$, it is possible to reach $t_{out}^{\bullet}$ with the maximal energy level B. For a given recharge transition t, we define $\mathit{last}(t)$ to be either the minimal value x such that $f_{A[t]}(x) = k$ or B in case no such value exists:

$$\mathit{last}(t) = \begin{cases} f_{A[t]}^{-1}(k) & \text{if } f_{A[t]}^{-1}(k) \neq \bot \\ B & \text{otherwise} \end{cases}$$

We can now use Algorithm 4 to construct the energy function for the snippet $\mathcal{S}$. Initially, the time function array A is initialized by calling *TimeFunction*. We then look for a proper representation of an energy function to abstract the paths in $\mathit{Paths}(t_{in}, t_{out})$.

If the guard of t_{out} is of the form "$c = 0$" or "*true*", the algorithm returns a representation that maps any energy level to either full energy in case there is a recharge transition along the path or a "$x = y$" function in case there are none, meaning that we end up with the same energy that we started with.

Otherwise, the guard of t_{out} will have the form "$c = k$", where $k \neq 0$. This means that we are required to delay k time units before taking the transition t_{out}. If t_{out} has a recharge, the energy function will return the maximal energy whenever it can delay enough time to satisfy the guard. For all levels of energy at which this delay cannot be performed the function will instead be undefined.

If t_{out} does not have a recharge, we will consider all recharge transitions t such that $t \lhd t_{out}$. Each of these transitions may be the last transition with a recharge before taking t_{out}. As we did when finding the time function, we look for a location with minimal rate and calculate the final energy function based on this and the time that we still need to wait to satisfy the guard of t_{out}. The function will then be defined in the function *SetupVectors* based on $\mathit{first}(t)$. The vectors $\vec{m}$, $\vec{u}$ and $\vec{\ell}$ are set up similarly.

Note that $\mathcal{R}$ will not be defined if no input energy is enough to satisfy the guard of t_{out} in the snippet.

Algorithm 4: Calculating the energy function abstracting a snippet.

EnergyFunction($\mathcal{S}$)
Input - $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$, where $t_{out} = ({}^\bullet t_{out}, g_{out}, z_{out}, \{c\}, t_{out}^\bullet)$
Output - A representation $\mathcal{R}$ such that, if $\mathcal{R}$ is defined, it holds that for all $e_{in} \in [0, B]$, $f_{\mathcal{R}}(e_{in}) = \max\Big\{e_{out} \Big|\ \exists\ \gamma \in \mathit{Runs}(\mathcal{S})$ s.t. $\gamma = (t_{in}^\bullet, v_{in} = [c = 0], e_{in}) \rightarrow \ldots \xrightarrow{t_{out}} (t_{out}^\bullet, v_{out} = [c = 0], e_{out})\Big\}$

begin
  $A = \mathit{TimeFunction}(\mathcal{S})$
  if $g_{out} = \mathit{true}$ *or* $g_{out} =$ "$c = 0$" **then**
    if $RT(\mathcal{S}) \neq \emptyset$ **then**
      $\mathcal{R} = (\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$ $\quad //f_{\mathcal{R}}(x) = B,\ x \in [0, B]$
    else
      $\mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$ $\quad //f_{\mathcal{R}}(x) = x,\ x \in [0, B]$
  else if $g_{out} =$ "$c = k$" **then**
    if $t_{out} \in RT(\mathcal{S})$ **then**
      $e_{in} = f^{-1}_{A[t_{out}]}(k)$
      if $e_{in} \neq \perp$ **then**
        if $e_{in} \neq B$ **then**
          $\mathcal{R} = (\langle e_{in}, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$ $\quad //f_{\mathcal{R}}(x) = B,\ x \in [e_{in}, B]$
        else
          $\mathcal{R} = (\langle e_{in}\rangle, \langle 0\rangle, \langle\ \rangle, \langle\ \rangle)$ $\quad //f_{\mathcal{R}}(x) = 0$ if $x = B$
    else
      if $t_{in} \lhd t_{out}$ **then**
        if $\mu(t_{in}, t_{out}) = 0$ **then**
          $\mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$ $\quad //f_{\mathcal{R}}(x) = x,\ x \in [0, B]$
        else
          $\mathcal{T} = \mathit{ceil}\Big((\langle 0, B\rangle, \langle 0, \frac{B}{\mu(t_{in}, t_{out})}\rangle), k\Big)$
          $e_{in} = f^{-1}_{\mathcal{T}}(k)$
          if $e_{in} \neq \perp$ **then**
            if $e_{in} \neq B$ **then**
              $\mathcal{R} = \Big(\langle e_{in}, B\rangle, \langle 0, B - k \cdot \mu(t_{in}, t_{out})\rangle, \langle 0\rangle, \langle B - k \cdot \mu(t_{in}, t_{out})\rangle\Big)$
              $//f_{\mathcal{R}}(x) = x - k \cdot \mu(t_{in}, t_{out}),\ x \in [e_{in}, B]$
            else
              $\mathcal{R} = (\langle e_{in}\rangle, \langle 0\rangle, \langle\ \rangle, \langle\ \rangle)$ $\quad //f_{\mathcal{R}}(x) = 0$ if $x = B$
      forall the *transitions* $t \in RT(\mathcal{S}) \setminus \{t_{out}\}$ *where* $t \lhd t_{out}$ **do**
        if $\mu(t, t_{out}) = 0$ **then**
          $\mathcal{R} = (\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$ $\quad //f_{\mathcal{R}}(x) = B,\ x \in [0, B]$
        else if *first*$(t) \neq \perp$ **then**
          $\mathcal{R}_t =$ *SetupVectors*$(A[t], t)$
          if $\mathcal{R}$ *is defined* **then**
            $\mathcal{R} = \mathit{Maximum}(\mathcal{R}, \mathcal{R}_t)$
          else
            $\mathcal{R} = \mathcal{R}_t$
  return $\mathcal{R}$

Algorithm 5: The function *SetupVectors* which is used in Algorithm 4.

SetupVectors($\mathcal{T},\ t$)

Input - $\mathcal{T} = (\vec{x}_{\mathcal{T}}, \vec{y}_{\mathcal{T}}),\ t \in \Delta$ such that $f_{\mathcal{T}}(e_{in}) = \max\left\{ \sum_{0<i\leq j} d_i \middle| \right.$

$\exists \gamma \in \mathit{Runs}(\mathcal{S})$ s.t. $\gamma = ({t_{in}}^{\bullet}, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} ({t_{in}}^{\bullet}, v'_{in}, e'_{in}) \xrightarrow{t_1} \ldots$

$\left. \xrightarrow{d_j} ({}^{\bullet}t, v'_j, e'_j) \xrightarrow{t} (t^{\bullet}, v_t, e_t) \right\}$

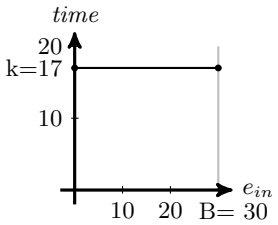
Output - $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$ such that $f_{\mathcal{R}}(e_{in}) = \max\left\{ e_{out} \middle| \right.$ $\exists \gamma \in \mathit{Runs}(\mathcal{S})$ s.t.

$\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} (t^{\bullet}_{in}, v'_{in}, e'_{in}) \xrightarrow{t_1} \ldots \xrightarrow{t_n} ({}^{\bullet}t_{out}, v_n, e_n)$

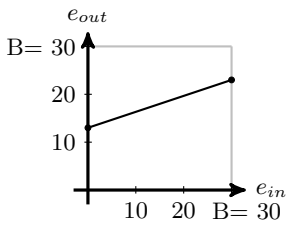
$\xrightarrow{d_n} ({}^{\bullet}t_{out}, v'_n, e'_n) \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out}),$ there exists an i,

$\left. 1 \leq i \leq n,\ \text{where } t_i = t,\ \text{and for all } j,\ i+1 \leq j \leq n, z_j \neq r \right\}$

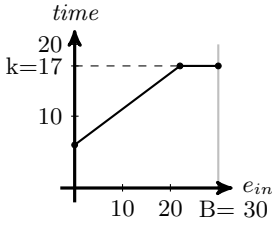
```
1  begin
2      x⃗[1] = first(t)
3      i = 2
4      forall the j, 1 ≤ j ≤ |x⃗_T| do
5          if x⃗[1] < x⃗_T[j] < last(t) then
6              x⃗[i] = x⃗_T[j]
7              i = i + 1
8      x⃗[i] = last(t)
9      forall the j, 1 ≤ j ≤ i do
10         m⃗[j] = B − (k − f_T(x⃗[j])) · μ(t, t_out)
11     if last(t) ≠ B then
12         i = i + 1
13         x⃗[i] = B
14         m⃗[i] = B
15     forall the j, 1 ≤ j ≤ i − 1 do
16         u⃗[j] = m⃗[j]
17     forall the j, 2 ≤ j ≤ i do
18         ℓ⃗[j] = m⃗[j]
19     return (x⃗, m⃗, u⃗, ℓ⃗)
```



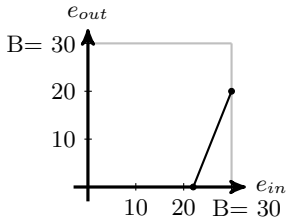
(a) The time function for t_{out} through t_1.



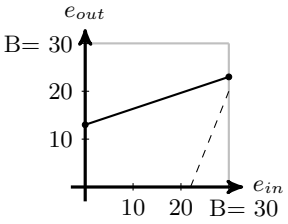
(b) The energy function for the paths through the transition t_1.



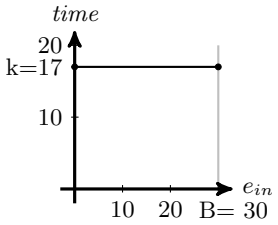
(c) The time function for t_{out} through t_2.



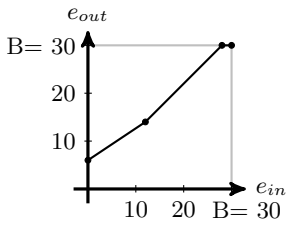
(d) The energy function for the paths through the transition t_2.



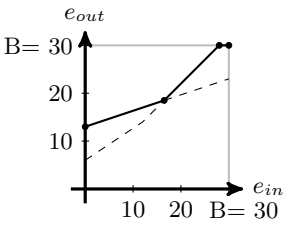
(e) Maximum between the energy function through t_1 and t_2.



(f) The time function for t_{out} through t_3.



(g) The energy function for the paths through the transition t_3.



(h) Maximum of the energy functions through t_1 and t_3.

Fig. 11: Calculation of an energy function.

Example 9. We can use Algorithm 4 to calculate the energy function representing the automaton segment shown in Fig. 8(a). The functions obtained in this process are shown in Fig. 11. Moreover recall that the time functions used by the algorithm were found in Example 8 and shown in Fig. 10.

Initially, we find out how much time we can spend by taking the path going from t_{in} through t_1 to t_{out}. Since we are recharged by t_1 and have a rate of 1 in q_{13}, we can always reach the value of the invariant with an energy bound of 30, as shown in Fig. 11(a). Of course delaying in q_{13} also means that the final energy e_{out} is going to be less that maximum. Depending on the energy we start with, e_{in}, it can be that we can wait in q_5 before taking t_1 and thereby increasing e_{out}, as shown in Fig. 11(b).

In the same way we can create the time and energy function for t_{out} when only considering t_2 as shown in Fig. 11(c) and 11(d). When taking the maximum between the two to find out which path leads to the highest energy result (Fig. 11(e)) we notice that the function considering only t_1 dominates during the entire domain.

The same calculations are made to find the time and energy function for t_3 in Fig. 11(f) and 11(g). Finally the maximum between the computed energy functions is found in Fig. 11(h).

We will now state the correctness of *EnergyFunction*. Note that we consider max of an empty set to return the bottom element, $\bot$.

Theorem 6. *Let $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$ be a snippet and let $\mathcal{R}$ be the energy function representation returned by EnergyFunction($\mathcal{S}$).*

Then if $\mathcal{R}$ is defined, we have that for all $e_{in} \in [0, B]$

$$f_{\mathcal{R}}(e_{in}) = \max \Big\{ e_{out} \Big| \exists \gamma \in \mathit{Runs}(\mathcal{S}) \text{ s.t. } \gamma = (t_{in}^{\bullet}, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} (t_{in}^{\bullet}, v'_{in}, e'_{in}) \xrightarrow{t_1} \dots \xrightarrow{d_j} ({}^{\bullet}t_{out}, v'_j, e'_j) \xrightarrow{t_{out}} (t_{out}^{\bullet}, v_{out} = [c = 0], e_{out}) \Big\}$$

If $\mathcal{R}$ is not defined, then there does not exist any run $\gamma = (t_{in}^{\bullet}, v_{in} = [c = 0], e_{in}) \to \dots \xrightarrow{t_{out}} (t_{out}^{\bullet}, v_{out} = [c = 0], e_{out})$ in $\mathit{Runs}(\mathcal{S})$.

The proof is shown in Appendix E on page 60.

Lemma 2. *Let $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$ be a snippet. Then EnergyFunction($\mathcal{S}$) runs in time $O(|\Delta|^3 + |\Delta|^2 \cdot |Q|)$.*

The proof can be found in Appendix F on page 64.

7 Energy Function Automaton

We will now create an abstraction over a one-clock recharge automaton $\mathcal{A}$ to find winning runs in the automaton. This abstraction will have transitions annotated with energy function representations, making it possible to represent a whole set of paths between a source and a target location with a single transition. Additionally, information on whether time is spent or not on the abstracted paths is represented on the transitions.

Definition 9 (Energy Function Automaton). *An energy function automaton is a tuple $\mathcal{E} = (Q, q_0, B, \Delta)$ where*

- *Q is a set of locations,*
- *$q_0 \in Q$ is the initial location,*
- *$B \in \mathbb{N}_0$ is the maximum energy level,*
- *$\Delta : Q \times \mathcal{F}_\mathcal{R} \times \{0, 1\} \times Q$ is a set of transitions, each with an energy function representation and an integer value which is 0 if no time is spent when taking the transition and 1 otherwise.*

Note that the time information on transitions can be seen as a boolean value. We call a transition $(q, \mathcal{R}, b, q')$ time-diverging if $b = 1$.

The semantics of an energy function automaton $\mathcal{E} = (Q, q_0, B, \Delta)$ is a labelled transition system $[\![\mathcal{E}]\!] = (S, s_0, \to)$, where the states are $S = \{(q, e) \mid q \in Q, e \in \mathbb{R}, 0 \leq e \leq B\}$ and the start state is $s_0 = (q_0, B)$. There is a transition $(q, e) \xrightarrow{\mathcal{R},b} (q', e')$ if there exists $(q, \mathcal{R}, b, q') \in \Delta$ such that $f_\mathcal{R}(e) = e'$.

A run ρ of an energy function automaton $\mathcal{E}$ is an infinite sequence of states and transitions starting from s_0: $\rho = s_0 \xrightarrow{\mathcal{R}_0,b_0} s_1 \xrightarrow{\mathcal{R}_1,b_1} \dots$.

We denote by $time(\rho)$ the amount of transitions on which time is spent along ρ.

$$time(\rho) = \sum_{i=0}^{\infty} b_i$$

A run ρ is winning if $time(\rho) = \infty$.

7.1 Construction of an Energy Function Automaton

In the following we will describe how we can create an energy function automaton $\mathcal{E}$ from a one-clock closed recharge automaton $\mathcal{A}$ in rfcNF, such that there exists a winning run in $\mathcal{E}$ iff there exists a winning run in $\mathcal{A}$.

Definition 10 (Start Normal Form). *Given a one-clock closed recharge automaton $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$, we say that $\mathcal{A}$ is in start normal form (sNF) if it is in rfcNF, $rate(q_0) = 0$ and there exists only one transition $t \in \Delta$, such that ${}^\bullet t = q_0$ and this is a reset transition.*

Without loss of generality assume that $\mathcal{A}$ is in sNF. Then we can use Algorithm 7 to construct an energy function automaton preserving the answer to the infinite run problem.

Note that an automaton in rfcNF can easily be transformed into one in sNF by simply adding a new start location with rate 0 and with a reset transition to the old start location.

Algorithm 6: Auxiliary function for *EnergyFunctionAutomaton* that may only add one transition.

