TECHNICAL REPORT - Contributions to ω-regular energy problems

Rania Saadi
(supervisor: Philipp Schlehuber)

Technical Report n^ooutput, July 8th 2024
revision bf5e93f

This report explores solutions for energy problems on ω-regular automata as well as game theory concepts and how they can be utilized for our solution. We introduce a method for solving ω-regular energy problems on generalized Büchi automata with multiple acceptance conditions in one run. The proposed approach builds on a lattice structure that guarantees convergence to the same fixed point, regardless of the back-propagation order, enabling parallelization. Upon full development and optimization, including an extension for path retrieval, an implementation of the algorithm is planned on the Spot library.[1]

Ce rapport explore des solutions pour les problèmes d'énergie sur les automates ω-réguliers ainsi que des concepts de théorie des jeux et comment ils peuvent être utilisés pour notre solution. Nous introduisons une méthode pour résoudre les problèmes d'énergie ω-réguliers sur les automates de Büchi généralisés avec plusieurs conditions d'acceptation en une seule exécution. L'approche proposée utilise une structure de treillis qui garantit la convergence vers le même point fixe, quelle que soit l'ordre de rétropropagation, permettant ainsi la parallélisation. Une fois pleinement développé et optimisé, y compris une extension pour la récupération de chemin, une implémentation de l'algorithme est prévue sur la bibliothèque Spot.[1]

Keywords
Büchi energy problem, weighted automata, progress measure, score sheet, energy constraint, acceptance condition, lattice, parity games

Laboratoire de Recherche de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
rsaadi@lre.epita.fr – http://www.lre.epita.fr/

Copying this document

Contents

0.1 Introduction

Automata theory is a theoretical branch of computer science and mathematics, a tool mathematicians (such as Alan Turing, Alonzo Church, and John von Neumann) developed in the mid-20th century by application and abstraction to imitate human actions seeking a way to perform computations more efficiently and reliably. An automaton is an abstract model of a machine that performs computations on an input by transitioning through a graph of states until it reaches an accepting configuration. Systems described by automata were at first restricted to a series of tasks that recieve a fixed set of inputs at predictable intervals then terminate.

finite automata: is used to recognize finite words, known as regular languages, they operate on a finite set of states and transitions and are expressive enough to describe systems with finite behavior.

timed automata: To work with real-time systems, we sometimes need knowledge of time that decides behavior. Each transition is associated with a time constraint and the whole automaton is associated with one or more clocks. While they are expressive enough to model systems where behavior depends on time, they are often better translated to other types of automata for easier problem-solving.

automata as weighted graphs: On a weighted graph we assign a cost to transitions between states. In the context of automata theory, they often represent energy consumption or gain which is useful to study feasible behavior. A well-known algorithm to study paths and behavior relative to weights is the Bellman-Ford algorithm.

automata for infinite words: With the evolution of automation needs, we have progressed towards reactive systems, systems that do not follow a definite sequence of actions and terminate, but instead run indefinitely, responding continuously to input. We aim to be able to formally prove their feasibility and robustness: feasibility is checked by solving for known initial conditions and costs, while robustness involves considering all possible responses from an environment treated as an adversary against acceptance conditions. Applications range from communication protocols to energy management in smart grids... etc.
Automata theory and temporal logic have proven effective in describing such systems. ω-regular automata in particular, a subclass of finite automata, describes languages with infinite words, which corresponds to infinite runs in automated systems. The letter "ω" denotes an infinite sequence, and "regularity" refers to describing automata with a finite number of states.
ω-regular automata is mainly a tool to express liveliness properties that cannot be expressed using looping automata as all states of looping automata are accepting and can only represent a subset of ω-regular languages that doesn't involve eventuality. There are several classes of ω-regular automata tailored for various applications, such as Büchi automata, Rabin automata, and parity automata. Each class is best suited for specific problems and acceptance conditions. For theoretically infinite runs in timed weighted automata, we focus on Büchi automata.

ω-regular languages A regular language A is a language that can be described by a regular expression and verified with finite automata. A^ω is the concatenation of A infinitely many times and is an ω-regular language for A not containing the empty word ϵ (in which case $A = A^\omega$).
The concatenation of A and A^ω is, by definition, an ω-regular language.
A^ω concatenated to A is not defined.

The union of a finite number of ω-regular languages is an ω-regular language as well.
Every ω-regular language is recognized by a nondeterministic Büchi automaton.

Our Objective:

We aim to extend the concept of energy feasible paths to energy ω-regular problems. This involves investigating multiple directions, including the use of a modified Bellman-Ford algorithm to construct a coherent trace and reconstruct a run from it, and leveraging the game-like interaction between the agent and the environment to handle multiple acceptance conditions.
In the following sections, we will introduce the ω-regular energy problems and provide an overview of the current literature. We will discuss trace extraction using the modified Bellman-Ford algorithm[2] and its associated challenges. We will then introduce a unified solver algorithm by gradually building on game theory concepts, highlighting similarities with parity games[3]. We will delve into the details of our implementation and the benefits of using a lattice structure[4]. Finally, we will conclude by outlining future improvements we aim to incorporate into our research and discussing the potential of our solution. Additionally, we will briefly mention recent advances made toward similar goals.
I would like to thank my supervisor, Philipp Schlehuber, for his guidance throughout this exciting semester and for introducing me to the world of research, as well as for his valuable feedback on my report.

0.2 State of the art : ω-regular energy problems

Consider a network of electric vehicle charging stations over a busy part of the city connected to a smart grid. It would be ideal for the vehicles if energy were available for full charging when they reach a station. What we seek is a good balance between preventing energy shortage and responding to the vehicles' needs. Each charging station has the eventuality of charging a vehicle and operates under the different energy levels the grid supplies (energy constraint). In this chapter, we explore tools to model such systems, check their feasibility, and extract their feasible configurations.

0.2.1 Büchi automata

A Büchi automaton is a tuple $A = (M, S, s_0, T)$ with :

- M a finite set of colors representing acceptance conditions.
- S a set of states.
- s_0 the starting state with $s_0 \in S$.
- T a set of transitions $t = (s, M, s')$ with $T \subseteq S \times 2^M \times S$. For a weighted Büchi automaton (WBA) that expresses energy for example, we write $t = (s, M, w, s')$ and $T \subseteq S \times 2^M \times R \times S$ with $w \in R$ representing the assigned weight. For a finite WBA w is an integer.