ExtraTransition$(t, t', \Delta_{\mathcal{E}},$ *snippets*, *rech*$)$
Input - t, t' are reset transitions in Δ,
$\Delta_{\mathcal{E}}$ is a set of transitions,
snippets is the set of snippets to consider,
rech $\in \{$*true*, *false*$\}$ depending on whether the path has gone through a recharge transition yet
Output - $\Delta_{\mathcal{E}} \cup T$ where $T = \{(t^\bullet, (\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle), 1, t^\bullet)\}$ iff there exists a path from $t^\bullet$ and back through snippets in *snippets* where at least one includes a recharge transition and has the guard "$c = 0$" on t_{out} and the others do not include any recharge transition and have a *true* guard or "$c = 0$" on t_{out}, otherwise $T = \emptyset$,
snippets is the set of snippets not yet considered

```
1  begin
2      while ∃S = (t_in, t_out, Q_S, B, C, k, rate, Δ_S) ∈ snippets, where t' = t_in do
3          if ∃S' = (t'_in, t'_out, Q_S', B, C, k', rate, Δ_S') ∈ snippets, where
           t' = t'_in, RT(S') ≠ ∅, g_out = "c = 0" being g_out the guard of t'_out then
4              current= S'
5          else
6              current = S
7          snippets = snippets \{current}
8          if RT(current) = ∅ and either g_out = "c = 0" or g_out = true where g_out
           is the guard of t_out of the snippet current then
9              if t_out = t and rech= true then
10                 Δ_E = Δ_E ∪ {(t•, (⟨0, B⟩, ⟨B, B⟩, ⟨B⟩, ⟨B⟩), 1, t•)}
11             else
12                 (Δ_E, snippets) = ExtraTransition(t, t_out, Δ_E, snippets, rech)
13         else if g_out = "c = 0" and RT(current) ≠ ∅ then
14             if t_out = t then
15                 Δ_E = Δ_E ∪ {(t•, (⟨0, B⟩, ⟨B, B⟩, ⟨B⟩, ⟨B⟩), 1, t•)}
16             else
17                 (Δ_E, snippets) = ExtraTransition(t, t_out, Δ_E, snippets, true)
18     return (Δ_E, snippets)
```

Algorithm 7 takes as input a recharge automaton $\mathcal{A} = (Q, q_0, B, C, I, rate, \Delta)$ in sNF. Initially it finds all transitions with a reset in the automaton $\mathcal{A}$. The set of locations of the energy function automaton $Q_{\mathcal{E}}$ consists of all the locations that these transitions go to.

Algorithm 7: The algorithm creates an energy function automaton based on a recharge automaton in sNF.

EnergyFunctionAutomaton($\mathcal{A}$)
Input - $\mathcal{A} = (Q, q_0, B, C, I, \mathit{rate}, \Delta)$ in sNF
Output - $\mathcal{E} = (Q_\mathcal{E}, q_\mathcal{E}, B, \Delta_\mathcal{E})$ such that there exists a winning run in $\mathcal{E}$ iff there exists a winning run in $\mathcal{A}$

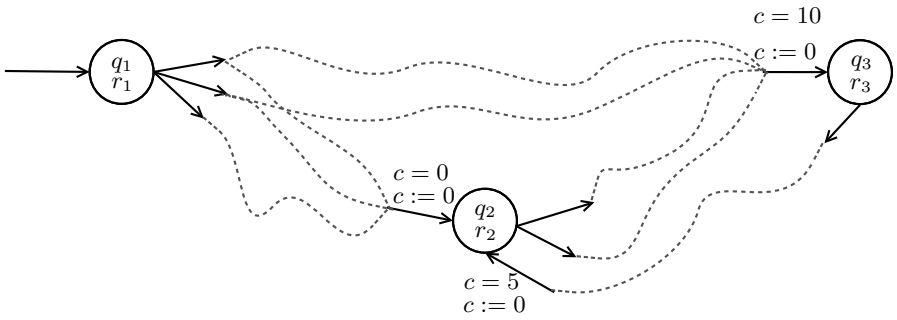
begin
  Reset $= \{t \mid t = (q, g, z, \{c\}, q'), t \in \Delta\}$
  $Q_\mathcal{E} = \{q' \mid (q, g, z, \{c\}, q') \in \text{Reset}\}$
  $q_\mathcal{E} = q$ where $\exists (q_0, g, z, \{c\}, q) \in \Delta$
  $\Delta_\mathcal{E} = \emptyset$
  snippets $= \emptyset$
  forall the *pairs of transitions* $(t_{in},\ t_{out}) \in \mathit{Reset} \times \mathit{Reset}$ **do**
    if $\mathit{Paths}(t_{in}, t_{out}) \neq \emptyset$ **then**
      $Q_\mathcal{S} = \{q \mid q \in \mathit{loc}(\pi)$ for some path $\pi \in \mathit{Paths}(t_{in}, t_{out})\}$
      $k = a$ where $I(t_{in}^\bullet) =$ "$c \leq a$"
      $\Delta_\mathcal{S} = \{t \mid t \in \mathit{trans}(\pi)$ for some $\pi \in \mathit{Paths}(t_{in}, t_{out})\}$
      $\mathcal{S} = (t_{in}, t_{out}, Q_\mathcal{S}, B, C, k, \mathit{rate}, \Delta_\mathcal{S})$
      snippets $=$ *snippets* $\cup \{\mathcal{S}\}$
  forall the $\mathcal{S} \in$ *snippets, where* $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$ **do**
    $\mathcal{R} = \mathit{EnergyFunction}(\mathcal{S})$
    if $\mathcal{R}$ *is defined* **then**
      if $g_{out} =$ "$c = 0$" *or* $(g_{out} = \mathit{true}, \mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$ *and* $\mu(t_{in}, t_{out}) > 0)$ *where* g_{out} *is the guard of* t_{out} **then**
        $b = 0$
      else
        $b = 1$
      $\Delta_\mathcal{E} = \Delta_\mathcal{E} \cup \{(t_{in}^\bullet, \mathcal{R}, b, t_{out}^\bullet)\}$
      if $(g_{out} = \mathit{true}, \mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$ *and* $\mu(t_{in}, t_{out}) > 0)$ *where* g_{out} *is the guard of* t_{out} **then**
        $(\Delta_\mathcal{E},\ var) = \mathit{ExtraTransition}(t_{in}, t_{out}, \Delta_\mathcal{E},\ \mathit{snippets},\ \mathit{false})$
  return $(Q_\mathcal{E}, q_\mathcal{E}, B, \Delta_\mathcal{E})$

Afterwards, a snippet is constructed for each pair of reset transitions (t_{in}, t_{out}), where $Paths(t_{in}, t_{out}) \neq \emptyset$, and *EnergyFunction* is used to calculate a representation for the energy function abstracting the runs between them. If the energy function is defined for some values, a transition between $t_{in}^{\bullet}$ and $t_{out}^{\bullet}$ is added to $\Delta_{\mathcal{E}}$ with a value b depending on whether we are allowed to spend time before taking t_{out} or not.

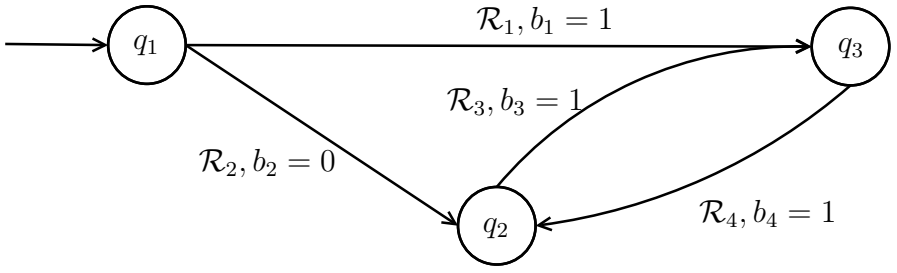
The function *ExtraTransition* is called in the special case that a snippet, where time is allowed to pass, does not include recharge transitions. This function explores the graph by recursive calls to itself to check whether there exists a path from this snippet back to itself, through snippets of this same type or where time is not allowed to pass. If the path includes a snippet with recharge transitions, but not allowing time to pass, an infinite delay can indeed be performed by repeating the path. We therefore add an extra transition with $b = 1$ and $\mathcal{R} = (\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$.

Finally, the energy function automaton is returned.

As an example, observe Fig. 12(a) where a recharge automaton $\mathcal{A}$ is sketched. On Fig. 12(b) we can observe the energy function automaton that results from running *EnergyFunctionAutomaton*($\mathcal{A}$).



(a) Recharge automaton $\mathcal{A}$.



(b) Energy function automaton $\mathcal{E}$ returned by *EnergyFunctionAutomaton*($\mathcal{A}$).

Fig. 12: Construction of an energy function automaton from a recharge automaton.

Theorem 7. *Let $\mathcal{A}$ be a one-clock closed recharge automaton in sNF. Then the algorithm EnergyFunctionAutomaton($\mathcal{A}$) runs in time $O(|\Delta|^5 + |\Delta|^4 \cdot |Q|)$ and constructs an energy function automaton $\mathcal{E}$ such that there exists a winning run of $\mathcal{A}$ iff there exists a winning run of $\mathcal{E}$.*

The proof of the theorem can be found in Appendix G on page 67. The correctness part of the proof consists not only of proving that the existence of an infinite run is preserved in each automaton, but also of proving that time divergence is preserved in the corresponding runs.

8 Winning Runs in Energy Function Automata

In this section we will investigate how to use the energy function automaton to get an answer to the infinite run problem. To do this, we will show that it is enough to inspect all cycles with at most $2 \cdot |Q| + 1$ locations when looking for a winning run.

Let $\mathcal{E} = (Q, q_0, B, \Delta)$ be an energy function automaton and $\pi = q_1 \xrightarrow{\mathcal{R}_1, b_1} q_2 \xrightarrow{\mathcal{R}_2, b_2} \dots \xrightarrow{\mathcal{R}_{n-1}, b_{n-1}} q_n$ be a path in $\mathcal{E}$. We say that π is a *cycle* if $q_n = q_1$. Moreover we say that π is a *short cycle*, if it has length of at most $2|Q| + 1$ locations.

Finally, a short cycle π is a *simple cycle* if the first location in π appears exactly twice and every other location appears exactly once in π.

Let $\pi = q_1 \xrightarrow{\mathcal{R}_1, b_1} q_2 \xrightarrow{\mathcal{R}_2, b_2} \dots \xrightarrow{\mathcal{R}_{n-1}, b_{n-1}} q_n$ be a cycle, let $\mathcal{R}_\pi$ be the representation of the function $(\dots((f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2}) \circ f_{\mathcal{R}_3}) \circ \dots) \circ f_{\mathcal{R}_{n-1}}$ and let the infimum of an empty set be ∞. We then define e_π as the least amount of energy required to be able to repeat the cycle infinitely.

$$e_\pi = \inf \Bigg(\{e \mid f_{\mathcal{R}_\pi}(e) \geq e\} \cup \{\vec{x}[i] \mid \vec{u}[i] = \vec{x}[i], \vec{\ell}[i+1] \neq \vec{u}[i], \mathcal{R}_\pi = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell}), 1 \leq i < |\vec{x}|\} \Bigg)$$

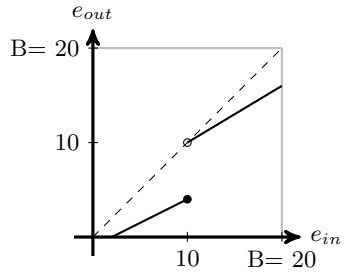


Fig. 13: An energy function with an undefined point on the line $f(e_{in}) = e_{in}$.

Whenever $f_{\mathcal{R}_\pi}(e) \geq e$, we know that for all $e' \geq e$ it is the case that $f_{\mathcal{R}_\pi}(e') \geq e$, since the function is non-decreasing. Note however that it might be the case that we have a discontinuous point on the diagonal line $f(e_{in}) = e_{in}$, meaning that we by continuous iterations of the function for values above the point can get arbitrarily close to it without ever reaching it. This case is illustrated in Fig. 13. Because of this, all function values on the diagonal line are added when determining e_π. Equally a point of discontinuity could result in values arbitrarily close to and higher than the discontinuous point being over the line $f(e_{in}) = e_{in}$ while the function value at the point itself is under. Therefore we are looking for an infimum value.

To be able to repeat the cycle infinitely we need to reach it with an energy higher than e_π, though a value equal to e_π is enough when $f_{\mathcal{R}_\pi}(e_\pi) \geq e_\pi$. If it is not possible to reach π with an energy of at least e_π, only a finite amount of iterations can be performed before running out of energy. This is the case since the function then can only decrease and only when the function is nearing an intersection with the diagonal line we can have an infinite series of iterations of decreasing function values.

We say that a cycle π is winning if there is a time-diverging transition in π and either π is reachable with an energy level $e > e_\pi$ or $f_{\mathcal{R}_\pi}(e_\pi) \geq e_\pi$ and π is reachable with an energy level $e \geq e_\pi$.

When searching for a winning cycle, it is not sufficient to look for a simple cycle. The intuition is that cycles without time-divergent transitions may increase the energy level to the maximum after just one repetition, though alone they do not constitute a solution to the problem. Adding these cycles along a simple cycle which includes a time-divergent transition, could decrease the energy required to perform the cycle infinitely, and thereby provide a solution to the problem. Though $2|Q| + 1$ locations are enough, since the path can include all locations twice, once as a part of a cycle without time-divergent transitions and once as part of a cycle where time does diverge.

Lemma 3. *Let $\mathcal{E} = (Q, q_0, B, \Delta)$ be an energy function automaton and π be a winning cycle in $\mathcal{E}$. Then there exists a short cycle π' in $\mathcal{E}$ which is also winning.*

Proof. (Sketch) Assume that there is no short cycle π' in $\mathcal{E}$ which is winning. Let π be the smallest winning cycle in $\mathcal{E}$. We will now show that it is possible to reduce its length to obtain a short winning cycle π'.

Since π is not short, either there are at least two locations that are repeated more than 2 times or one location that is repeated more than 3 times. Consider the first 3 repetitions of a location q, where q can only be the first location in π if this is repeated at least 4 times along π. Let $\pi' = q \xrightarrow{\mathcal{R}_0,b_0} q_1 \xrightarrow{\mathcal{R}_1,b_1} \dots \xrightarrow{\mathcal{R}_m,b_m} q \xrightarrow{\mathcal{R}_{m+1},b_{m+1}} \dots \xrightarrow{\mathcal{R}_{n-1},b_{n-1}} q_n \xrightarrow{\mathcal{R}_n,b_n} q$ be a subpath of π. Moreover, let the path π' be composed of $\pi'_1 = q \xrightarrow{\mathcal{R}_0,b_0} q_1 \xrightarrow{\mathcal{R}_1,b_1} \dots \xrightarrow{\mathcal{R}_{m-1},b_{m-1}} q_m \xrightarrow{\mathcal{R}_m,b_m} q$ and $\pi'_2 = q \xrightarrow{\mathcal{R}_{m+1},b_{m+1}} q_{m+2} \xrightarrow{\mathcal{R}_{m+2},b_{m+2}} \dots \xrightarrow{\mathcal{R}_{n-1},b_{n-1}} q_n \xrightarrow{\mathcal{R}_n,b_n} q$. Then we have the following cases:

1. For all $i,\ 0 \leq i \leq n$, $b_i = 0$.

2. There exists an $i,\ 0 \leq i \leq n$, such that $b_i = 1$.
   (a) It is possible to reach π' with energy level $e > e_{\pi'}$ or $f_{\mathcal{R}_{\pi'}}(e_{\pi'}) \geq e_{\pi'}$ and π' is reachable with an energy level $e \geq e_{\pi'}$.
   (b) It is not possible to reach π' with energy level $e > e_{\pi'}$ or $f_{\mathcal{R}_{\pi'}}(e_{\pi'}) \geq e_{\pi'}$ and π' is not reachable with an energy level $e \geq e_{\pi'}$.
      i. For all $i,\ 0 \leq i \leq m$, $b_i = 0$
      ii. For all $i,\ m+1 \leq i \leq n$, $b_i = 0$.
      iii. There exists an $i,\ 0 \leq i \leq m$, such that $b_i = 1$ and there exists a $j,\ m+1 \leq j \leq n$, $b_j = 1$.

If all transitions have $b = 0$ (case 1), then either all the functions on the transitions of π' are on the form $f_{\mathcal{R}_i}(e_{in}) = e_{in}$ for all $e_{in} \in [0, B]$ and we can remove π' from π, or there is at least one function on the form $f_{\mathcal{R}_i}(e_{in}) = B$ for all $e_{in} \in [0, B]$. In the latter case we can remove either of the paths π'_1 or π'_2 as long as we keep one transition with a function that always returns B. Clearly in this case the removed path would not have changed the energy level.

Now consider case 2 where there is at least one transition in π' with $b = 1$.

If it is possible to reach π' along the path in π with an energy level $e > e_{\pi'}$ or $f_{\mathcal{R}_{\pi'}}(e_{\pi'}) \geq e_{\pi'}$ and π' is reachable with an energy level $e \geq e_{\pi'}$ (case 2a), then we can replace π with π' as π' is also winning.

Otherwise, we have case 2b. Consider first subcase 2(b)i. If there is no time-diverging transition $(q_i, \mathcal{R}_i, 1, q_{i+1})$ in π'_1, where $0 \leq i \leq m$, then $f_{\mathcal{R}_i}(e_{in}) = e_{in}$ for all $e_{in} \in [0, B]$ and for all $i,\ 0 \leq i \leq m$. Thus, since all functions in π'_1 do not change the energy, we can remove π'_1 from π.

Note that it cannot be the case that there exists an $i,\ 1 \leq i \leq m$ where $f_{\mathcal{R}_i}(e_{in}) = B$ for all $e_{in} \in [0, B]$. If this was the case then the function $f_{\mathcal{R}_{\pi'_2}}$ would be undefined at B. If it were not undefined, it would be possible to reach π' with energy level $e \geq e_{\pi'}$, being $e_{\pi'} = 0$. Since π'_2 leads to an undefined function, π would not be winning, resulting in a contradiction.

The same considerations hold for π'_2 in subcase 2(b)ii.

Now we consider subcase 2(b)iii, where we know that both π'_1 and π'_2 have at least one time-diverging transition. If π'_1 is a winning cycle it can be repeated infinitely and we can consider this cycle instead of π. If it is not winning, we can remove π'_1, as it only decreases the energy available.

In all of the possible cases we have reduced the number of repetitions of a location with at least one. Therefore by repeating this process it is possible to obtain a short cycle π' that is winning, which is a contradiction to the initial assumption. □

This property is used by Algorithm 9 to verify whether there exists a winning run of a given energy function automaton $\mathcal{E}$. To begin with, the algorithm finds all short cycles in $\mathcal{E}$. Then, for each cycle it is possible to calculate the minimal energy level that is required to be able to take the cycle infinitely, while making sure that time diverges. This computation is performed in *MinimalEnergy* which is called for each cycle. The algorithm takes a cycle π as input and calls *Composition* once for each transition in the cycle to make an energy function

for the entire cycle. The algorithm then finds and returns the value e_π needed to be able to go around the cycle from its first location, and a boolean value that is true only if an energy level of exactly e_π is enough to do so. Note that the calculation of e_π can be performed in linear time in the size of the representation of the energy function computed, by inspecting the points in the function representation.

Finally, the algorithm calls *RelaxTransitions* which calculates the energy that we can reach each location with by updating the energy $2|Q|-1$ times, each time based on the calculated values of neighbour locations. It performs the update $2|Q|-1$ times, since even the energy value for the locations furthest away from the start location will then be updated at least two times. This is necessary since taking a cycle without time-divergent transitions would increase the energy that we can reach a location with, without it being part of a solution. Again using the same cycle twice would not be desirable, as the maximal energy is reached after just one iteration.

If the computed energy for a location is more than the required minimal energy (or equal to, in the case where $f_{\mathcal{R}_\pi}(e_\pi) \geq e_\pi$), we have found a short cycle that can be reached with enough energy to be repeated infinitely, and as such an infinite winning run of the energy function automaton.

Algorithm 8: The algorithm calculates the minimal energy necessary to be able to repeat the cycle π infinitely.

MinimalEnergy(π)
begin
  if $\exists q_i \xrightarrow{\mathcal{R}_i,1} q_{i+1} \in \pi$ *for some* $i, 1 \leq i \leq n-1$ **then**
    $\mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$
    forall the $i, 1 \leq i \leq n-1$ **do**
      $\mathcal{R} = \mathit{Composition}(\mathcal{R}, \mathcal{R}_i)$ where $q_i \xrightarrow{\mathcal{R}_i,b_i} q_{i+1} \in \pi$
    $e_\pi = \inf\Big(\{e \mid f_{\mathcal{R}_\pi}(e) \geq e\} \cup \{\vec{x}[i] \mid \vec{u}[i] = \vec{x}[i], \vec{\ell}[i+1] \neq \vec{u}[i],$
    $\mathcal{R}_\pi = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell}), 1 \leq i < |\vec{x}|\}\Big)$
    if $f_{\mathcal{R}}(e_\pi) \geq e_\pi$ **then**
      return (e_π, true)
    else
      return (e_π, false)
  return (∞, false)

Algorithm 9: Algorithm to find out whether there exists a winning run in a energy function automaton $\mathcal{E}$.

RelaxTransitions$(C, \mathcal{E})$

begin
  forall the $i,\ 1 \leq i \leq 2|Q| - 1$ **do**
    forall the *transitions* $(q, \mathcal{R}, b, q') \in \Delta$ **do**
      if $f_{\mathcal{R}}(C(q)) \neq \perp$ *and* $C(q') < f_{\mathcal{R}}(C(q))$ **then**
        $C(q') = f_{\mathcal{R}}(C(q))$
  return C

ExistsWinningRun$(\mathcal{E})$
Input - $\mathcal{E} = (Q, q_0, B, \Delta)$
Output - *true* iff there exists a winning run in $\mathcal{E}$, otherwise *false*

begin
  forall the $q \in Q$ **do**
    $E(q) = \infty$
    $D(q) = false$
    $C(q) = -\infty$
  $C(q_0) = B$
  Let *Cycles* be the set of all short cycles in $\mathcal{E}$
  foreach *cycle* $\pi = q_1 \xrightarrow{\mathcal{R}_1, b_1} q_2 \xrightarrow{\mathcal{R}_2, b_2} \ldots \xrightarrow{\mathcal{R}_{n-1}, b_{n-1}} q_n \in$ *Cycles* **do**
    $(e_\pi, D_\pi) =$ *MinimalEnergy*(π)
    if $e_\pi < E(q_1)$ *or* $(e_\pi = E(q_1)$ *and* $D_\pi = true)$ **then**
      $D(q_1) = D_\pi$
    $E(q_1) = \min\{E(q_1), e_\pi\}$
  $C =$ *RelaxTransitions*$(C, \mathcal{E})$
  forall the $q \in Q$ **do**
    if $C(q) > E(q)$ *or* $(C(q) = E(q)$ *and* $D(q) = true)$ **then**
      return *true*
  return *false*

Theorem 8. *Let $\mathcal{E} = (Q, q_0, B, \Delta)$ be an energy function automaton and let n be the number of short cycles in $\mathcal{E}$. Then ExistsWinningRun($\mathcal{E}$) runs in time $O(n \cdot |Q| \cdot |\Delta|^2 + FindCycles)$, where $FindCycles$ represents the complexity of finding all short cycles. Moreover, ExistsWinningRun($\mathcal{E}$) returns true iff there exists a winning run in $\mathcal{E}$.*

Proof. We start by proving the complexity of the algorithm.

The initialization runs in $O(|Q|)$. To find all short cycles, we have the complexity $FindCycles$. Then we go through all short cycles and call *MinimalEnergy*. This algorithm runs in $O(|Q| \cdot |\Delta|^2)$ as each energy function can at most have size $|\Delta|^2$ and we call *Composition* $|Q|$ times, every time adding at most the amount of vector values in the function representations. Note that computing e_π can be done in linear time in the size of this function representation, by inspecting the values in the representation. Since the function is called for each cycle we have a complexity of $O(n \cdot |Q| \cdot |\Delta|^2)$ for this part of the algorithm.

Finally, the function *RelaxTransitions* is called once, with a complexity of $O(|Q| \cdot |\Delta|)$, and afterwards $|Q|$ comparisons are made.

The total complexity is thus either dominated by the computation of energy functions or the complexity of the algorithm finding the cycles. The algorithm then has complexity $O(n \cdot |Q| \cdot |\Delta|^2 + FindCycles)$.

Correctness follows from Lemma 3 and the fact that the path to the cycle itself has at most $2|Q|$ locations. This is the case since a cycle in which all transitions are not time-divergent may give a higher energy, though we do not benefit from repeating them more than once. If the path is longer than $2|Q|$, there is either a cycle which has a negative effect on the energy or a cycle preserving the energy. Both types of cycles can be removed, while still having a winning run. □

Corollary 1. *The infinite run problem is in NP.*

Proof. The path used in Theorem 8 to construct a winning run has polynomial size. It is therefore possible to guess a short cycle and a path to its first location and check whether it can be used to construct a winning run in nondeterministic polynomial time. □

8.1 Flat Recharge Automata

Inspired by [15], if we restrict to flat recharge automata, we can show that the solution to the infinite run problem can then be found in polynomial time.

Definition 11 (Flat Recharge Automaton). *We say that a one-clock closed recharge automaton $\mathcal{A}$ is flat if $\mathcal{A}$ is in rfcNF and each location belongs to at most one cycle.*

Note that when constructing the energy function automaton, we may only add one cycle per location, represented by a self-loop. We know that all short cycles in $\mathcal{E}$ that are not simple must include a transition added by *ExtraTransition*.

This transition is itself a simple cycle requiring a minimal energy of 0. Therefore for an energy function automaton constructed from a flat automaton, it suffices to consider simple cycles, when searching for a winning run. Since each location belongs to at most one simple cycle consisting of more than one transition, it is possible to use Depth-First Search to find all the simple cycles in the energy function automaton.

Finally, we can state the following result.

Corollary 2. *Given a flat recharge automaton $\mathcal{A}$, the infinite run problem can be solved in time $O(|\Delta|^4 \cdot (|\Delta| + |Q|) + |Q| \cdot (|\Delta| + |Q|)^3)$.*

Proof. Follows from Theorem 7 and Theorem 8. Note that the constructed energy function automaton has at most $|Q|$ locations and $|\Delta| + |Q|$ transitions. □

9 Conclusion

We presented recharge automata as a formalism to model resource consumption in systems, where the resource can be recharged instantaneously to its maximal level. We considered the problem of deciding whether there exists an infinite non-zeno run, where the energy level never drops below zero in a recharge automaton with just one clock.

To find a solution to the problem, we used energy function automata, an abstraction of recharge automata where transitions are annotated with time information and functions describing the change in energy caused by taking the transition. We showed how to construct an energy function automaton from a recharge automaton in polynomial time while preserving the answer to the problem considered.

The general problem of finding a winning run in an energy function automaton was found to be in NP. We have shown, though, how it is possible to solve in polynomial time the infinite run problem for the subclass of flat recharge automata, where each location is only allowed to be part of one cycle.

Observe that the process of finding a winning run in the automaton would be shorter if we did not require it to be time-diverging or if we assumed that all infinite runs in the given recharge automaton were non-zeno. Then the used normal form would be simplified and thereby also its construction. Moreover, the construction of an energy function automaton and the problem of finding a winning run in this model would only require considering simple cycles.

9.1 Future work

We have not succeeded in finding neither a polynomial time algorithm solving the general problem nor an NP-hardness proof. Therefore the exact complexity of the problem is still unknown. The maximal energy level can be seen as a soft bound, and as such it cannot be exceeded, which makes it difficult to prove NP-hardness of the problem.

A dual problem to the one presented in this paper is to determine whether a given capacity for the resource allows to perform all runs of the automaton. We believe that a strategy similar to the one presented in this paper can be used, with the purpose of finding a counterexample, that is a run that cannot be performed with the given capacity. Here, energy functions would be used to minimize instead of maximize the energy that can be achieved on paths of a given automaton segment (snippet). This strategy would result in performing delays in the location with the highest rate just before taking a reset transition, thereby minimizing the energy and the run's likelihood of being able to satisfy the guard of the transition.

This strategy can also be used to determine the least capacity required to ensure that we can perform all runs without running out of energy. As before it would require observing the most energy demanding run and make sure that the energy capacity is high enough to support it, if possible. Similarly, we can determine the least resource capacity, such that there exists a winning run in the given automaton. This could be done by using an energy-maximizing strategy like the one used in this paper and then find the least energy demanding run to determine the lowest capacity needed.

Going a bit further along these lines, we could observe the problem from a game perspective, meaning that some locations would be out of our control. In this case, we would then require a strategy to make sure to always steer the run into a direction in which time will progress and we will always have enough energy to make a next move no matter what happens.

Other interesting problems for recharge automata could include observing time limited runs, such that we no longer require the runs to be infinite. Here, we know that the problem is NP-hard even for recharge automata without cycles (see Appendix H on page 72). In general, problems concerning time limited runs would require another strategy than the one used in this paper, since all time information is currently abstracted away.

References

1. Intergovernmental panel on climate change (ipcc), 2007, `http://www.ipcc.ch/`.
2. Copenhagen climate change conference - december 2009, `http://unfccc.int/meetings/copenhagen_dec_2009/meeting/6295.php`.
3. Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi. R-automata. In *Proceedings of CONCUR'08, Toronto, Canada*, volume 5201 of *Lecture Notes in Computer Science*, pages 67–81. Springer-Verlag, 2008.
4. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
5. Rajeev Alur, Salvatore La Torre, and George Pappas. Optimal paths in weighted timed automata. In Maria Di Benedetto and Alberto Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer Berlin / Heidelberg, 2001.
6. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed

automata. In Maria Di Benedetto and Alberto Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer Berlin / Heidelberg, 2001.

7. Patricia Bouyer, Ed Brinksma, and Kim Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32:3–23, 2008.
8. Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, HSCC '10, pages 61–70, New York, NY, USA, 2010. ACM.
9. Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS '08, pages 33–47, Berlin, Heidelberg, 2008. Springer-Verlag.
10. Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Lower-bound constrained runs in weighted timed automata. In *Proceedings of QEST'12*, 2012.
11. Climate Consortium Denmark. State of green - energy efficiency, 2011, `http://www.stateofgreen.com/en/Energy-Efficiency`.
12. Daniel Ejsing-Duun and Lisa Fontani. Infinite runs in recharge automata.
13. Guillermo Escrivá-Escrivá, Carlos Álvarez Bel, Carlos Roldán-Blay, and Manuel Alcázar-Ortega. New artificial neural network prediction method for electrical consumption forecasting based on building end-uses. *Energy and Buildings*, 43(11):3112 – 3119, 2011.
14. Uli Fahrenberg and Kim G. Larsen. Discount-optimal infinite runs in priced timed automata. *Electronic Notes in Theoretical Computer Science*, 239:179 – 191, 2009. Joint Proceedings of the 8th, 9th, and 10th International Workshops on Verification of Infinite-State Systems (INFINITY 2006, 2007, 2008).
15. Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In *FOSSACS*, pages 229–244, 2011.
16. Kim Larsen, Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As cheap as possible: Effcient cost-optimal reachability for priced timed automata. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer Berlin / Heidelberg, 2001.
17. Hafiz Fahad Sheikh, Hengxing Tan, Ishfaq Ahmad, Sanjay Ranka, and Phanisekhar Bv. Energy- and performance-aware scheduling of tasks on parallel and distributed systems. *J. Emerg. Technol. Comput. Syst.*, 8(4):32:1–32:37, November 2012.
18. S.A. Tassou, G. De-Lille, and Y.T. Ge. Food transport refrigeration - approaches to reduce energy consumption and environmental impacts of road transport. *Applied Thermal Engineering*, 29(8-9):1467 – 1477, 2009.
19. G. Wood and M. Newborough. Dynamic energy-consumption indicators for domestic appliances: environment, behaviour and design. *Energy and Buildings*, 35(8):821 – 841, 2003.
20. Qingbo Zhu, Francis M. David, Christo F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. Reducing energy consumption of disk storage using power-aware cache management. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA '04, pages 118–, Washington DC, USA, 2004. IEEE Computer Society.