A run in a Büchi automaton is a finite or infinite sequence of states p.
A concatenation of two runs $p_1 p_2$ is defined if p_1 is finite and the last state of $p1$ is equal to the first state of p_2.
p_ω denotes infinite iteration and is the injective limit of p_n (or direct limit, could be thought of as the point at which a composition returns the same object, the limit)
For a weighted Büchi automaton we also associate an accumulated weight called initial credit ic to each state and limit it by an upper bound b.

0.2.2 ω-regular energy problems

Energy problems are feasibility problems with additional energy constraints that need to be respected throughout an infinite run. In the context of WBA, energy is represented by weights. We cite[5]:

- **Lower-bound problems:** given an initial credit, the accumulated weight never drops below 0 during a run. They express the feasibility of a run given an energy level.
- **Interval-bound problems:** the accumulated weight stays within a given interval. Meaning the run is deemed invalid as soon as the energy crosses either bounds.
- **Lower-weak-upper-bound problems:** the accumulated weights never drop below zero and are bounded at the weak upper bound if they exceed it. They express the feasibility of a run given an energy level and a limited capacity.

Each one of these problems deals with different decidability properties and complexities. During the rest of the report, we will discuss lower weak upper bound problems as they are the

closest to a physical system with bounds on availability and capacity. We define a weak upper bound wup and an initial credit ic.

0.2.3 Emptiness check and path retrieval

A path is said to be (c, b)-feasible if accumulated energy is never negative for a given initial credit and weak upper bound.
The problem is seen as equivalent to searching for (c, b)-feasible lassos which consist of a possibly empty prefix that's traversed only once and a cycle that's looped indefinitely.
Such lassos are obtained by first finding all energy optimal paths to construct the prefix, running Couvreur's algorithm to retrieve the SCCs, degeneralizing them then checking for energy feasible cycles.
A modified Bellman-Ford algorithm allows the detection of negative cycles in polynomial time (if energy is propagated in between iterations, which becomes necessary because of the weak upper bound).
The algorithm answers for feasibility and leaves a useful trace but the latter needs to be reconstructed as there are reasons that make it that the trace gets broken and overwritten.
For instance, loops that need to be taken a certain number of times to allow access to more rewarding ones have their trace overwritten. To avoid the necessity for a complete reconstruction of the path, modifications have been brought to the algorithm to store all predecessors.
The retrieval of the path is then done through backward exploration over lists of predecessors which is essentially a combinatorial problem over all possible lists with repeated predecessors for a standard graph traversal technique. This calls for optimization; therefore, a few lemmas are established :

Lemma 1. *Nested loops are not necessary for energy feasibility. Indeed, every path of the form*

$$s \to (a \to y_1 \to (\tau)^+ \to y_2 \to a)^+ \to d$$

can be rewritten as:

- $s \to a \to y_1 \to (\tau)^+ \to y_2 \to a \to d$ *when the inner loop and its suffix are energy optimal for* d.
- $s \to a \to y_1 \to (\tau)^+ \to y_2 \to (a \to y_1 \to y_2 \to a)^+ \to d$ *when the inner loop is energy optimal for* y_2 *and the outer loop is energy optimal for* d.

By decomposition, we can do the same for more than two nested loops. All runs can then be written as a series of cycles and infixes, both possibly empty, that are taken one time followed by a looping cycle.

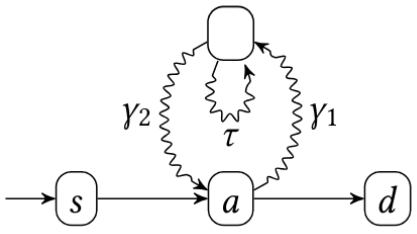

Figure 1: Figure illustrating nested loops from [2]

Lemma 2. *Given two strictly energy positive loops τ_1 and τ_2 as in the figure below, sharing a common reachable state that belongs to the path from the source to the destination. Given some initial credit and*

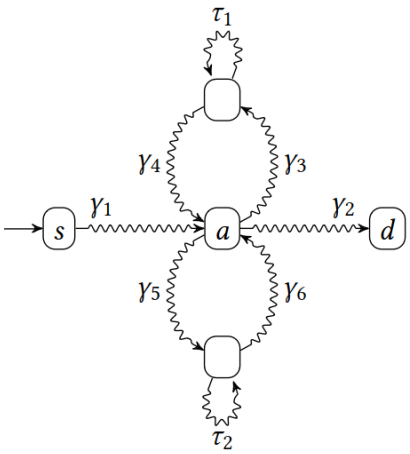

Figure 2: Figure illustrating the loops τ_1 and τ_2 sharing a common reachable state from [2]

a weak upper bound, the maximal energy is attainable in d by one of the following paths:

- $s \to y_1 \to a \to y_2 \to d$
- $s \to y_1 \to a \to y_3 \to \tau_1^+ \to y_4 \to a \to y_2 \to d$
- $s \to y_1 \to a \to y_5 \to \tau_2^+ \to y_6 \to a \to y_2 \to d$
- $s \to y_1 \to a \to y_3 \to \tau_1^+ \to y_4 \to a \to y_5 \to \tau_2^+ \to y_6 \to a \to y_2 \to d$
- $s \to y_1 \to a \to y_5 \to \tau_2^+ \to y_6 \to a \to y_3 \to \tau_1^+ \to y_4 \to a \to y_2 \to d$

Depending on the energy optimal predecessors. This shows that revisiting an energy-positive loop is never necessary.

0.2.4 The trace extraction algorithm

The algorithm propagates initial credit along a path and compares it to the minimum desired energy at the destination. Following the last predecessors is not possible as they can lead to an infinite loop. The notion of chronological coherence is then introduced.
A run is considered chronologically coherent with the extended predecessors P if and only if:

- Predecessors of states appearing once in a run are in P.
- For states appearing more than once, for a new P_a of predecessors there exists a monotone index array such that $P_a[i] = P[idx[i]]$.