A Correctness of *Maximum*

Theorem 1. *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be representations of energy functions and let $\mathcal{R} = Maximum(\mathcal{R}_1, \mathcal{R}_2)$. Then $f_{\mathcal{R}} = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})$ and $\mathcal{R}$ can be constructed in time $O(|\mathcal{R}_1| + |\mathcal{R}_2|)$.*

Proof. It can be shown by exhaustive case analysis that $\mathcal{R}$ indeed is a representation of an energy function. Moreover, the size of the set S_{inter} is at most $|\mathcal{R}_1| + |\mathcal{R}_2| - 1$, since we can have at most $|\vec{x}_1| + |\vec{x}_2| - 1$ values delimiting intervals. In each interval the energy functions are defined as linear functions, and thus there can be at most one point in the set S_{inter} for each interval.

We now prove that $f_{\mathcal{R}} = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})$. We have different cases depending on which x we calculate the function for:

- $x < \vec{x}_1[1]$
  - $x < \vec{x}_2[1]$ (1)
  - $x \geq \vec{x}_2[1]$
    - $x \in elem(\vec{x}_2)$ (2)
    - $x \notin elem(\vec{x}_2)$ (3)
- $x \geq \vec{x}_1[1]$
  - $x < \vec{x}_2[1]$
    - $x \in elem(\vec{x}_1)$ (4)
    - $x \notin elem(\vec{x}_1)$ (5)
  - $x \geq \vec{x}_2[1]$
    - $x \in elem(\vec{x}_1)$
      - $x \in elem(\vec{x}_2)$ (6)
      - $x \notin elem(\vec{x}_2)$ (7)
    - $x \notin elem(\vec{x}_1)$
      - $x \in elem(\vec{x}_2)$ (8)
      - $x \notin elem(\vec{x}_2)$ (see note in case 9 below)
        - $\exists s \in S_{inter}$, s.t. $a < s < b$
          - $x < s$ (9)
          - $x = s$ (10)
          - $x > s$ (11)
        - $\nexists s \in S_{inter}$, s.t. $a < s < b$ (12)

Case 1
In this case $x < \vec{x}_1[1]$ and $x < \vec{x}_2[1]$.

We first calculate the maximum between the two functions. We know that $x < \vec{x}_1[1]$ and $x < \vec{x}_2[1]$ thus we have the following:

$$f_{\mathcal{R}_1}(x) = \bot \qquad f_{\mathcal{R}_2}(x) = \bot$$

Therefore in this case $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = \bot$.

For calculating $f_{\mathcal{R}}(x)$ we need the value $\vec{x}[1]$. Since this is defined as the lowest number in the set S_{max}, which does not include any value lower than the lowest of both $\vec{x}_1$ and $\vec{x}_2$, we have that $x < \vec{x}[1]$, and thus $f_{\mathcal{R}}(x) = \bot$.

Case 2
In this case $x < \vec{x}_1[1]$, $x \geq \vec{x}_2[1]$ and $x \in \mathit{elem}(\vec{x}_2)$.

Since $x \in \mathit{elem}(\vec{x}_2)$, we know that there must exist an $i,\ 1 \leq i \leq |\vec{x}_2|$ s.t. $x = \vec{x}_2[i]$. Since S_{max} contains all elements from $\vec{x}_2$, we also know that there exists a $j,\ 1 \leq j \leq |\vec{x}|$ s.t. $\vec{x}[j] = \vec{x}_2[i]$.

We then have that $f_{\mathcal{R}}(x) = \vec{m}[j]$, which, by Equation 1, is defined as:

$$\vec{m}[j] = \max\{f_{\mathcal{R}_1}(\vec{x}[j]), f_{\mathcal{R}_2}(\vec{x}[j])\} = \max\{f_{\mathcal{R}_1}(x), f_{\mathcal{R}_2}(x)\}$$

This is the definition of $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x)$, thus proving this case.

Case 3
In this case $x < \vec{x}_1[1]$, $x \geq \vec{x}_2[1]$ and $x \notin \mathit{elem}(\vec{x}_2)$.

We start by calculating the maximum between the two energy functions. We know that $x < \vec{x}_1[1]$, thus $f_{\mathcal{R}_1}(x) = \bot$. Since $x \geq \vec{x}_2[1]$ and $x \notin \mathit{elem}(\vec{x}_2)$, we know that there exists an index $i,\ 1 \leq i \leq |\vec{x}_2| - 1$ s.t. $\vec{x}_2[i] < x < \vec{x}_2[i+1]$. The value of $f_{\mathcal{R}_2}$ at x is then:

$$f_{\mathcal{R}_2}(x) = \vec{u}_2[i] + (\vec{\ell}_2[i+1] - \vec{u}_2[i])\frac{x - \vec{x}_2[i]}{\vec{x}_2[i+1] - \vec{x}_2[i]}$$

We can then calculate the maximum between the two energy functions:

$$\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(x)$$

We will now find the value of $f_{\mathcal{R}}(x)$. As before, since S_{max} contains all elements from $\vec{x}_2$, we know that there exists an index $j,\ 1 \leq j \leq |\vec{x}| - 1$, s.t. $\vec{x}[j] = \vec{x}_2[i]$.

Also, since $x < \vec{x}_1[1]$, there is no index $k,\ 1 \leq k \leq |\vec{x}|$, s.t. $\vec{x}[k] = x$ and thus $\vec{x}[j] < x < \vec{x}[j+1]$. The value $f_{\mathcal{R}}(x)$ is then defined as:

$$f_{\mathcal{R}}(x) = \vec{u}[j] + (\vec{\ell}[j+1] - \vec{u}[j])\frac{x - \vec{x}[j]}{\vec{x}[j+1] - \vec{x}[j]}$$

To calculate this value we need therefore the value of $\vec{u}[j]$ and $\vec{\ell}[j+1]$. We know that $\vec{x}[j] = \vec{x}_2[i]$ and $\vec{x}[j] < \vec{x}_1[1]$. Therefore we use Equation 2b for calculating the value of $\vec{u}[j]$.

$$\vec{u}[j] = \max\{f_{\mathcal{R}_1}(\vec{x}[j]), \vec{u}_2[i]\} = \vec{u}_2[i]$$

To calculate the value of $\vec{\ell}[j+1]$ we now have three subcases, depending on which vector the value $\vec{x}[j+1]$ was taken from:

1. $\vec{x}[j+1] = \vec{x}_1[1] \neq \vec{x}_2[i+1]$
2. $\vec{x}[j+1] = \vec{x}_2[i+1] \neq \vec{x}_1[1]$
3. $\vec{x}[j+1] = \vec{x}_1[1] = \vec{x}_2[i+1]$

In case 1 we use Equation 3a.

$$\begin{aligned}\vec{\ell}[j+1] &= \max\{\vec{\ell}_1[1], f_{\mathcal{R}_2}(\vec{x}[j+1])\} = \max\{\bot, f_{\mathcal{R}_2}(\vec{x}[j+1])\} \\ &= \vec{u}_2[i] + (\vec{\ell}_2[i+1] - \vec{u}_2[i])\frac{\vec{x}[j+1] - \vec{x}_2[i]}{\vec{x}_2[i+1] - \vec{x}_2[i]}\end{aligned}$$

Then by substituting the values in the equation for $f_{\mathcal{R}}(x)$ we get:

$$\begin{aligned}f_{\mathcal{R}}(x) &= \vec{u}_2[i] + \left(\vec{u}_2[i] + (\vec{\ell}_2[i+1] - \vec{u}_2[i])\frac{\vec{x}[j+1] - \vec{x}_2[i]}{\vec{x}_2[i+1] - \vec{x}_2[i]} - \vec{u}_2[i]\right)\frac{x - \vec{x}_2[i]}{\vec{x}[j+1] - \vec{x}_2[i]} \\ &= \vec{u}_2[i] + (\vec{\ell}_2[i+1] - \vec{u}_2[i])\frac{x - \vec{x}_2[i]}{\vec{x}_2[i+1] - \vec{x}_2[i]} = f_{\mathcal{R}_2}(x)\end{aligned}$$

In case 2 the value of $\vec{\ell}[j+1]$ is given by Equation 3b.

$$\vec{\ell}[j+1] = \max\{f_{\mathcal{R}_1}(\vec{x}[j+1]), \vec{\ell}_2[i+1]\}$$

We know that $\vec{x}_2[i+1] < \vec{x}_1[1]$ since $\vec{x}[j+1] = \vec{x}_2[i+1] \neq \vec{x}_1[1]$ and $\vec{x}[j] < \vec{x}_1[1]$. This gives that $f_{\mathcal{R}_1}(\vec{x}[j+1]) = \bot$, and thus $\vec{\ell}[j+1] = \vec{\ell}_2[i+1]$. Again, by substitution we obtain $f_{\mathcal{R}_2}(x)$.

In case 3 the value of $\vec{\ell}[j+1]$ is given by Equation 3c.

$$\vec{\ell}[j+1] = \max\{\vec{\ell}_1[1], \vec{\ell}_2[i+1]\} = \max\{\bot, \vec{\ell}_2[i+1]\} = \vec{\ell}_2[i+1]$$

Thus this case reduces to the case just discussed.

Case 4-8
Similar to the cases previously discussed.

Case 9
In general for cases 9-12, we know that there exists an i such that $\vec{x}_1[i] < x < \vec{x}_1[i+1]$, and there exists a j such that $\vec{x}_2[j] < x < \vec{x}_2[j+1]$. Let $a = \max\{\vec{x}_1[i], \vec{x}_2[j]\}$ and $b = \min\{\vec{x}_1[i+1], \vec{x}_2[j+1]\}$. Then the interval $]a, b[$ includes x, and we have cases 9-12.

In Case 9:

- $x \geq \vec{x}_1[1]$,
- $x \geq \vec{x}_2[1]$,
- $x \notin \mathit{elem}(\vec{x}_1)$,
- $x \notin \mathit{elem}(\vec{x}_2)$ and
- $\exists s \in S_{inter}$, s.t. $a < s < b$ and $x < s$.

In this case we have an intersection s, which has to be unique, since the two functions are linear in the interval. We first find the function values at x:

$$f_{\mathcal{R}_1}(x) = \vec{u}_1[i] + (\vec{\ell}_1[i+1] - \vec{u}_1[i])\frac{x - \vec{x}_1[i]}{\vec{x}_1[i+1] - \vec{x}_1[i]}$$

$$f_{\mathcal{R}_2}(x) = \vec{u}_2[j] + (\vec{\ell}_2[j+1] - \vec{u}_2[j])\frac{x - \vec{x}_2[j]}{\vec{x}_2[j+1] - \vec{x}_2[j]}$$

We know that there exists an index k, $1 \leq k \leq |\vec{x}|$ s.t. $\vec{x}[k] = a$ and $\vec{x}[k+1] = s$, since both a and s are elements in S_{max}.
We now calculate the value of $f_{\mathcal{R}}$ at x:

$$f_{\mathcal{R}}(x) = \vec{u}[k] + (\vec{\ell}[k+1] - \vec{u}[k])\frac{x - \vec{x}[k]}{\vec{x}[k+1] - \vec{x}[k]}$$

We can easily determine the value $\vec{\ell}[k+1]$ using Equation 3d:

$$\vec{\ell}[k+1] = f_{\mathcal{R}_1}(\vec{x}[k+1]) = f_{\mathcal{R}_2}(\vec{x}[k+1])$$

To determine $\vec{u}[k]$, we need to consider three cases, each with three subcases depending on the used definition for $\vec{u}[k]$. In all of the cases remember that $\vec{x}[k+1] = s$ and $f_{\mathcal{R}_1}(s) = f_{\mathcal{R}_2}(s)$.

1. If $\vec{x}_2[j] < \vec{x}_1[i]$ we have that $\vec{x}[k] = \vec{x}_1[i]$ and, by Equation 2a, $\vec{u}[k] = \max(\vec{u}_1[i], f_{\mathcal{R}_2}(\vec{x}[k])) = \max\{\vec{u}_1[i], f_{\mathcal{R}_2}(\vec{x}_1[i])\}$.
   (a) If $f_{\mathcal{R}_2}(\vec{x}_1[i]) > \vec{u}_1[i]$ then $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(x)$.
   (b) If $f_{\mathcal{R}_2}(\vec{x}_1[i]) < \vec{u}_1[i]$ then $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_1}(x)$.
   (c) It cannot be the case that $f_{\mathcal{R}_2}(\vec{x}_1[i]) = \vec{u}_1[i]$ since then s would not exist.
2. If $\vec{x}_2[j] > \vec{x}_1[i]$ we have that $\vec{x}[k] = \vec{x}_2[j]$ and, by Equation 2b, $\vec{u}[k] = \max\{f_{\mathcal{R}_1}(\vec{x}[k]), \vec{u}_2[j]\} = \max\{f_{\mathcal{R}_1}(\vec{x}_2[j]), \vec{u}_2[j]\}$.
   (a) If $f_{\mathcal{R}_1}(\vec{x}_2[j]) > \vec{u}_2[j]$ then $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_1}(x)$.
   (b) If $f_{\mathcal{R}_1}(\vec{x}_2[j]) < \vec{u}_2[j]$ then $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(x)$.
   (c) It cannot be the case that $f_{\mathcal{R}_1}(\vec{x}_2[j]) = \vec{u}_2[j]$, since then s would not exist.
3. If $\vec{x}_2[j] = \vec{x}_1[i]$ then $\vec{x}[k] = \vec{x}_1[i] = x_2[j]$ and, by Equation 2c, $\vec{u}[k] = \max\{\vec{u}_1[i], \vec{u}_2[j]\}$.
   (a) If $\vec{u}_2[j] > \vec{u}_1[i]$ then $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(x)$.
   (b) If $\vec{u}_2[j] < \vec{u}_1[i]$ then $\mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_1}(x)$.
   (c) It cannot be the case that $\vec{u}_2[j] = \vec{u}_1[i]$, otherwise s would not exist.

It can now be shown by substitution that $f_{\mathcal{R}}(x) = \mathcal{M}\boldsymbol{ax}(f_{\mathcal{R}_1}, f_{\mathcal{R}_2})(x)$ in each of the listed cases.

Case 10, 11 and 12
Similar to previous cases.

□

B Correctness of *Composition*

Theorem 2. *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be energy functions. We can in time $O(|\mathcal{R}_1|+|\mathcal{R}_2|)$ construct $\mathcal{R} = Composition(\mathcal{R}_1, \mathcal{R}_2)$, s.t. $f_{\mathcal{R}} = (f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})$, since the algorithm computes each of these values in linear time.*

Proof. Let $\mathcal{R}_1 = (\vec{x}_1, \vec{m}_1, \vec{u}_1, \vec{\ell}_1)$, $\mathcal{R}_2 = (\vec{x}_2, \vec{m}_2, \vec{u}_2, \vec{\ell}_2)$ and $\mathcal{R} = (\vec{x}, \vec{m}, \vec{u}, \vec{\ell})$.

It is clear from the definition of S_{comp} that $\vec{x}$ can have at most size $|\vec{x}_1|+|\vec{x}_2|$ and thus $\mathcal{R}$ can be constructed in time $O(|\mathcal{R}_1| + |\mathcal{R}_2|)$. Moreover it can be shown by exhaustive case analysis that $\mathcal{R}$ indeed is the representation of an energy function.