Due to the asymmetry caused by the weak upper bound, back-propagating over all chronologically coherent traces does not return the reversed path, checks of energy feasibility by forward propagation are still necessary.

This method works well for one acceptance condition but must be run multiple times for multiple acceptance conditions. This calls for the creation of an operator that can be applied to a whole graph and stores more information on its trace. This takes us to the next chapter, unified solver algorithms.

0.3 Unified solver algorithms

In this section, our goal is to explore the foundations to develop an algorithm that holistically analyzes the entire automaton and effectively eliminates the need for separate searches over all back edges. To achieve this, we are going to propose a method inspired by parity game solvers, which will be thoroughly introduced in the following sections. The reasons behind this choice and the specific adaptations made to suit our problem will be discussed in detail in the next section. Before that, we will present the fundamental principles of game theory, parity games, and their solvers. Game theory is the study of strategic decision-making. Traditionally associated with economics and social sciences, it offers an effective model for settings where multiple actors make interconnected dependent choices. Today, it finds its place in computer science, particularly in the analysis of reactive systems; the systems on which we defined our energy problems.
Interactive computation, a paradigm shift from the classical view of input-output processes, recognizes that modern computational systems have to take into consideration their environment as they don't simply execute and terminate. They run indefinitely and maintain a dialogue with the environment.
The model allows a rich set of tools and operations to study the properties of a setting. The system and its environment represent players, their possible actions represent moves in the game, and the desired outcome becomes the winning condition. We find ourselves answering the same questions, feasibility and robustness.
In addition to the new perspective we get on our energy problems, we also move to more global definitions of the solutions on our automata instead of path-like separated ones.

0.3.1 Weighted automata and games

We start by presenting the defining components of a game.

Definition 1. *An* arena *is a tuple $A = (V, V_0, V_1, E)$ with*

- *V a finite set of vertices.*
- *$V_0, V_1 \subseteq V$ disjoint subsets such that $V = V_0 \cup V_1$, denoting the vertices of Player 0 and Player 1 respectively.*
- *$E \subseteq V \times V$ a set of (directed) edges such that every vertex has at least one outgoing edge (the automaton must be non-blocking).*

We also define a sub-arena *of A induced by v_0 as:*

$$A_{V_0} = (V \cap V_0, V_0 \cap V_0, V_1 \cap V_0, E \cap (V_0 \times V_0))$$

Definition 2. *A* play *is an infinite sequence $\rho = \rho_0\rho_1\rho_2 \cdots \in V^\omega$ such that $(\rho_n, \rho_{n+1}) \in E$ holds for every $n \in \mathbb{N}$. The set of plays in A is denoted by Plays(A), the set of all plays starting in v by*

$$\forall V_0 \subseteq V, \quad \textit{Plays}(A, V_0) = \bigcup_{v \in V_0} \textit{Plays}(A, v)$$

Definition 3. *$\forall i \in \{0, 1\}$, a* strategy *for Player i in an arena (V, V_0, V_1, E) is a function*
$\sigma : V^* \times V_i \to V$ such that
$\forall w \in V^*, \forall v \in V_i, \exists v' \in V$ with $\sigma(w, v) = v'$ and $(v, v') \in E$

Now that we defined the building blocks, there are a few interesting properties that can be associated with a game to give information about the nature of the solution we are looking for and how to approach it.

Definition 4. *A* consistent play $\rho_0\rho_1\rho_2\cdots$ *in an arena* $A = (V, V_0, V_1, E)$ *is consistent with a strategy* σ *for Player* i *in* A *if:*
$\forall n \in \mathbb{N}\ \rho_{n+1} = \sigma(\rho_0 \cdots \rho_n)$ *with* $\rho_n \in V_i$.
Given a vertex v_0*, we denote the set of plays that are consistent with* σ *and start in* v_0 *with Plays*(A, v_0, σ).
Finally, we define Plays(A, V_0, σ) *for* $V_0 \subseteq V$ *by*

$$\textit{Plays}(A, V_0, \sigma) = \bigcup_{v \in V_0} \textit{Plays}(A, v, \sigma)$$

Definition 5. *A* game $G = (A, \textit{Win})$ *consists of an arena* A *with a vertex set* V *and a set of winning sequences* $\textit{Win} \subseteq V^\omega$.
A sequence ρ *is winning for Player 0 if, and only if,* $\rho \in \textit{Win}$*, otherwise, it's winning for Player 1.*

Definition 6. *The* winning region $W_i(G)$ *of Player* i *in a game* G *is the set of vertices from which Player* i *has a winning strategy.*

Lemma 3. *We have* $W_0(G) \cap W_1(G) = \emptyset$ *for every game* G.

Definition 7. *A* winning strategy σ *for Player* i *in* A *is a winning strategy from a vertex* $v \in V$ *if every play that is consistent with* σ *and starts in* v *is winning for Player* i*. We write:*
σ *is a winning strategy if* $\textit{Plays}(A, v, \sigma) \subseteq \textit{Win}$ *for* $i = 0$ *and* $\textit{Plays}(A, v, \sigma) \subseteq V^\omega \setminus \textit{Win}$ *for* $i = 1$.

Definition 8. *A* positional strategy σ *for Player* i *in an arena* $A = (V, V_0, V_1, E)$ *is positional if:*
$\forall (w, v) \in V^* \times V_i;\ \sigma(wv) = \sigma(v)$

Identifying when a game can be won using a positional strategy is crucial, as it allows for significant optimization of computational resources. Positional strategies focus solely on the current state of play, eliminating the need to store or process information about past moves or future possibilities. This approach substantially reduces memory requirements and computational complexity, leading to more efficient algorithms.

Definition 9. Determinacy *and* Positional Determinacy. *Let* G *be a game with vertex set* V*. We say that* G *is determined if* $W_0(G) \cup W_1(G) = V$.
In addition to that, if one of the players has a positional winning strategy from every vertex in the Arena, we say that G *is positionally determined.*

Definition 10. *A* uniform positional winning strategy. *Let the game* $G = (A, \textit{Win})$*. A strategy* σ *for Player* i *is a uniform positional winning strategy if it is positional and winning from every vertex in the winning region* $W_i(G)$.