For proving that $f_{\mathcal{R}} = (f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})$ we have to consider different cases depending on the input value x:

- $x < \vec{x}_1[1]$ (1)
- $x \geq \vec{x}_1[1]$
  - $x = \vec{x}_1[i]$
    - $\vec{m}_1[i] < \vec{x}_2[1]$ (2)
    - $\vec{m}_1[i] \geq \vec{x}_2[1]$
      - $\vec{m}_1[i] \in elem(\vec{x}_2)$ (3)
      - $\vec{m}_1[i] \notin elem(\vec{x}_2)$ (4)
  - $\exists i$ s.t. $\vec{x}_1[i] < x < \vec{x}_1[i+1]$
    - $\exists x_2 \in elem(\vec{x}_2)$ s.t. $\vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x_2) < \vec{x}_1[i+1]$
      - $\exists j, 1 \leq j \leq |\vec{x}_2|$, s.t. $x = f_{\mathcal{R}_1}^{-1}(\vec{x}_2[j])$ (5)
      - $\exists j, 1 \leq j \leq |\vec{x}_2|$, s.t. $\vec{x}_2[j] = \min\{x' \mid x' \in elem(\vec{x}_2), x < f_{\mathcal{R}_1}^{-1}(x') < \vec{x}_1[i+1]\}, \nexists x_{low} \in elem(\vec{x}_2)$ s.t. $\vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x_{low}) < x$
        - $j = 1$ (6)
        - $j > 1$
          - $\vec{u}_1[i] \in elem(\vec{x}_2)$ (7)
          - $\vec{u}_1[i] \notin elem(\vec{x}_2)$ (8)
      - $\exists x_{high}$ s.t. $x_{high} = \min\{x' \mid x' \in elem(\vec{x}_2), x < f_{\mathcal{R}_1}^{-1}(x') < \vec{x}_1[i+1]\}, \exists x_{low}$ s.t. $x_{low} = \max\{x' \mid x' \in elem(\vec{x}_2), \vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x') < x\}$ (9)
      - $\nexists x_{high} \in elem(\vec{x}_2)$ s.t. $x < f_{\mathcal{R}_1}^{-1}(x_{high}) < \vec{x}_1[i+1]\}, \exists x_{low}$ s.t. $x_{low} = \max\{x' \mid x' \in elem(\vec{x}_2), \vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x') < x\}$
        - $\vec{\ell}_1[i+1] \in elem(\vec{x}_2)$ (10)
        - $\vec{\ell}_1[i+1] \notin elem(\vec{x}_2)$ (11)
    - $\nexists x_2 \in elem(\vec{x}_2)$ s.t. $\vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x_2) < \vec{x}_1[i+1]$
      - $\vec{u}_1[i] < \vec{x}_2[1]$ (12)
      - $\vec{u}_1[i] \geq \vec{x}_2[1]$
        - $\vec{u}_1[i] \in elem(\vec{x}_2)$
          - $\vec{\ell}_1[i+1] \in elem(\vec{x}_2)$
            - $\vec{u}_1[i] = \vec{\ell}_1[i+1]$ (13)
            - $\vec{u}_1[i] \neq \vec{\ell}_1[i+1]$ (14)
          - $\vec{\ell}_1[i+1] \notin elem(\vec{x}_2)$ (15)

- $\vec{u}_1[i] \notin elem(\vec{x}_2)$
  - $\vec{\ell_1}[i+1] \in elem(\vec{x}_2)$ (16)
  - $\vec{\ell_1}[i+1] \notin elem(\vec{x}_2)$ (17)

Case 1
In this case $x < \vec{x}_1[1]$.

We start by calculating $(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x)$.

$$(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(x)) = f_{\mathcal{R}_2}(\bot) = \bot$$

We now calculate $f_{\mathcal{R}}(x)$ and show that we get the same result.
We know that $\vec{x}[1] \geq \vec{x}_1[1]$ since S_{comp} either includes elements from $\vec{x}_1$ or values for which $f_{\mathcal{R}_1}$ gives an element in $\vec{x}_2$ ($f_{\mathcal{R}_1}$ is undefined before $\vec{x}[1]$). Thus, $x < \vec{x}[1]$ and $f_{\mathcal{R}}(x) = \bot$.

Case 2
In this case $x \geq \vec{x}_1[1]$, $x = \vec{x}_1[i]$ and $\vec{m}_1[i] < \vec{x}_2[1]$.

We again calculate the composed function at x:

$$(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(x)) = f_{\mathcal{R}_2}(\vec{m}[i])) = \bot$$

By definition of S_{comp} we know that $\vec{x}[1] > \vec{x}_1[i]$ since $\vec{m}_1[i] < \vec{x}_2[1]$. Thus $f_{\mathcal{R}}(x) = \bot$, and we again have the same result.

Case 3
In this case $x \geq \vec{x}_1[1]$, $x = \vec{x}_1[i]$, $\vec{m}_1[i] \geq \vec{x}_2[1]$ and $\vec{m}_1[i] \in elem(\vec{x}_2)$.

By the definition of S_{comp} we know that there exists an index $k, 1 \leq k \leq |\vec{x}|$, s.t. $\vec{x}_1[i] = \vec{x}[k]$. We can then prove this case by applying Equation 4.

$$f_{\mathcal{R}}(x) = \vec{m}[k] = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(\vec{x}[k])) = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(x)) = (f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x)$$

Case 4 and 5
Similar to case 3.

Case 6
In this case:

- $\exists i$ s.t. $\vec{x}_1[i] < x < \vec{x}_1[i+1]$, $\vec{x}_2[1] = \min\{x' \mid x' \in elem(\vec{x}_2), x < f^{-1}_{\mathcal{R}_1}(x') < \vec{x}_1[i+1]\}$ and
- $\nexists x_{low} \in elem(\vec{x}_2)$ s.t. $\vec{x}_1[i] < f^{-1}_{\mathcal{R}_1}(x_{low}) < x$.

Therefore we know that $\vec{x}_1[i] < x < f^{-1}_{\mathcal{R}_1}(x_2[1])$.
We then can calculate the value of the composed function:

$$(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x) = f_{\mathcal{R}_2}(f_{\mathcal{R}_1}(x)) = \bot$$

We proceed by calculating $f_{\mathcal{R}}(x)$. We know that $\vec{m}_1[i] < \vec{x}_2[1]$ since $\vec{x}_1[i] < f^{-1}_{\mathcal{R}_1}(x_2[1])$. Thus $\vec{x}[1] = f^{-1}_{\mathcal{R}_2}(\vec{x}_2[1])$ and $f_{\mathcal{R}}(x) = \bot$.

Case 7
In this case:

– $\exists i$ s.t. $\vec{x}_1[i] < x < \vec{x}_1[i+1]$,
– $\exists j, 1 < j \leq |\vec{x}_2|$, s.t. $\vec{x}_2[j] = \min\{x' \mid x' \in elem(\vec{x}_2),\ x < f_{\mathcal{R}_1}^{-1}(x') < \vec{x}_1[i{+}1]\}$,
– $\nexists x_{low} \in elem(\vec{x}_2)$ s.t. $\vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x_{low}) < x$ and
– $\vec{u}_1[i] \in elem(\vec{x}_2)$.

Here the composed function has the following value:

$$(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x) =$$

$$\vec{u}_2[j-1] + (\vec{\ell}_2[j] - \vec{u}_2[j-1]) \frac{\vec{u}_1[i] + (\vec{\ell}_1[i+1] - \vec{u}_1[i]) \frac{x - \vec{x}_1[i]}{\vec{x}_1[i+1] - \vec{x}_1[i]} - \vec{x}_2[j-1]}{\vec{x}_2[j] - \vec{x}_2[j-1]}$$

Again by construction of S_{comp} we know that there exists an index $k,\ 1 \leq k \leq |\vec{x}| - 1$, such that:

– $\vec{x}[k] = \vec{x}_1[i]$ and
– $\vec{x}[k+1] = f_{\mathcal{R}_1}^{-1}(\vec{x}_2[j]) = \vec{x}_1[i] + (\vec{x}_1[i+1] - \vec{x}_1[i]) \frac{\vec{x}_2[j] - \vec{u}_1[i]}{\vec{\ell}_1[i+1] - \vec{u}_1[i]}$.

The value $f_{\mathcal{R}}(x)$ can then be found by interpolation as usual. We therefore need the values of $\vec{u}[k]$ and $\vec{\ell}[k+1]$ to calculate the interpolation.
In the case of $\vec{\ell}$ we get $\vec{\ell}[k+1] = \vec{\ell}_2[j]$ by Equation 6d.
We use Equation 5c for deciding on the value of $\vec{u}[k]$ since we know that $\vec{x}[k] = \vec{x}_1[i]$ and $\vec{u}_1[i] \in elem(\vec{x}_2)$. Note that $\vec{u}_1[i] \neq \vec{\ell}_1[i+1]$. Otherwise, there would not exist a j that would satisfy the case. Moreover, $\vec{u}_1[i]$ must be equal to $\vec{x}_2[j-1]$. If it were not so, x_{low} would exist. Thus, $\vec{u}[k] = \vec{u}_2[j-1]$.
By substituting this values in the interpolation we get the same result as for $(f_{\mathcal{R}_1} \circ f_{\mathcal{R}_2})(x)$.

Case 8
In this case:

– $\exists i$ s.t. $\vec{x}_1[i] < x < \vec{x}_1[i+1]$,
– $\exists j, 1 < j \leq |\vec{x}_2|$, s.t. $\vec{x}_2[j] = \min\{x' \mid x' \in elem(\vec{x}_2),\ x < f_{\mathcal{R}_1}^{-1}(x') < \vec{x}_1[i{+}1]\}$,
– $\nexists x_{low} \in elem(\vec{x}_2)$ s.t. $\vec{x}_1[i] < f_{\mathcal{R}_1}^{-1}(x_{low}) < x$ and
– $\vec{u}_1[i] \notin elem(\vec{x}_2)$.

This case is very similar to the previous one. The only difference resides in the rule used to get the value of $\vec{u}[k]$. Here we know that $\vec{x}_2[j-1] < \vec{u}_1[i] < \vec{x}_2[j]$ since $f_{\mathcal{R}_1}^{-1}(\vec{x}_2[j-1]) < \vec{x}_1$ and $f_{\mathcal{R}_1}^{-1}(\vec{x}_2[j]) > \vec{x}_1[i]$. Thus, by Equation 5a we get:

$$\vec{u}[k] = f_{\mathcal{R}_2}(\vec{u}_1[i]) = \vec{u}_2[j-1] + (\vec{\ell}_2[j] - \vec{u}_2[j-1]) \frac{\vec{u}_1[i] - \vec{x}_2[j-1]}{\vec{x}_2[j] - \vec{x}_2[j-1]}$$

Again this case can then be shown by substitution when solving $f_{\mathcal{R}}(x)$.

Case 9
In this case:

– $\exists i$ s.t. $\vec{x}_1[i] < x < \vec{x}_1[i+1]$,
– $\exists x_{high}$ s.t. $x_{high} = \min\{x' \mid x' \in elem(\vec{x}_2), x < f^{-1}_{\mathcal{R}_1}(x') < \vec{x}_1[i+1]\}$ and
– $\exists x_{low}$ s.t. $x_{low} = \max\{x' \mid x' \in elem(\vec{x}_2), \vec{x}_1[i] < f^{-1}_{\mathcal{R}_1}(x') < x\}$.

By the definition of x_{high} and x_{low} we know that there exists an index j, $1 \leq j \leq |\vec{x}_2| - 1$ s.t. $x_{low} = \vec{x}_2[j]$ and $x_{high} = \vec{x}_2[j+1]$. Thus the value of the composed function is again the same as for case 7.
We proceed to calculate $f_{\mathcal{R}}(x)$. We know, by the definition of S_{comp}, that there exists an index $k, 1 \leq k \leq |\vec{x}|-1$, s.t. $\vec{x}[k] = f^{-1}_{\mathcal{R}_1}(\vec{x}_2[j])$ and $\vec{x}[k+1] = f^{-1}_{\mathcal{R}_1}(\vec{x}_2[j+1])$. The value of $f_{\mathcal{R}}$ at x is then found by interpolation. Here, we need again to substitute the values of $\vec{u}[k]$ and $\vec{\ell}[k+1]$ given by Equation 5d and 6d respectively.

$$\vec{u}[k] = \vec{u}_2[j] \qquad\qquad \vec{\ell}[k+1] = \vec{\ell}_2[j+1]$$

Case 10-17
Similar to the previous ones. □

C Addition and Ceiling for Time Functions

It is possible to construct a representation $\mathcal{T}'$ such that $f_{\mathcal{T}'} = f_{\mathcal{T}} + n$. This is done in Algorithm 10.

Algorithm 10: The algorithm computes the representation for the function representing the sum between a time function and a rational number.

Sum$(\mathcal{T}, n)$
Input - $\mathcal{T} = (\vec{x}, \vec{y}),\ n \in \mathbb{Q}_{\geq 0}$
Output - $\mathcal{T}' = (\vec{x}', \vec{y}')$ such that $f_{\mathcal{T}'} = f_{\mathcal{T}} + n$

For all $i,\ 1 \leq i \leq |\vec{x}|$ we have:
$\vec{x}'[i] = \vec{x}[i]$
$\vec{y}'[i] = \vec{y}[i] + n$

Proposition 4. *Let $\mathcal{T}$ be a time function representation and $\mathcal{T}'$ be the result of Sum$(\mathcal{T}, n)$ for some value $n \in \mathbb{Q}_{\geq 0}$. Then $f_{\mathcal{T}'} = f_{\mathcal{T}} + n$.*

Similarly for the ceiling operation, we can compute a representation $\mathcal{T}'$ such that $f_{\mathcal{T}'} = \mathit{ceil}(f_{\mathcal{T}}, k)$. To do this we first need to find out whether the value k is reached by any value of the time function and, if this is the case, at which input value this happens.

First of all we find the maximal index i_{max} of $\vec{y}$ which is still below k.

$$i_{max} = \begin{cases} \max\{i \mid \vec{y}[i] < k\} & \text{if } \{i \mid \vec{y}[i] < k\} \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

If $i_{max} = 0$, the function already returns a value higher or equal to k at an energy level of 0. On the other hand if $0 < i_{max} < |\vec{x}|$ the value k is reached at a higher energy level, which can be found by interpolation. We denote by x_{min} the minimal energy level needed to obtain the value k. Finally, if $i_{max} = |\vec{x}|$ then the function never reaches the value k, and therefore we do not define x_{min}.

$$x_{min} = \begin{cases} 0 & \text{if } i_{max} = 0 \\ \vec{x}[i_{max}] + (\vec{x}[i_{max}+1] - \vec{x}[i_{max}]) \frac{k - \vec{y}[i_{max}]}{\vec{y}[i_{max}+1] - \vec{y}[i_{max}]} & \text{if } 0 < i_{max} < |\vec{x}| \end{cases}$$

Now we can use Algorithm 11 to find the representation $\mathcal{T}'$.

Proposition 5. *Let $\mathcal{T}$ be a time function representation and $\mathcal{T}'$ be the result of Ceil$(\mathcal{T}, k)$ for some value $k \in \mathbb{N}_0$. Then $f_{\mathcal{T}'} = \mathit{ceil}(f_{\mathcal{T}}, k)$.*

Algorithm 11: The algorithm ensures that no value of vector $\vec{y}'$ is higher than k.

$Ceil(\mathcal{T}, k)$
Input - $\mathcal{T} = (\vec{x}, \vec{y}),\ k \in \mathbb{N}_0$
Output - $\mathcal{T}' = (\vec{x}', \vec{y}')$ such that $f_{\mathcal{T}'} = ceil(f_{\mathcal{T}}, k)$

If $i_{max} = |\vec{x}|$, then $\vec{x}' = \vec{x}$ and $\vec{y}' = \vec{y}$.
Otherwise, $\vec{x}'$ is defined as the increasing sequence of the elements contained in the following set:

$$S = elem(\vec{x}) \setminus \{\vec{x}[i] \mid \vec{x}[i] \in elem(\vec{x}), i > i_{max}\} \cup \{x_{min}, B\}$$

Furthermore, $\vec{y}'$ is defined as follows for all $i, 1 \leq i \leq |\vec{x}'|$:

$$\vec{y}'[i] = \begin{cases} \vec{y}[i] & \text{if } \vec{y}[i] < k \\ k & \text{otherwise} \end{cases}$$

D Correctness of *TimeFunction*

Theorem 5. *Let $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$ be a snippet. Then the algorithm TimeFunction($\mathcal{S}$) terminates and returns an array A, such that for all $t \in RT(\mathcal{S})$, and for all $e_{in} \in [0, B]$*

$$f_{A[t]}(e_{in}) = \max\Big\{ \sum_{0<i\leq j} d_i \Big| (t_{in}{}^\bullet, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} (t_{in}{}^\bullet, v'_{in}, e'_{in}) \xrightarrow{t_1} \dots \xrightarrow{d_j} ({}^\bullet t, v'_j, e'_j)\Big\}.$$

Proof. We will prove the correctness of the algorithm by proving that the following loop invariant for the loop "while $S \neq \emptyset$" holds:

For all $t \in RT(\mathcal{S}) \setminus S$ and for all $e_{in} \in [0, B]$,

$$f_{A[t]}(e_{in}) = \max\left\{ \sum_{0<i\leq j} d_i \Big| (t_{in}{}^\bullet, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} (t_{in}{}^\bullet, v'_{in}, e'_{in}) \xrightarrow{t_1} \dots \xrightarrow{d_j} ({}^\bullet t, v'_j, e'_j)\right\}.$$

– **Initialization**
We start by showing that the loop invariant holds before the first loop iteration. Here, $RT(\mathcal{S}) \setminus S = \emptyset$, so the invariant is trivially satisfied.