To develop strategies, we analyze patterns in how moves are controlled by regions and establish the following definitions.

Definition 11. *A* trap. *Let* $A = (V, V_0, V_1, E)$ *be an arena and let* $T \subseteq V$*.* T *is a trap for Player* i*, if*

for a Player i $\forall v \in T \cap V_i, \forall (v, v') \in E, v' \in T$

and

for a Player 1 - i $\forall v \in T \cap V_{1-i}, \exists v' \in T; (v, v') \in E$

Lemma 4. *Let T be a trap for Player i in an arena A.*

1. *The restriction A_T of A to T is a valid sub-arena.*
2. *If T' is a trap for Player i in A_T, then it is also a trap for Player i in A.*

To make use of traps in our strategies and identify whether they can be used in a game configuration we define prefix independence.

Definition 12. Prefix-independence. *A winning condition Win $\subseteq V^\omega$ is prefix-independent, if:* $\forall \rho \in V^\omega, \rho \in Win \iff \forall w \in V^*, w \cdot \rho \in Win$

Lemma 5. *Let $G = (A, Win)$ be a game with prefix-independent winning condition Win. Then, $W_i(G)$ is a trap for Player $1 - i$.*

Definition 13. *A game on weighted automata is a tuple $G = (S_1, S_2, s_0, T)$ where:*

- *S_1 and S_2 are two disjoint sets of locations (states).*
- *s_0 is the initial location.*
- *$T \subseteq (S_1 \cup S_2) \times \mathbb{R} \times (S_1 \cup S_2)$ is the set of transitions.*

$A_G = (S_1 \cup S_2, s_0, T)$ is a weighted automaton.

- *The automaton has to be non-blocking, meaning, every location has at least one outgoing transition. According to Martin's determinacy theorem, such games are determined.*

Definition 14. *A Polish space is a topological space homeomorphic (topologically equivalent) to a complete metric space (a set with a defined distance function between its elements). Furthermore, the space must possess a countable dense subset, meaning a subset where every point in the space is arbitrarily close to at least one point in this subset.*

Theorem 1. *Martin's determinacy theorem: every two-player perfect information game (all players have complete knowledge of moves made in the game) where players take turns playing natural numbers, and the set of all possible moves forms a Polish space, is determined.*

0.3.2 Büchi games

Definition 15. *Let $A = (V, V_0, V_1, E)$ be an arena and let $F \subseteq V$ be a subset of A's vertices. Then, the Büchi condition Büchi(F) is defined as:*

$$\textit{Büchi}(F) := \{\rho \in V^\omega \mid \textit{Inf}(\rho) \cap F \neq \emptyset\}$$

We call a game $G = (A, \textit{Büchi}(F))$ a Büchi game with recurrence set F.

to build the attractor-based method to finding the winning region of a Büchi game we define $F^n \subseteq F$ that represents vertices to which recurring visits are still desirable for Player 0 for a given round n.
We initialize with $F^0 = F$, as every vertex in F is potentially desirable at the start of a game. Then, in each round, we determine the vertices from which Player 1 can prevent reaching F^n at all. They represent the n-th under-approximation of W_1^n and they become undesirable so we remove them to obtain the new set F^{n+1} and we repeat the process described above until the under-approximations become stationary. The final approximation yields the winning region of Player 1 from which we can deduce the winning region of Player 0.
Formally we write the construction below:

Construction 1 (Recurrence)**.** *Let $A = (V, V_0, V_1, E)$ be an arena and let $F \subseteq V$. The recurrence construction for Player i is defined inductively as:*

- $F^0 = F$
- $W_1^n = V \setminus \textit{Attr}_0(F^n)$ *for every* $n \geq 0$
- $F^{n+1} = F \setminus \textit{CPre}_1(W_1^n)$ *for every* $n \geq 0$

Lemma 6. *Let $G = (A, \textit{Büchi}(F))$ be a Büchi game. Then,*

$$W_1(G) = \bigcup_{n \in \mathbb{N}} W_1^n \quad \textit{and} \quad W_0(G) = V \setminus W_1(G)$$

0.3.3 Co-Buchi games

Before defining a Co-Büchi game we define a dual arena to establish the link with Büchi games.

Definition 16 (Dual Arena)**.** *Let $A = (V, V_0, V_1, E)$ be an arena. The dual arena A' of A is defined as $A' = (V, V_1, V_0, E)$*

Co-Büchi games are the dual of Büchi games, Player 0 aims to visit vertices from a given set C only finitely often. Stated differently, from some point onwards the play has to be restricted to the complement of C.

Definition 17. *Let $A = (V, V_0, V_1, E)$ be an arena and let $C \subseteq V$ be a subset of A's vertices. Then, the co-Büchi condition is defined as:*
co-Büchi(C):= $\{\rho \in V^\omega \mid \textit{Inf}(\rho) \subseteq C\}$
We call a game $G = (A, \textit{co-Büchi}(C))$ a co-Büchi game with persistence set C.

Lemma 7. *Since co-Büchi and Büchi games are dual. Consequently, we conclude the same results for co-Büchi games as for Büchi games.*

Just like Büchi games :

Theorem 2. *Co-Büchi games are determined with uniform positional winning strategies and can be solved in polynomial time in the number of edges of the underlying arena.*

0.3.4 Parity games

In Büchi games, Player 0's goal is to reach a fixed set of vertices infinitely often or at least once for co-Büchi games. The vertices of the target set have equal priorities. A generalization of that and a more expressive game is a game where each state is assigned a different priority and the Player aims to satisfy a condition on the highest priority state encountered. Such games are known as Parity games.
In a parity game, each state is assigned a color, natural numbers that represent priority. In general, Player i wins a play if and only if the minimal (or maximal depending on the type of game) encountered infinitely often is of parity i. There exist multiple configurations of parity games combining min max and parity. In the context of this report, we focus on the min even parity games during which Player 0 ensures that the minimal color seen infinitely often is even.

Parity games play a central role in the theory of infinite games and have important applications in logic and automata theory.

Definition 18 (Parity Game)**.** *Let $A = (V, V_0, V_1, E)$ be an arena and let $\Omega : V \to \mathbb{N}$ be a coloring of its vertices. The parity condition is defined as:*

$$\mathit{Parity}(\Omega) := \{\rho \in V^\omega \mid \min \mathit{Inf}(\Omega(\rho_0)\Omega(\rho_1)\Omega(\rho_2)\cdots) \text{ is } even\}$$

We call a game $G = (A, \mathit{Parity}(\Omega))$ a parity game.

Lemma 8. *Every parity condition $\mathit{Parity}(\Omega)$ is a boolean combination of Büchi conditions (thus equivalently, of co-Büchi conditions).*

Due to Büchi and co-Büchi games being expressible as two colors parity games (color 0 to vertices in F and color 1 to the rest of vertices for a Büchi game), we can make the same conclusion as with Büchi games.

Theorem 3. *Parity games are determined with uniform positional winning strategies.*

0.3.5 A parity games solver with progress measures

The idea behind progress measures is to conceptualize a witness for worst-case scenarios for Player 0 over the whole automaton. And a monotone operator that pushes the witness towards a fixed point that outlines a loss region for Player 0. This also implies the need to define values on the witness after which we are certain that a state is not desirable for our Player[6].

Definition 19 (Score Sheets)**.** *Let $G = (A, \mathit{Parity}(\Omega))$ be a parity game with vertex set V and let $n_c = |\{v \in V \mid \Omega(v) = c\}|$ denote the number of vertices labeled by color c with $d = \max\{c \in \Omega(V) \mid c \text{ odd}\}$ being the largest odd color in G.*

A score sheet s of G is a tuple of counters:

$$s = (s_1, s_3, \ldots, s_{d-2}, s_d) \in \prod_{c \in \{1,3,\ldots,d\}} [n_c + 1]$$

with the element $\top$ representing a tuple with all its counters maxed out.
The set of all score sheets of G is denoted by $\mathit{Sh}(G)$.
Score sheets are partially ordered according to their lexicographical order which represents desirability for Player 0.

Definition 20 (Score Sheet Update). *We define the update of a score sheet s by a color c.*

$$s \oplus c = \begin{cases} s \oplus c = \top \quad \textit{if } s = T \\ (s_1, \ldots, s_{c-1}, 0, \ldots, 0) & \textit{if } c \textit{ is even} \\ (s_1, \ldots, s_{c'-2}, s_{c'} + 1, 0, \ldots, 0) & \textit{if } c \textit{ is odd and } c' \textit{ is defined} \\ \top & \textit{otherwise.} \end{cases}$$

where $c' = \max c^* \leq c \mid c^*$ *odd and* $s_{c<n_c}$.

Lemma 9.

$$\forall s_0, s_1 \in S, \forall c \in C : (s_0 \leq s_1) \Rightarrow (s_0 \oplus c \leq s_1 \oplus c)$$

Where:

- S *is the set of all score sheets.*
- C *is the set of all colors.*
- $\leq$ *is the partial order on score sheets.*
- $\oplus$ *is the update operation defined for score sheets and colors.*

Definition 21 (Progress Measure). *Let G be a parity game with vertex set V. A function $\wp : V \to Sh(G)$ is a progress measure for G if*

- $\forall v \in V_0, \exists v' \in Succ(v) : \wp(v) \geq \wp(v') \oplus \Omega(v)$
- $\forall v \in V_1, \forall v' \in Succ(v) : \wp(v) \geq \wp(v') \oplus \Omega(v)$

$PM(G)$ is the set of all progress measures for G.
The evaluation $\|\cdot\| : PM(G) \to 2^V$ of a progress measure $\wp$ is defined as the set of vertices for which the score sheet isn't maxed:

$$\|\wp\| = \{v \in V \mid \wp(v) \neq \top\}$$

Definition 22 (Progress Measure Ordering). *Let $G = (A, Parity(\Omega))$ be a parity game with vertex set V and let $PG = \{\wp \mid \wp : V \to Sh(G)\}$.*
We define a partial order $\preceq$ on PG via $\wp \preceq \wp'$ if $\forall v \in V \wp(v) \leq \wp'(v)$

Definition 23 (Lift-operator). *Let $G = (A, Parity(\Omega))$ be a parity game with $A = (V, V_0, V_1, E)$. For every $v \in V$, we define a function $Lift_v : PG \to PG$ via*

$$Lift_v(\wp)(u) = \begin{cases} \wp(u) & \textit{if } u \neq v, \\ \max\{\wp(v), \min\{\wp(v') \oplus \Omega(v) \mid (v, v') \in E\}\} & \textit{if } u = v \textit{ and } u \in V_0, \\ \max\{\wp(v), \max\{\wp(v') \oplus \Omega(v) \mid (v, v') \in E\}\} & \textit{if } u = v \textit{ and } u \in V_1. \end{cases}$$

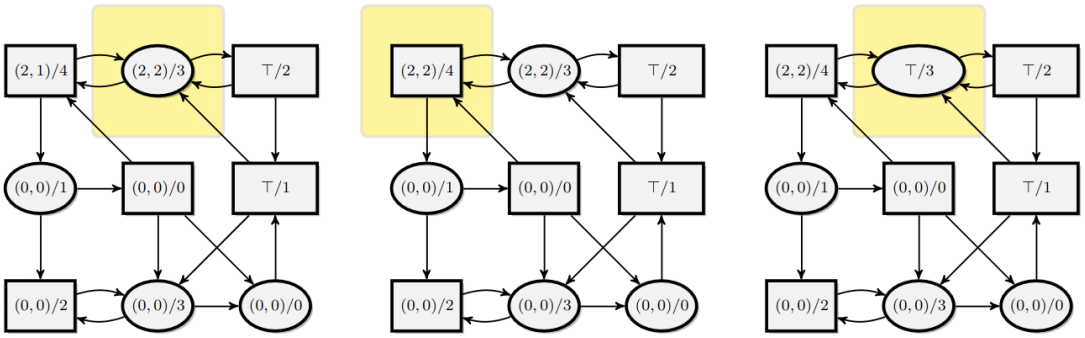

Figure 3: A few iterations from the progress measures algorithm from [7]

0.3.6 Extended progress measures for energy Büchi problems

The progress measure method for solving parity games is notable for its structure, which is particularly advantageous for algorithms operating on entire sets of states. This characteristic aligns well with our ultimate goal of solving generalized energy problems on generalized Büchi automata. Moreover, adapting this method would enable us to extend our modified version to parity games as well, thus broadening its applicability[7]. Before detailing our solution, let's first explore the mathematical structure of progress measures.