– **Maintenance**
For each $t' \in RT(\mathcal{S}) \setminus S$, we assume that we have found the time function returning the most time that can be spend before reaching the transition t'. Now a minimum element $t \in S$ is found and removed from S. Therefore, to prove that the loop invariant holds after this iteration of the loop, we need to prove that:
A) there exists a run from $(t_{in}{}^\bullet, [c = 0], e_{in})$ to $({}^\bullet t, v'_j, e'_j)$, such that the sum of the delays in the run is equal to $f_{A[t]}(e_{in})$ and
B) the sum of the delays along any other run from $(t_{in}{}^\bullet, [c = 0], e_{in})$ to $({}^\bullet t, v'_j, e'_j)$ is lesser than or equal to $f_{A[t]}(e_{in})$.

We start by proving A).

Since $t \in \Delta$ we know that there exists a path from t_{in} to t, by the definition of a snippet. We then only need to prove that the sum of the delays corresponds to the value of the function for a given input energy. We do that by constructing the time function and afterwards a run between the two transitions with delays summing up to the function value.
Naturally the time function depends on the path, it abstracts. If $t_{in} \lhd t$ we know that there exists a path without recharge transitions between the two transitions. A location q with minimum rate $\mu(t_{in}, t)$ between t_{in} and

t is then found and used to construct a time function representation. If $\mu(t_{in}, t) = 0$, the representation is $\mathcal{R} = (\langle 0, B\rangle, \langle k, k\rangle)$ and $f_{\mathcal{R}}(e_{in}) = k$ for all values $e_{in} \in [0, B]$.
Otherwise, we know that $\mu(t_{in}, t) > 0$, so the algorithm constructs the representation $\mathcal{R} = ceil\Big((\langle 0, B\rangle, \langle 0, \frac{B}{\mu(t_{in},t)}\rangle), k\Big)$. Then $f_{\mathcal{R}}(e_{in}) = \min\{\frac{e_{in}}{\mu(t_{in},t)}, k\}$ for all $e_{in} \in [0, B]$ by the definition of *ceil*.
We can then construct a run γ starting in $({t_{in}}^\bullet, [c = 0], e_{in})$ and ending in $({}^\bullet t, v'_j, B)$ such that the sum of all delays in the run is $f_{\mathcal{R}}(e_{in})$.

$$\gamma = ({t_{in}}^\bullet, [c = 0], e_{in}) \xrightarrow{0} ({t_{in}}^\bullet, [c = 0], e_{in}) \xrightarrow{true,\epsilon,\emptyset} \dots \xrightarrow{true,\epsilon,\emptyset} (q, [c = 0], e_{in})$$
$$\xrightarrow{d} (q, [c = d], e) \xrightarrow{true,\epsilon,\emptyset} (q', [c = d], e) \xrightarrow{0} \dots \xrightarrow{0} ({}^\bullet t, v'_j, B)$$

Here $e = e_{in} - d \cdot \mu(t_{in}, t)$ and if $\mu(t_{in}, t) = 0$, then $d = k$, otherwise $d = \min\{\frac{e_{in}}{\mu(t_{in},t)}, k\}$. The reader can easily check that the energy level never goes below zero along the run.
Moreover, if $t' \lhd t$ for some $t' \in RT(\mathcal{S})$, there exists a path from t' to t without recharge transitions, by the definition of $\lhd$. Additionally, since t was chosen as a minimal element of S we know that $t' \notin S$ prior to this iteration of the while-loop and therefore it holds that for all $e_{in} \in [0, B]$:

$$f_{A[t']}(e_{in}) = \max\Bigg\{ \sum_{0<i\le j} d_i \Bigg| \ ({t_{in}}^\bullet, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} ({t_{in}}^\bullet, v'_{in}, e'_{in}) \xrightarrow{t_1} \dots$$
$$\xrightarrow{d_j} ({}^\bullet t', v'_j, e'_j) \Bigg\}.$$

Again, the location q with minimal rate $\mu(t', t)$ on any path between t' and t is found and if $\mu(t', t) = 0$, the algorithm constructs the representation $\mathcal{R} = (\langle 0, B\rangle, \langle k, k\rangle)$, giving the energy function $f_{\mathcal{R}}(e_{in}) = k$ for all $e_{in} \in [0, B]$.
If $\mu(t', t) > 0$, the algorithm constructs the representation $\mathcal{R} = ceil\Big(A[t'] + \frac{B}{\mu(t',t)}, k\Big)$.
Then, $f_{\mathcal{R}}(e_{in}) = \min\{f_{A[t']}(e_{in}) + \frac{B}{\mu(t',t)}, k\}$ for all $e_{in} \in [0, B]$ by the definition of the addition and ceiling operations. Since $A[t']$ is correct, there exists a run γ starting in $({t_{in}}^\bullet, [c = 0], e_{in})$ and ending in $({}^\bullet t', v'_j, e'_j)$, where $v'_j = f_{A[t']}(e_{in})$. Let $t_{j+1} = t'$ and $q_n = {}^\bullet t$. The run γ can now be extended in the following way:

$$({}^\bullet t', v'_j, e'_j) \xrightarrow{t_{j+1}} (q_{j+1}, v_{j+1}, e_{j+1}) \xrightarrow{d_{j+1}} \dots \xrightarrow{t_n} (q_n, v_n, e_n) \xrightarrow{d_n} (q_n, v'_n, e'_n),$$

where $d_i = 0$ for all $i,\ j + 1 \le i \le n$ such that $q_i \neq q$, and otherwise

$$d_i = \begin{cases} \min\left\{\frac{B}{\mu(t',t)}, k - f_{A[t']}(e_{in})\right\} & \text{if } \mu(t', t) > 0 \\ k - f_{A[t']}(e_{in}) & \text{if } \mu(t', t) = 0 \end{cases}$$

Since t_{j+1} is a recharge transition, $e_{j+1} = B$ and it is easy to see that the energy level never goes below zero along the run.
Taking the maximum between two representations only results in choosing different runs for different values of e_{in} and therefore does not have an influence on the runs constructed. We have thus proved that there exists a run which delays $f_{A[t]}(e_{in})$ for each $e_{in} \in [0, B]$.

We now proceed to prove B).

Assume there exists a run γ' starting in $({t_{in}}^\bullet, [c = 0], e_{in})$ and ending in $({}^\bullet t, v'_j, e'_j)$. We have two cases:

1. there is a recharge transition other than t in the run
2. there are no other recharge transitions in the run.

We first analyse case 1. Let t' be the last recharge transition on the run before t. Since there is a path from t' to t, we have that $t' \lhd t$ and $f_{A[t']}(e_{in})$ returns the highest delay that can be achieved on a run from $({t_{in}}^\bullet, [c = 0], e_{in})$ to $({}^\bullet t', v'_j, e'_j)$ (by the loop invariant). This run has been extended by the algorithm by performing a delay in the location with least rate. It is therefore clear that delaying in any other location can only result in a total delay that is lesser or equal to the one of the constructed run.
For case 2 the algorithm has again chosen a location with minimal rate to delay in. Therefore any other distribution of the delay can at most result in a delay equal to the one returned by $f_{A[t']}(e_{in})$.

– **Termination**
At each iteration of the while-loop exactly one element is removed from S. Moreover the for-loop can at most be performed $RT(\mathcal{S}) - 1$ times for each recharge transition. Therefore the algorithm terminates after a finite number of iterations. At this point we have that $S = \emptyset$ and thus $RT(\mathcal{S}) \setminus S = RT(\mathcal{S})$. Therefore, by the loop invariant, we have that for all $t \in RT(\mathcal{S})$ and for all $e_{in} \in [0, B]$,

$$f_{A[t]}(e_{in}) = \max \left\{ \sum_{0<i\leq j} d_i \,\middle|\, ({t_{in}}^\bullet, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} ({t_{in}}^\bullet, v'_{in}, e'_{in}) \xrightarrow{t_1} \ldots \xrightarrow{d_j} ({}^\bullet t, v'_j, e'_j) \right\}.$$

This proves the theorem.

□

E Correctness of *EnergyFunction*

Theorem 6. *Let $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$ be a snippet and let $\mathcal{R}$ be the energy function representation returned by EnergyFunction($\mathcal{S}$).*

Then if $\mathcal{R}$ is defined, we have that for all $e_{in} \in [0, B]$

$$f_{\mathcal{R}}(e_{in}) = \max\Big\{e_{out}\Big|\ \exists \gamma \in \mathit{Runs}(\mathcal{S})\ \mathit{s.t.}\ \gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} (t^{\bullet}_{in}, v'_{in}, e'_{in}) \xrightarrow{t_1} \ldots \xrightarrow{d_j} ({}^{\bullet}t_{out}, v'_j, e'_j) \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})\Big\}$$

If $\mathcal{R}$ is not defined, then there does not exist any run $\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \to \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})$ in $\mathit{Runs}(\mathcal{S})$.

Proof. We start by proving that if $\mathcal{R}$ is defined, then:

A) For all $e_{in} \in [0, B]$, there exists a run $\gamma \in \mathit{Runs}(\mathcal{S})$, where $\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \to \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})$ such that $e_{out} = f_{\mathcal{R}}(e_{in})$.

B) For any run $\gamma \in \mathit{Runs}(\mathcal{S})$, where $\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \to \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})$, it holds that $e_{out} \leq f_{\mathcal{R}}(e_{in})$.

We start by proving A).

Let g_{out} be the guard of t_{out}. We then have the following cases:

1. $g_{out} = \mathit{true}$ or $g_{out} =$ "$c = 0$"
   a) $RT(\mathcal{S}) \neq \emptyset$
   b) $RT(\mathcal{S}) = \emptyset$
2. $g_{out} =$ "$c = k$", $k \neq 0$
   a) $t_{out} \in RT(\mathcal{S})$
   b) $t_{out} \notin RT(\mathcal{S})$

We start by considering case 1. The guard does not require any delay, and it can therefore always be satisfied. Moreover, for subcase a), $RT(\mathcal{S}) \neq \emptyset$ and the algorithm constructs the representation $\mathcal{R} = (\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$. Thus, $f_{\mathcal{R}}(e_{in}) = B$ for all $e_{in} \in [0, B]$. We also know that there exists a recharge transition $t \in RT(\mathcal{S})$. We can therefore construct the following run that achieves the same energy as the function $f_{\mathcal{R}}$.

$$\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \xrightarrow{0} \ldots \xrightarrow{t} (t^{\bullet}, [c = 0], B) \xrightarrow{0} \ldots \xrightarrow{0} ({}^{\bullet}t_{out}, [c = 0], B) \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], B)$$

Note that if $t = t_{out}$, the energy level B is is not obtained until after t_{out}, where the run terminates.

Now consider subcase b), where $RT(\mathcal{S}) = \emptyset$. Here the algorithm constructs the representation $\mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$, and thus, $f_{\mathcal{R}}(e_{in}) = e_{in}$ for all $e_{in} \in [0, B]$. We can then construct the following run:

$$\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \xrightarrow{0} \ldots \xrightarrow{0} ({}^{\bullet}t_{out}, [c = 0], e_{in}) \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{in})$$

We now proceed to prove case 2. Here, g_{out} = "$c = k$", $k \neq 0$ and thus we are required to delay k time units before we can take the transition t_{out}. Consider subcase a). Since $t_{out} \in RT(\mathcal{S})$, *TimeFunction* has calculated a time function for t_{out} and, if $f^{-1}_{A[t_{out}]}(k) \neq \bot$, we construct the representation $\mathcal{R} = (\langle f^{-1}_{A[t_{out}]}(k), B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$. Note that when $f^{-1}_{A[t_{out}]}(k) = B$ we construct a representation with only this value in the vector $\vec{x}$ to satisfy the requirements for an energy function representation. In any case, we then have that:

$$f_{\mathcal{R}}(e_{in}) = \begin{cases} B & \text{if } e_{in} \in [f^{-1}_{A[t_{out}]}(k), B] \\ \bot & \text{otherwise} \end{cases}$$

Now, since $A[t_{out}]$ is correct, we know that there exists a run $\gamma \in \mathit{Runs}(\mathcal{S})$ that starts in $(t^{\bullet}_{in}, v_{in} = [c = 0], e_{in})$ and ends in $({}^{\bullet}t_{out}, [c = k], e_{out})$ for all $e_{in} \in [f^{-1}_{A[t_{out}]}(k), B]$. We can extend this run with the transition t_{out}, thus reaching the state $(t^{\bullet}_{out}, [c = 0], B)$. For all $e_{in} < f^{-1}_{A[t_{out}]}(k)$ there does not exist a run reaching ${}^{\bullet}t_{out}$ with valuation $[c = k]$ and therefore there does not exists any run of the desired form in $\mathit{Runs}(\mathcal{S})$.

Now consider case b). Here $t_{out} \notin RT(\mathcal{S})$, and thus we do not have a time function defined for t_{out}. Then the representation $\mathcal{R}$ is constructed as the maximum of different representations. We will show that for each of the representations used for this operation there exists a run, that achieves the same energy as the one returned by the energy function. Taking the maximum will only change the run chosen for the given input energy e_{in}.

First the algorithm checks whether there exists a path without recharges from t_{in} to t_{out}. If this is the case and the minimum rate location between those has rate zero, it constructs the representation $\mathcal{R} = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$. Then, for this first step, we have that $f_{\mathcal{R}}(e_{in}) = e_{in}$ for all $e_{in} \in [0, B]$.

If the minimum rate location has a rate larger than zero, it calculates the time function $\mathcal{T}$ for a run of this form as previous algorithms. This is then used to find the minimal energy for which the run can delay the k time units required to satisfy the guard of t_{out}. Then the representation $\mathcal{R} = (\langle f^{-1}_{\mathcal{T}}(k), B\rangle, \langle 0, B - k \cdot \mu(t_{in}, t_{out})\rangle, \langle 0\rangle, \langle B - k \cdot \mu(t_{in}, t_{out})\rangle)$ is constructed. For all $e_{in} \in [f^{-1}_{\mathcal{T}}(k), B]$ the function is $f_{\mathcal{R}}(e_{in}) = e_{in} - k \cdot \mu(t_{in}, t_{out})$. Clearly for lower energy levels it is not possible to satisfy the guard on this set of paths. Let q be a location with rate $\mu(t_{in}, t_{out})$. We can construct the following run:

$$\begin{aligned} \gamma = (t_{in}{}^{\bullet}, [c = 0], e_{in}) &\xrightarrow{0} (t_{in}{}^{\bullet}, [c = 0], e_{in}) \xrightarrow{true,\epsilon,\emptyset} \ldots \xrightarrow{true,\epsilon,\emptyset} (q, [c = 0], e_{in}) \\ &\xrightarrow{k} (q, [c = k], e_{in} - k \cdot \mu(t_{in}, t_{out})) \xrightarrow{true,\epsilon,\emptyset} (q', [c = k], e_{in} - k \cdot \mu(t_{in}, t_{out})) \\ &\xrightarrow{0} \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, [c = 0], e_{in} - k \cdot \mu(t_{in}, t_{out})) \end{aligned}$$

Afterwards the algorithm goes through all recharge transitions t that may directly precede t_{out} and calculates the time that can be spent based on the minimum rate location and the result of *TimeFunction*($\mathcal{S}$), again taking special care if the minimum rate location has rate zero. The representation of the energy

function is then constructed in the function *SetupVectors*, where values from $A[t]$ are used to construct a function $f_{\mathcal{R}} = B - (k - f_{A[t]}(e_{in})) \cdot \mu(t, t_{out})$. From the correctness of *TimeFunction* we know that for all $e_{in} \in [0, B]$ there exists a run starting in state $(t^{\bullet}_{in}, [c = 0], e_{in})$ and ending in state $({}^{\bullet}t, [c = f_{A[t]}(e_{in})], e)$ for some energy level $e \in [0, B]$. Let q be a location with rate $\mu(t, t_{out})$, we can then extend this run in the following way:

$$({}^{\bullet}t, [c = f_{A[t]}(e_{in})], e) \xrightarrow{t} (t^{\bullet}, [c = f_{A[t]}(e_{in})], B) \xrightarrow{0} \ldots \xrightarrow{true,\epsilon,\emptyset}$$
$$(q, [c = f_{A[t]}(e_{in})], B) \xrightarrow{k - f_{A[t]}(e_{in})} (q, [c = k], B - (k - f_{A[t]}(e_{in})) \cdot \mu(t, t_{out}))$$
$$\xrightarrow{true,\epsilon,\emptyset} (q', [c = k], B - (k - f_{A[t]}(e_{in})) \cdot \mu(t, t_{out})) \xrightarrow{0} \ldots \xrightarrow{t_{out}}$$
$$(t^{\bullet}_{out}, [c = 0], B - (k - f_{A[t]}(e_{in})) \cdot \mu(t, t_{out}))$$

This concludes the proof of A).