Lemma 10. *Let $G = (A, Parity(\Omega))$ be a parity game with vertex set V. Then, every lift-operator $Lift_v$ is $\preceq$-monotonic, i.e., $\wp \preceq \wp'$ implies $Lift_v(\wp) \preceq Lift_v(\wp')$.*

Lemma 11. *Let G be a parity game. Then, $(PG, \preceq)$ is a complete lattice.*

Lattices are an important structure for computation as they help define an order for events in distributed computations thus guaranteeing both robustness and termination of algorithms. They are frequently used in model checking as they are ideal for static analysis and verification of properties like correctness.

0.3.7 Lattices

Definition 24. *A lattice is a partially ordered set (poset) in which every pair of elements has a unique supremum (least upper bound) and a unique infimum (greatest lower bound). We define it as the tuple $(L, , ,)$ with:*

- *L a set of elements.*
- *$\preceq$ a partial order relation on L.*
- *(meet) is the infimum operation.*
- *(join) is the supremum operation.*

$\forall (a, b) \in L$:

- *$a \wedge b$ is the greatest element of L that is less than or equal to both a and b.*
- *$a \vee b$ is the least element of L that is greater than or equal to both a and b.*

Properties

- **Idempotent Laws**:
$$a \wedge a = a \quad \text{and} \quad a \vee a = a$$
- **Commutative Laws**:
$$a \wedge b = b \wedge a \quad \text{and} \quad a \vee b = b \vee a$$
- **Associative Laws**:
$$(a \wedge b) \wedge c = a \wedge (b \wedge c) \quad \text{and} \quad (a \vee b) \vee c = a \vee (b \vee c)$$
- **Absorption Laws**:
$$a \wedge (a \vee b) = a \quad \text{and} \quad a \vee (a \wedge b) = a$$
- **Monotonicity**: If $a \leq b$, then:
$$a \wedge c \leq b \wedge c \quad \text{and} \quad a \vee c \leq b \vee c$$

Example

The power set of $\{a, b, c\}$ ordered by set inclusion ($\subseteq$):

- $L = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.
- Meet ($\wedge$) is set intersection ($\cap$).
- Join ($\vee$) is set union ($\cup$).

Some types of lattices

- **Complete Lattice**: A lattice in which every subset has both a supremum and an infimum.
- **Bounded Lattice**: A lattice with a greatest element (top) and a least element (bottom).
- **Distributive Lattice**: A lattice that satisfies the distributive laws:
$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$
- **Complemented Lattice**: A lattice L is complemented if every element a has a complement b such that:
$$a \vee b = 1 \quad \text{and} \quad a \wedge b = 0$$
- **Boolean Lattice**: A complemented distributive lattice.

Lattices in game theory

In game theory, lattices can represent the structure of strategy spaces. In a two-player game, the set of strategy profiles (complete specification of strategies for all players in a game) can form a lattice, where the partial order represents the dominance relation between strategies. The strategy profiles in the progress measures method for parity games are encoded in the score sheets.

Tarski's Fixed Point Theorem

Theorem 4.
Let $(L, \preceq)$ be a complete lattice.
Let $f : L \to L$ be an increasing mapping.
Let F be the set (or class) of fixed points of f.
Then $(F, \preceq)$ is a complete lattice.

Let $S \subseteq F$ and $s = \bigvee S$ be the supremum of S in L. We define the upper closure of s: $U = [s, \top] = \{x \in L : s \preceq x\}$

We prove that U is closed under f:

$$\begin{aligned}
\forall a \in S : a \preceq s &\implies a = f(a) \preceq f(s) \\
&\implies s \preceq f(s) \text{ (by definition of supremum)} \\
\forall x \in U : s \preceq x &\implies f(s) \preceq f(x) \\
&\implies s \preceq f(x) \text{ (transitivity)} \\
&\implies f(x) \in U
\end{aligned}$$

Since U is a closed interval in a complete lattice, $(U, \preceq)$ is also a complete lattice. The restriction of f to U is an increasing mapping from U to U (a mapping that also preserves the order relations between the elements of the set).

This theorem is crucial for progress measures in parity games as it guarantees the algorithm's termination. It ensures that the Lift, our order-preserving function, applied to the complete lattice of score sheets, will converge to a fixed point—the solution.

0.3.8 The algorithm

Given a weighted directed graph, $G = (V, T)$ with V the finite set of vertices being the N first integers and $T \subseteq V \times N \times V$ the set of transitions $t = (v, w, v')$ a transition from v to v' with the cost of w. A finite run $r = v_0, v_1, \cdots, v_{L-1}, v_L$ is called energy feasible for the initial credit ic and the weak-upper bound b if the energy accumulated along the run never falls below zero. The corresponding energy sequence $es = e_0, e_1, \cdots, e_{L-1}$ is defined as

$$e_0 = c$$
$$e_i = \min(e_{i-1} + w, b) \text{ for } (v_{i-1}, w, v_i) \in T$$

We denote e_{L-i} the attained energy of the run.

Energy Reachability

A typical question for such graphs is the energy optimal reachability problem which answers for the maximally attainable energy for some states given a weighted graph G, an initial state $s0$, an initial credit ic and a weak-upper bound b.
As discussed in the previous chapter for Lower-weak-upper-bound problems, even in the presence of energy-positive loops, applying Bellman-Ford algorithm until a fixed point is reached yields the solution. The algorithm can be simplified as shown in Alg. 1.

Algorithm 1 Forward BF

```
E = [−1, ··· , −1]
E[x] = c
while true do
    Eo = E
    for (v, w, v′) ∈ T do
        e′ = min(Eo[v] + w, b)
        if e′ ≥ 0 then
            E[v′] = max(Eo[v′], e′)
        end if
    end for
    if E = Eo then
        break
    end if
end while
```

In Alg. 1, an energy of -1 for a state indicates that it is not energy reachable.

Minimal Initial Credit

To build a question that aims for a more descriptive solution, consider the following problem, given the graph G, some final vertex y, and a weak-upper bound b, what is the minimal initial credit required such that there exists an energy feasible run to y from some or all states? This problem can be treated as the dual of the Energy Reachability problem. Rather than calculating how far energy can spread from a starting point, we traverse the graph backward to determine the minimum energy needed at a potential starting point to reach a specific end state. Suppose we have some state v' with necessary initial energy e' and there is a transition (v, w, v'). We

want to find the necessary initial energy e in v such that a passage through the transition puts us at energy e'. That can be expressed with:

$$e = e' - w$$

One must still keep in mind that several due to asymmetry, cases can occur depending on the values w, e, and b controlling determining the value of the initial energy e:

$$e = \begin{cases} \bot & \text{if } e' - w > b \\ \max(0, e' - w) & \text{else} \end{cases} \tag{1}$$

While the second case is normal propagation with the restriction that initial credit can not be negative (as needing negative energy is nonsensical), the first case corresponds to transitions that can not be taken since the initial credit is bounded by the weak-upper bound (we would need more energy than the upper bound allows us to store).

This problem can also be solved by a modified version of Bellman-Ford as shown in Alg. 2

Algorithm 2 Forward BF

```
1:  E = [b + 1, ··· , b + 1]
2:  E[y] = 0
3:  while true do
4:      Eo = E
5:      for (v, w, v') ∈ T do
6:          e = max(0, Eo[v'] − w)
7:          if e ≤ b then
8:              E[v] = min(Eo[v], e)
9:          end if
10:     end for
11:     if E = Eo then
12:         break
13:     end if
14: end while
```

A state is assigned the necessary initial energy of $b+1$ if there is no energy-feasible path from it to the target state y. It's a value we will never encounter in a normal run, we can replace it to our convenience for application of other operations as long as it doesn't conflict with existing possible energy values.

A progress measure for Büchi automata

After establishing how we're going to reason about energy we will construct our extended progress measures for Büchi automata with, this time, a positive witness. The solution will be obtained by means of a fixed point computation as with classic progress measures with the exception of the starting point which will be the worst element for each state since we are operating with a positive witness.

Preliminaries Consider the unweighted Büchi automaton B and some finite run $r = v_0, v_1, \cdots, v_{L-2}, v_{L-1}$. Is it possible to find a simple structural condition on the run that allows us to infer that there exists a subrun $r_s = v_n, v_{n+1}, \cdots, v_{m-2}, v_m$ with $0 \leq n < m \leq L-1$

that is with certainty an accepting cycle? meaning is we have $v_n = v_m$ and there is at least one accepting transition in the subrun.
Note that we do not yet look for proof of the shortest such run nor do we demand that the cycle has to be simple. We aim to find a condition that implies an accepting cycle. We are not trying to decide if there exists a cycle on where the run the condition must hold.
Under these assumptions, we argue that a simple condition is to count the number of accepting transitions in the run r. If the latter exceeds the number of accepting transitions in B, then the run is guaranteed to include an accepting cycle as at least one accepting edge has to be visited at least twice.

We assume a finite run containing $N_{\text{acc}} + 1$ accepting transitions
$r = v_0, v_1, v_n, v_{n+1}, \ldots, v_m, v_{m+1}, \ldots, v_{L-1}$ with $n < m < L$.
Since we encountered $N_{\text{acc}} + 1$ accepting transitions, there must be one repeated accepting transition in the run.
Let's mark its first occurrence (v_n, v_{n+1}) and its second occurrence (v_m, v_{m+1}), which implies $v_n = v_m$ and $v_{n+1} = v_{m+1}$.
A finite run with a repeated state is a cycle. What's left to prove is that this cycle must contain an accepting transition.

- **Case 01**:
  $m = n + 1$, $v_m = v_{n+1}$
  $v_n = v_m = v_{n+1} = v_{m+1}$
  Our cycle is thus a self-loop containing the accepting transition (v_n, v_{n+1}).

- **Case 02**:
  $m > n + 1$
  There exists a path between v_n and v_m and it contains v_{n+1}, which means it contains the transition (v_n, v_{n+1}), implying at least one accepting transition.
  In all cases, we obtain a sub-path that is a lasso (prefix, cycle) with the prefix possibly empty and the cycle starting at the first repeated transition, which is guaranteed to happen the latest at v_n.

Simple Büchi measure for acceptance

We transform the condition detailed above into a simple measure for Büchi automata. This measure tracks the number of accepting transitions visited to help infer the existence of an accepting cycle.

The *score sheet* Sh for the simple Büchi measure is a natural number as it's essentially a counter. It's updated by incrementing whenever an accepting transition is encountered. From the previous condition we established for accepting cycles, we bound the value by the number of accepting transitions in the graph plus one. Once this value is reached, we are sure that there exists at least one accepting transition that was visited at least two times.
We denote the number of accepting transitions in B as N_{acc} and we define the following operations for updates

$$a \oplus^K b = \min(a + b, K) \; and \; a \oplus_K b = \max(a + b, K)$$

$\ominus^K$ and $\ominus_K$ are defined in a similar manner.

As before, the progress measure is defined as a function that associates a score sheet to each vertex: $\rho \colon V \mapsto Sh(B)$ with additional constraints that we will talk about in detail later.

Lift operation and algorithm

The Lift operation updates the score sheet of some vertex given the current progress measure. For just the current score sheet, it is defined as follows:

$$\text{Lift}_v(\rho)(u) = \begin{cases} \rho(v) & \text{if } u \neq v \\ \max(\{\rho(v') \oplus^{N_{acc}+1} acc \mid (v, acc, v') \in T\}) & \text{else} \end{cases} \tag{2}$$

That is the new score for some vertex is the best of the successors' scores updated with their respective transitions. Note that the value does not depend on the current value. Unlike score sheets for parity conditions that can both increase and decrease depending on the parity and the magnitude of the color we are adding, score sheets that measure the number of encountered accepting transitions can't decrease, as long as we take the best transitions the sum will be the same.