We now prove B). Again we have different cases depending on the guard of t_{out}.

1. $g_{out} = true$ or $g_{out} =$ "$c = 0$"
2. $g_{out} =$ "$c = k$", $k \neq 0$

In case 1, if $RT(\mathcal{S}) \neq \emptyset$ the function constructed always returns B, that is the highest energy level that can be achieved and thus there does not exist a run having a higher energy. On the other hand if $RT(\mathcal{S}) = \emptyset$, the energy function returns e_{in}. Clearly any run segment in $Runs(\mathcal{S})$, can only obtain an energy level that is lower or equal to the one at the start of the run, as there is no recharge transition in the snippet.

In case 2 we shall see that all the constructed runs obtain the highest energy level. Consider a run $\gamma \in Runs(\mathcal{S})$, $\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \rightarrow \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})$. If γ does not have a recharge, then $e_{out} \leq f_{\mathcal{R}}(e_{in})$ as $\mathcal{R}$ is constructed by performing a delay in the location with minimal rate $\mu(t_{in}, t_{out})$. Now assume that γ has at least one recharge transition. Let t be the last recharge transition. Then the run can at most spend $f_{A[t]}(e_{in})$ time units before taking the transition t. If γ spends less time in this part of the run it would be forced by the guard g_{out} to spend more time after t. This results in spending more time after the last recharge transition on the run, thus obtaining an energy level that is lesser or equal (in case $\mu(t, t_{out}) = 0$) to the one returned by $f_{\mathcal{R}}$.

This concludes part B) of the proof.

Finally, we have to prove that if $\mathcal{R}$ is not defined then there does not exist a run $\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \rightarrow \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})$ in $Runs(\mathcal{S})$.

This case can only happen if the guard of t_{out} is on the form "$c = k$". Here, we have two cases depending on whether there is a recharge on t_{out} or not.

If t_{out} has a recharge, *TimeFunction* has computed a function returning the longest delay that can be achieved on a run from $t^{\bullet}_{in}$ to ${}^{\bullet}t_{out}$. The energy function $\mathcal{R}$ is undefined if $f^{-1}_{A[t_{out}]}(k) = \perp$. This means that there does not exist a simple

run segment in the snippet $\mathcal{S}$ that can delay enough time to satisfy the guard of t_{out}. Thereby no run of the desired form exists in $\mathit{Runs}(\mathcal{S})$.

If t_{out} does not have a recharge, we consider whether there exists a path without recharge transitions from $t_{in}^{\bullet}$ to ${}^{\bullet}t_{out}$. If this is the case, we can use the technique from Algorithm 3 of finding the relation between energy levels and minimum rate locations to construct a time function that returns the longest delay that can be performed on a run following this kind of path. Again we use the inverse time function to find out whether the delay required by the guard of t_{out} can be performed. If the inverse function is not defined for this value, the guard cannot be satisfied without taking recharge transitions and the representation $\mathcal{R}$ will not yet be initialized. The algorithm then considers all recharge transitions t, where $t \lhd t_{out}$ and computes the maximal delay in the usual way while maximizing the time function. If after having considered all these transitions the energy function representation $\mathcal{R}$ still has not been initialized, then it is not possible to delay enough time on any path in the snippet to satisfy the guard of t_{out} and thereby there does not exist a run taking this transition in $\mathit{Runs}(\mathcal{S})$.

□

F Time Complexity of *EnergyFunction*

Lemma 2. *Let* $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, rate, \Delta)$ *be a snippet. Then* *EnergyFunction*$(\mathcal{S})$ *runs in time* $O(|\Delta|^3 + |\Delta|^2 \cdot |Q|)$.

Proof. Before analyzing *EnergyFunction*, we will determine the complexity of *TimeFunction*, which is used within it.

TimeFunction

Let $\lhd^+$ denote the transitive closure of $\lhd$ and $\lhd^*$ denote the transitive and reflexive closure of $\lhd$. Then, given a recharge transition t, we can define the set $W_t = \{\mu(t_{in}, t') \mid t' \in RT(\mathcal{S}), t_{in} \lhd t' \text{ and } t' \lhd^* t\}$.

We will now argue that the time complexity of *TimeFunction* is $O(|\Delta|^3 + |\Delta|^2 \cdot |Q|)$ and the size of each constructed time function is $O(|\Delta|)$. Furthermore, the possible slopes of the time functions are given by $\mu(t_{in}, t)$ for each $t \in RT(\mathcal{S})$ or 0.

We initially observe that the size of the set $RT(\mathcal{S})$ can at most be Δ.

Note that it is possible to determine the recharge ordering between the transitions by running DFS once. This information can then be used to both determine the minimal element in $S \setminus \{t_{out}\}$ and the $\lhd$ relation between two transitions.

Given two transitions t and t', where $t \lhd t'$, we need to determine $\mu(t, t')$. We can use yet another DFS to find the location with least rate that lie on a path without recharge transitions between t and t'. Since this calculation is performed at most once for each pair of transitions, we get a total complexity of $O(|\Delta|^2 \cdot (|\Delta| + |Q|))$.

Now we will prove for each transition t that the constructed time function representation $A[t]$ has polynomial size.

By induction on the iteration number, we prove that the size of $|W_t|$ is at most i and that for all $t \in RT(\mathcal{S})$ the function $f_{A[t]}$ is composed of linear segments, each having either a higher slope than the previous segment with a value of $\frac{1}{n}$ for some $n \in W_t$, or a slope of 0 and function value k.

Base step

We start by considering the first iteration. Since the algorithm selects a minimum element t in $RT(\mathcal{S}) \setminus \{t_{out}\}$ there does not exist a $t' \in RT(\mathcal{S}) \setminus \{t_{out}\}$, such that $t' \lhd t$. Moreover it must be the case that $t_{in} \lhd t$ and $W_t = \{\mu(t_{in}, t)\}$, as otherwise t would not be a minimum element of the snippet $\mathcal{S}$.

The time function $A[t]$ is constructed in lines 9-13. If $\mu(t_{in}, t) = 0$ (line 13) we construct a representation with slope 0. Otherwise, in line 11 we construct a representation with slope $\frac{1}{\mu(t_{in}, t)}$. Then we use the ceiling operation which may add a line segment with a slope of 0, if the function for some value returns values higher than k. Since there does not exists $t' \in RT(\mathcal{S})$, such that $t' \lhd t$, the algorithm does not enter the forall-loop in line 16.

The function $f_{A[t]}$ is thus composed of linear segments each having a slope of $\frac{1}{\mu(t_{in}, t)}$ or 0 in this order, and if the slope is 0 the function has value k. The

representation $A[t]$ thus ends up with a final size of at most 3. This concludes the base step.

Induction step
Assume the hypothesis holds up to and including iteration i. Let t be the minimal transition chosen in iteration $i+1$. We want to show that $|W_t| \leq i$ and that the function $f_{A[t]}$ only consists of linear segments that each has either a higher slope than the previous segment with a value of $\frac{1}{n}$ for some $n \in W_t$, or a slope of 0 and function value k.

In lines 11 to 15, a representation with at most two linear segments is constructed, one with slope $\frac{1}{\mu(t_{in},t)}$ and one with slope 0, as in the base step. Then the algorithm goes through all transitions $t' \in RT(\mathcal{S}) \setminus \{t_{out}\}$, such that $t' \lhd t$. We know that none of these transitions is in the set S at iteration $i+1$. Otherwise the element t would not be minimal. Therefore by induction hypothesis we have that for all $t' \in RT(\mathcal{S}) \setminus \{t_{out}\}$, such that $t' \lhd t$, $A[t']$ is composed of linear segments that each has either a higher slope than the previous segment with a value of $\frac{1}{n}$ for some $n \in W_{t'}$, or a slope of 0 and function value k. Moreover, $|W_{t'}| \leq i$. Since t is a minimal element, we have already at iteration i considered all transitions $t' \in RT(\mathcal{S})$ s.t. $t' \lhd^+ t$, and we now have that $|W_t| \leq i+1$.

Let t' be the first transition considered in line 16. If $f_{A[t']}$ already contains the maximal amount of linear segments, $i+1$, taking the maximum in line 19 can at most add one new linear segment with slope $\frac{1}{\mu(t_{in},t)}$ to the function. This is the case since there then already is a line with slope 0 and function value k in $f_{A[t']}$. This line can either dominate the other function or have the same value. In both cases there will only be one segment with slope 0. On the other hand if a segment with slope 0 was not present, then it would be added. In both cases there would be a total of $i+2$ segments.

Now assume there exists another transition t_p, such that $t_p \lhd t$. Then the function $Maximum_{\mathcal{T}}$ is called once more. However, it holds that $|W_{t'} \cup W_{t_p}| \leq i$ since there are at most $i+1$ values in W_t and $\mu(t_{in},t)$ is not included in $W_{t'} \cup W_{t_p}$. Thereby there are at most $i+1$ different slopes in total when considering all functions $f_{A[t']}$ where $t' \in RT(\mathcal{S}) \setminus \{t_{out}\}$ and $t' \lhd t$. Moreover by induction hypothesis the lines in each function have an increasing slope until the function reaches the value k after which the slope becomes 0. If there is not one function that completely dominates the other when taking the maximum between two linear functions, then the slope of the maximum function will initially be the lowest of the two slopes and then after the intersection change to the highest for the upper domain values. Due to this and the fact that two lines with the same slope cannot intersect, each slope gives rise to at most one linear segment in $A[t]$. Thus the function $f_{A[t]}$ has at most $i+2$ linear segments with slopes in the desired order.

Then each time function representation has a size of at most $|\Delta| + 2$ ($|\Delta| + 1$ linear segments are represented by $|\Delta| + 2$ points). Since the operation $Maximum_{\mathcal{T}}$ goes through all points once to compare them, computing the

maximum between two such functions has a complexity of $O(|\Delta|)$. Finally, this operation is executed at most $|\Delta|^2$ times because of the loops.

Summing it all up, the calculation of the minimum rates turns out to be the most expensive operation, since it is embedded within two loops performed no more than $|\Delta|$ times each. Thus, the complexity of *TimeFunction* is $O(|\Delta|^3 + |\Delta|^2 \cdot |Q|)$.

EnergyFunction

The algorithm starts by calling *TimeFunction*, adding the complexity of it to this algorithm. This algorithm also makes use both of the calculation of the recharge ordering and the location with the minimum rate, which we determined to have complexity $O(|\Delta|^2 \cdot (|\Delta| + |Q|))$. Later we iterate at most $|\Delta|$ times in a for-loop. We then call *SetupVectors* which performs at most $|\vec{x}_{A[t]}|$ operations on all elements of $\vec{x}$, $\vec{m}$, $\vec{u}$ and $\vec{\ell}$, meaning that it is once again determined by the size of a time function, which we found to be at most $O(|\Delta|)$. Besides these, one extra value may be added to the representation of the energy function to include a last function value at the value B in $\vec{x}$. The actual size of the final energy function representations can then at most be $|\Delta| \cdot O(|\Delta|) = O(|\Delta|^2)$, since we go through the loop no more than $|\Delta|$ times and each time the maximum operation can add $|\Delta|$ new values to the representation based on values coming from a time function.

As we can see, the overall complexity of the algorithm is then $O(|\Delta|^3 + |\Delta|^2 \cdot |Q|)$.

□

G Correctness of *EnergyFunctionAutomaton*

Theorem 7. *Let $\mathcal{A}$ be a one-clock closed recharge automaton in sNF. Then the algorithm EnergyFunctionAutomaton($\mathcal{A}$) runs in time $O(|\Delta|^5 + |\Delta|^4 \cdot |Q|)$ and constructs an energy function automaton $\mathcal{E}$ such that there exists a winning run of $\mathcal{A}$ iff there exists a winning run of $\mathcal{E}$.*

Proof. We first analyse the complexity of the algorithm and then proceed with proving correctness.

First of all the algorithm constructs a snippet for each pair of reset transitions $t_{in},\ t_{out}$, where $Paths(t_{in}, t_{out}) \neq \emptyset$. We can use DFS to find out whether there exists such a path, and then to determine the set of transitions and locations that are included in the snippet. Thus the construction of all the snippets has complexity $O(|\Delta|^2 \cdot (|\Delta| + |Q|))$.

In the second forall-loop the function *EnergyFunction* is called, which has complexity $O(|\Delta|^3 + |\Delta|^2 \cdot |Q|)$. The algorithm calls *EnergyFunction* once for each iteration and since it goes through every snippet, it has at most $|\Delta|^2$ iterations. Moreover in each iteration it also calls *ExtraTransition*, which has at most $|\Delta|$ recursive calls to itself. Thus, the complexity of *EnergyFunctionAutomaton* will be $|\Delta|^2 \cdot O(|\Delta|^3 + |\Delta|^2 \cdot |Q|) = O(|\Delta|^5 + |\Delta|^4 \cdot |Q|)$.

We now proceed with proving the correctness of the algorithm.

To do this we need to show that:

A) if there exists a winning run of $\mathcal{A}$ we can construct a winning run of $\mathcal{E}$.
B) if there exists a winning run of $\mathcal{E}$ we can construct a winning run of $\mathcal{A}$.

We start by proving A).

Let $\Gamma = (q_1, v_1, e_1) \xrightarrow{d_1} (q_1, v_1', e_1') \xrightarrow{t_1} (q_2, v_2, e_2) \xrightarrow{d_2} (q_2, v_2', e_2') \xrightarrow{t_2} (q_3, v_3, e_3) \xrightarrow{d_3} \dots$ be a winning run of $\mathcal{A}$. We will now show how to construct a winning run of $\mathcal{E}$. Clearly the first discrete transition of any infinite run in $\mathcal{A}$ is a reset transition, since $\mathcal{A}$ is in sNF. Moreover since $time(\Gamma) = \infty$ and since every location has an invariant of the form $c \leq k, k \in \mathbb{N}$, there must be infinitely many reset transitions along the run. Let $t_{in} = (q_0, g, z, \{c\}, q)$ and let t_{out} be the second reset transition in Γ. By the construction of $\mathcal{E}$ there exists a transition on the form $(q, \mathcal{R}, b, t_{out}^{\bullet})$, where for all $e_{in} \in [0, B]$,

$$f_{\mathcal{R}}(e_{in}) = \max\Big\{ e_{out} \Big|\ \exists \gamma \in Runs(\mathcal{S}) \text{ s.t. } \gamma = (t_{in}^{\bullet}, v_{in} = [c = 0], e_{in}) \xrightarrow{d_1} \dots \xrightarrow{t_{out}} (t_{out}^{\bullet}, v_{out} = [c = 0], e_{out}), \Big\}.$$

Moreover since $rate(q_0) = 0$, Γ reaches q with energy level B. Therefore we can start the run ρ of $\mathcal{E}$ with the transition $(q, B) \xrightarrow{\mathcal{R}, b} (t_{out}^{\bullet}, f_{\mathcal{R}}(B))$. Let $(t_{out}^{\bullet}, [c = 0], e_{out})$ be the state reached by Γ after the transition t_{out}. Clearly $f_{\mathcal{R}}(B) \geq e_{out}$ and the transition is therefore legal.

We can now continue the run by considering the next reset transition t in Γ. Again, there exists a transition in $\mathcal{E}$ from $t^\bullet_{out}$ to $t^\bullet$ for which the same properties as above hold and, since energy functions are non-decreasing functions, having a higher energy level at any state will not have a negative effect. We can do this for all reset transitions in Γ, and thereby construct an infinite run ρ in $\mathcal{E}$.

Now we have to prove that ρ is winning, meaning that it has an infinite amount of transitions with $b = 1$.

Consider the transitions of ρ that have $b = 0$. These are constructed from a snippet where either

a) the guard on the last transition is "$c = 0$" or
b) the guard on the last transition is "*true*", there is no recharge transition in the snippet and all locations have a rate higher than 0.

In case a), there does not exist any infinite run in $\mathcal{A}$ which can delay in such a snippet.

In the case b), Γ can make a delay in the snippet, if the energy level at $t^\bullet_{in}$ is greater than zero. The maximal delay that Γ can perform in such a snippet is $\frac{e_{in}}{\mu(t_{in},t_{out})}$. Assume that $time(\rho) \neq \infty$. This means that ρ has only finitely many transitions with $b = 1$ and infinitely many with $b = 0$. Though if all transitions with $b = 0$ are obtained from case b), only a finite delay can be performed in Γ and it would not be winning. Thus there must be an alternation of transitions obtained from case a) and case b). This means that the run Γ must have infinitely many run segments of the form shown below as there can only be a delay in a snippet of the type described in case b) if there is a snippet with at least one recharge transition, but guard "$c = 0$".

$$\xrightarrow{g,z,\{c\}} (q_1, v_1, e_1) \xrightarrow{d_1} (q_1, v_1', e_1') \xrightarrow{t_2} (q_2, v_2, e_2) \xrightarrow{d_2} \ldots \xrightarrow{true,\epsilon,\{c\}} (q_i, v_i, e_i) \xrightarrow{d_i}$$
$$\ldots \xrightarrow{d_m} (q_m, v_m', e_m') \xrightarrow{\text{“}c=0\text{”},\epsilon,\{c\}} \ldots \xrightarrow{d_n} (q_n, v_n', e_n') \xrightarrow{\text{“}c=0\text{”},z_n,\{c\}}$$
$$(q_{n+1}, v_{n+1}, e_{n+1})$$

Here we have that for all $j,\ 2 \leq j \leq i-1$ or $m+1 \leq j < n$, $R_j = \emptyset$. Also, for all $j,\ i+1 \leq j \leq m$, either $t_j = (q_{j-1}, \text{“}c = 0\text{”}, \epsilon, \{c\}, q_j)$ or $R_j = \emptyset$. Finally, there exists a $j,\ m+1 \leq j \leq n$, such that $z_j = r$.

Since there are infinitely many such segments in Γ, the run must at some point repeat some location in a snippet of the type in b). But then we can substitute this cycle with the transition $\big(q_1, (\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle), 1, q_1\big)$. This transition was added by the function *ExtraTransition*. We now have a winning run in $\mathcal{E}$.

We now prove B).

Let $\rho = (q_0, e_0 = B) \xrightarrow{\mathcal{R}_0,b_0} (q_1, e_1) \xrightarrow{\mathcal{R}_1,b_1} (q_2, e_2) \xrightarrow{\mathcal{R}_2,b_2} \ldots$ be a winning run of $\mathcal{E}$. We will first prove that we can construct an infinite run in $\mathcal{A}$ and then that this is actually a winning run.

First of all, to construct an infinite run Γ in $\mathcal{A}$, we start with the following run segment, taking the first reset transition.

$$(q_0, [c = 0], B) \xrightarrow{0} (q_0, [c = 0], B) \xrightarrow{g,z,\{c\}} (q, [c = 0], B)$$

Then we consider all transitions in ρ in the order they are taken in the run. We know that for each transition $(q_i, \mathcal{R}_i, b_i, q_{i+1})$ in ρ there exists a transition in the energy function automaton, and by construction of this automaton, we have one of the following two cases:

a) Either there exists a snippet $\mathcal{S} = (t_{in}, t_{out}, Q, B, C, k, \mathit{rate}, \Delta)$ where $t^{\bullet}_{in} = q_i$, $t^{\bullet}_{out} = q_{i+1}$, and $\mathit{EnergyFunction}(\mathcal{S}) = \mathcal{R}_i$ or
b) $q_i = q_{i+1}$ and the transition was added using the function *ExtraTransition*.

Consider case a). By Theorem 6, we have that for all $e_{in} \in [0, B]$:

$$f_{\mathcal{R}_i}(e_{in}) = \max \Big\{ e_{out} \Big| \exists \gamma \in \mathit{Runs}(\mathcal{S}) \text{ s.t. } \gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) \rightarrow \\ \ldots \xrightarrow{t_{out}} (t^{\bullet}_{out}, v_{out} = [c = 0], e_{out}) \Big\}$$

Therefore for each transition taken in ρ of this type, it is possible to find a run segment that achieves the same energy.

Now consider case b). In this case there exists a sequence of snippets $\mathcal{S}_m \ldots \mathcal{S}_n$, where for all j, $m \leq j \leq n$, $\mathcal{S}_j = (t_{in_j}, t_{out_j}, Q_j, B_j, C_j, k_j, \mathit{rate}_j, \Delta_j)$, $t^{\bullet}_{in_m} = t^{\bullet}_{out_n} = q_i$, $g_{out_m} = \mathit{true}$, $RT(\mathcal{S}_m) = \emptyset$, $\mu(t_{in_m}, t_{out_m}) > 0$, for all j, $m \leq j < n$, $t_{out_j} = t_{in_{j+1}}$, there exists ℓ, $m < \ell \leq n$ such that $RT(\mathcal{S}_\ell) \neq \emptyset$ and $g_{out_\ell} =$ "$c = 0$", for all $j \neq \ell$, $m < j \leq n$, either $g_{out_j} =$ "$c = 0$" or $g_{out_j} = \mathit{true}$. Let $d = \frac{e_{in}}{\mathit{rate}_m(t^{\bullet}_{in_m})}$. Then we can add to Γ the run segment shown in Equation 7.

$$(t^{\bullet}_{in_m}, [c = 0], e_{in}) \xrightarrow{d} (t^{\bullet}_{in_m}, [c = d], 0) \xrightarrow{t_1} (t^{\bullet}_1, [c = d], 0) \xrightarrow{0} \ldots \xrightarrow{t_{out_m}} \\ (t^{\bullet}_{in_{m+1}}, [c = 0], 0) \xrightarrow{0} \ldots \xrightarrow{t_{out_n}} (t^{\bullet}_{out_n}, [c = 0], B) \tag{7}$$

Note that only the first delay in the run segment is nonzero, and that since there exists a snippet with a recharge transition $\mathcal{S}_\ell$ it is possible to achieve the energy level B at the end of the run independently of the value of e_{in}. Finally all transitions in the segment can be taken as the guard is either *true* or "$c = 0$".

Note that in both case a) and b) the run segments end in a state $(t^{\bullet}_{out}, v_{out} = [c = 0], e_{out})$, which is then used as the start state of the run segment which substitutes the next transition in ρ. This makes it possible to just concatenate the run segments, obtaining an infinite run of $\mathcal{A}$.

We now need to show that $\mathit{time}(\Gamma) = \infty$, thus proving that Γ is a winning run of $\mathcal{A}$.

Since ρ is winning, we know that $\mathit{time}(\rho) = \sum_{i=0}^{\infty} b_i = \infty$ and therefore there must be infinitely many transitions $(q_i, \mathcal{R}_i, b_i, q_{i+1})$ in ρ, where $b_i = 1$. As mentioned each transition corresponds to a transition in $\mathcal{E}$, which is constructed

either from a sequence of snippets or from a single snippet $\mathcal{S}$. Let g_{out} be the guard of the transition t_{out} for the snippet $\mathcal{S}$. Then the constructed transition is given the value $b_i = 1$ if:

1. it was constructed from a single snippet $\mathcal{S}$
   (a) $g_{out} =$ "$c = k$", where $k \neq 0$, or
   (b) $g_{out} = true$ and either
      i. $\mathcal{R}_i \neq (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$ or
      ii. $\mathcal{R}_i = (\langle 0, B\rangle, \langle 0, B\rangle, \langle 0\rangle, \langle B\rangle)$ and $\mu(t_{in}, t_{out}) = 0$.
2. it was constructed from *ExtraTransition*.

Clearly in case 1a the run segment which is used to replace the transition $(q_i, \mathcal{R}_i, b_i, q_{i+1})$ must delay k time units.

However in case 1b the run segment is not required to perform any delay. First consider case 1(b)ii. We will show that it is possible to construct a run segment γ starting in state $(q_i, v_{in} = [c = 0], e_{in})$ and ending in state $(q_{i+1}, v_{out} = [c = 0], f_{\mathcal{R}_i}(e_{in}))$ such that $time(\gamma) > 0$. Note that in this case $f_{\mathcal{R}_i}(e_{in}) = e_{in}$ for all $e_{in} \in [0, B]$. Let q be a location such that $rate(q) = \mu(t_{in}, t_{out}) = 0$. Then the run segment γ will be:

$$\begin{aligned}\gamma = (q_i, v_{in} = [c = 0], e_{in}) &\xrightarrow{0} (q_i, v_{in} = [c = 0], e_{in}) \xrightarrow{true,\epsilon,\emptyset} \ldots \xrightarrow{true,\epsilon,\emptyset} \\ &(q, [c = 0], e_{in}) \xrightarrow{k} (q, [c = k], e_{in}) \xrightarrow{true,\epsilon,\emptyset} (q', [c = k], e_{in}) \xrightarrow{0} \ldots \xrightarrow{t_{out}} \\ &(q_{i+1}, [c = 0], e_{in})\end{aligned}$$

Now consider case 1(b)i and case 2. We will first show that for all e_{in}, $0 < e_{in} \leq B$, it is possible to construct a run segment γ starting in state $(q_i, v_{in} = [c = 0], e_{in})$ and ending in state $(q_{i+1}, v_{out} = [c = 0], f_{\mathcal{R}_i}(e_{in}))$ such that $time(\gamma) > 0$. In both cases the representation $\mathcal{R}_i$ can only have the form $(\langle 0, B\rangle, \langle B, B\rangle, \langle B\rangle, \langle B\rangle)$ as can be seen in Algorithm 4. In case 1(b)i there exists at least one recharge transition in $RT(\mathcal{S})$. Let t be a recharge transition such that $t_{in} \lhd t$. We can then construct the run segment γ, where $t^{\bullet}_{in} = q_i,\ t^{\bullet}_{out} = q_{i+1}$ and

$$d = \begin{cases} k & \text{if } rate(t^{\bullet}_{in}) = 0 \\ \frac{e_{in}}{rate(t^{\bullet}_{in})} & \text{otherwise} \end{cases}$$

$$\begin{aligned}\gamma = (t^{\bullet}_{in}, v_{in} = [c = 0], e_{in}) &\xrightarrow{d} (t^{\bullet}_{in}, v_{in} = [c = d], e_{in} - d \cdot rate(t^{\bullet}_{in})) \xrightarrow{true,\epsilon,\emptyset} \\ &(q_2, v_{in} = [c = d], e_{in} - d \cdot rate(t^{\bullet}_{in})) \xrightarrow{0} \ldots \xrightarrow{t} (t^{\bullet}, [c = d], B) \xrightarrow{0} \ldots \\ &\xrightarrow{true,\epsilon,\emptyset} ({}^{\bullet}t_{out}, [c = d], B) \xrightarrow{0} ({}^{\bullet}t_{out}, [c = d], B) \xrightarrow{t_{out}} (t^{\bullet}_{out}, [c = 0], B)\end{aligned}$$

Thus for all e_{in}, $0 < e_{in} \leq B$ we have that $time(\gamma) > 0$. A similar run can be constructed in case 2 (see Equation 7).

Now consider the energy level $e_{in} = 0$. In both cases, even though the run segments constructed do not spend any delay, they reach the energy level B in their last state. Now we show that the energy level B is preserved until a delay is

performed. In the run ρ, the transition taken into consideration can be followed either by a transition with $b = 1$ or one with $b = 0$. If it has $b = 1$ we have already shown that we can construct a run segment γ where $time(\gamma) > 0$. On the other hand if $b = 0$ the energy function representation on the transition represents either the function $f(e_{in}) = e_{in}$ for all $e_{in} \in [0, B]$ or the function $f(e_{in}) = B$ for all $e_{in} \in [0, B]$. In both cases the energy level remains B and since $time(\rho) = \infty$ there will be at some later point in the run a transition with $b = 1$, which as shown corresponds to a run segment where some time is spent.

Thus we have shown that if there exists a winning run in $\mathcal{E}$ then there exists a winning run in $\mathcal{A}$.

We have now proved that there exists a winning run in $\mathcal{A}$ iff there exists a winning run in $\mathcal{E}$. □

H NP-Hardness of a Time-Bounded Run

Here we will prove that the time-bound problem is NP-hard even for recharge automata with only one clock and no cycles.

Problem 2. Given a recharge automaton $\mathcal{A}$ and a time bound $T \in \mathbb{N}_0$, does there exist a run γ of $\mathcal{A}$ where $time(\gamma) \geq T$?

Theorem 8. *The time-bounded run problem is NP-hard.*

Proof. This proof is by reduction from SUBSET-SUM. An instance of the SUBSET-SUM problem is a pair (A, t), where $A \subset \mathbb{N}$ is a finite set and $t \in \mathbb{N}$. The problem is to find out whether there exists a subset of A whose elements exactly sum up to t.

Assume an instance of SUBSET-SUM (A, t), where $A = \{t_1, t_2, \ldots, t_n\}$. We construct a recharge automaton $\mathcal{A} = (Q, q_1, t, C, I, rate, \Delta)$, where

- $Q = \{q_1, q_2, \ldots, q_n, q_{n+1}\}$,
- $C = \{c\}$,
- $rate(q_i) = 1$ for all $i,\ 1 \leq i \leq n$, $rate(q_{n+1}) = 0$,
- $I(q_i) = \ c \leq t_i$ for all $i,\ 1 \leq i \leq n$, $I(q_{n+1}) = \ c \leq 1$, and
- $\Delta = \{(q_i, c = t_i, \epsilon, \{c\}, q_{i+1}) \mid 1 \leq i \leq n\} \cup \{(q_i, c = 0, \epsilon, \emptyset, q_{i+1}) \mid 1 \leq i \leq n\}$.

The automaton is shown in Figure 14.

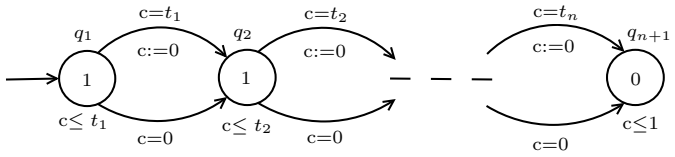


Fig. 14: Reduction from SUBSET-SUM.

Let the minimum time be $T = t + 1$. There exists a run γ with $time(\gamma) \geq T$ iff there is a solution to SUBSET-SUM.

In each location q_i we have the choice to either include t_i in the subset or not. If we choose to wait in a location, we include it in the sum and thus decrease the energy level, which will then correspond to what is missing to reach the number t. Note that it is not possible to wait in any of the locations from q_i to q_n for all the T time units required, since we will have 0 energy after only t time units. This way we are forced to spend the last time unit required in location q_{n+1}.

It is clear that if SUBSET-SUM has a solution, the time-bounded run problem also has a solution. On the other hand if SUBSET-SUM does not have a solution, there is no solution to the time-bounded run problem.

Any run can spend at most t time units in the locations q_1 to q_n, so we are forced to reach q_{n+1} to spend the last time unit, and to do so we can either include or exclude each number from the subset.

If the numbers chosen sum up to more than t, the run will not include the last number fully as it would make the resource go under 0.

If the numbers chosen sum up to less than t the run will reach q_{n+1} where it is only allowed to wait for one time unit, so the total time of the run would be less than T.

Thus, there would not be a solution to the time-bounded run problem either. □