We say that ρ is a progress measure for B if it is a common fixed point for all vertices. The states that can be extended to accepting runs are those with $\rho(v) = N_{acc} + 1$. The strategy to find an accepting lasso is to prolong a run with the first (in the order of outgoing transitions) transition to a state for which $\rho(v') = N_{acc} + 1$ holds as well until a cycle is found.

Simple Weighted Büchi Measure for acceptance and energy

we will now construct a score sheet to solve the Energy Büchi Problem in its minimal initial credit form. To this end, the score sheet is extended to be a tuple: (ic, cnt_b) with ic being the necessary initial credit and cnt_b being the counter of the simple Büchi measure.

Preliminaries

Consider some finite run in a weighted Büchi automaton B_w $r = v_0, v_1, \cdots, v_{L-2}, v_{L-1}$ with a weak upper bound b.
Now let us use the score sheet of simply weighted Büchi measure on this trace, by constructing a sequence of them $seq = sc_0, sc_1, \cdots, sc_{L-2}, sc_{L-1}$. As the score sheet is *back propagated* we have

$$sc_{L-1} = (0, 0) \tag{3}$$

$$sc_{i-1} = sc_i \odot (w, acc) \text{ with } (v_{i-1}, w, acc, v_i) \in T \tag{4}$$

where $\odot$ is defined between a score sheet and an accepting condition is defined as

$$sc' = (ic', cnt'_b) = sc \odot (w, acc) = (ic, cnt_b) \odot (w, acc) = (ic - w, cnt_b + acc) \tag{5}$$

The sequence of score sheets, as constructed, allows us to draw conclusions about potential accepting runs.
As already seen, if the required initial credit is less than or equal to b, the run is considered energy feasible given an initial energy of at least ic_0 (derived from $sc_0 = (ic_0, cnt_{b0})$).
For acceptance, cnt_b keeps track of the number of accepting transitions encountered during the run. If this number exceeds N_{acc}, we can conclude that a Büchi accepting cycle exists, disregarding energy. However, this alone does not confirm the existence of an energy-feasible cycle.

To find a condition that satisfies both energy feasibility and acceptance, let's divide the run into three segments: r_{pre}, r_{cyc}, and r_{post}.

In a scenario where r_{post} does not contain any accepting transitions and requires an initial credit of 0 before it is executed, r_{cyc} represents the cycle containing all accepting transitions. The energy accumulated during a single traversal of this cycle is -1, meaning that the necessary initial credit increases by one with each traversal in the reverse direction. Lastly, r_{pre} is energy neutral and does not contain any accepting transitions. In this situation, r_{cyc} can be traversed b times before the total run becomes unfeasible and we can visit $b(N_{acc}+1)$ accepting transitions this way. Reversely, this means that if we have $(b+1)(N_{acc}+1)$ accepting transitions, then not only are we certain that we have found an accepting cycle, but also we enforce its energy feasibility by repetition.

We have already proven that a path with $N_{acc}+1$ accepting transitions is an accepting cycle. To prove that it is energy feasible, we look for a cycle in energy, meaning we cycle back to the initial credit that allowed us to take the accepting cycle. Our possible values are modulo b, and with a minimal step of 1, we can take b steps to cycle back to our starting energy. For any bigger step, we go back to the starting initial credit after $n \leq b+1$, n being the number of steps.

$$\begin{aligned} (a+nc) \mod b &= a \mod b \\ nc \mod b &= 0 \mod b \end{aligned}$$

There has to exist an nc that is divisible by b, $n = b$ at worst.

Simple Weighted Büchi Measure

We transform the condition detailed above into a simple measure for weighted Büchi automata, called simple weighted Büchi measure (swbm).
To achieve this goal, we monitor the required initial energy and the count of accepting transitions visited, similar to the approach used for the simple Büchi measure. However, due to the eventuality of two competing acceptance conditions, maintaining a single sheet per state is insufficient. This suggests the existence of a memoryless strategy, which is not feasible in this case. There may exist states that need to differentiate between an energy-optimal successor and an acceptance-optimal successor.

Upon examination of the score sheet, it becomes apparent that a definitive comparison between, for example, the score sheet $(10, 10)$ and $(5, 5)$ cannot be established as the former requires higher initial energy but allows to traverse a greater number of accepting transitions while the latter demands less energy but ensures fewer encounters with accepting transitions. Each option presents its advantages and drawbacks; these elements are incomparable. We conclude that the score sheets can only be considered partially ordered

To circumvent this problem, instead of associating a single score sheet to each state, we associate a state to a set of score sheets, reflecting the different possibilities that are still desirable. Since they are partially ordered, we can restrict ourselves to the set of Pareto optimal elements**[watanabe_Pareto]**. For example the score sheets $(10, 10)$ and $(20, 8)$ are comparable so we only need to keep $(10, 10)$. We call this Pareto optimal set of score sheets an extended score sheet (ESh).

Given a weighted Büchi automaton B_w, a simple weighted Büchi measure is a function associating an extended score sheet to each vertex together with additional constraints which will be detailed in the next part.

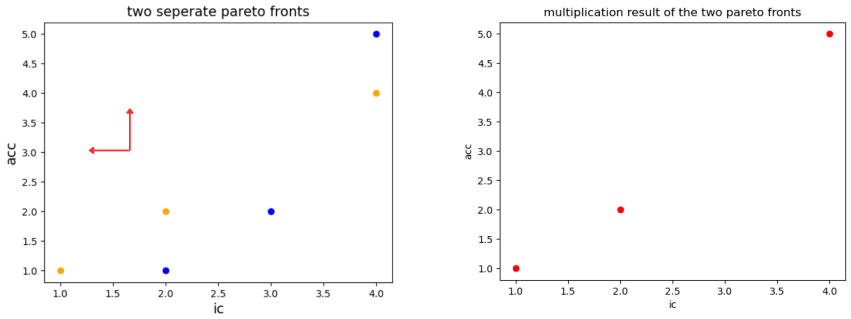

Figure 4: An example of a Pareto front join for ic and cnt_b

Propagating score sheets

(Back) propagating a score sheet sc' along a transition (v, w, acc, v') is done in the following manner:

$$sc = (ic, cnt_b) = sc' \odot_{B_w} (w, acc) = \begin{cases} (ic' \ominus_0 w, cnt'_b \oplus^{(b+1)(N_{acc}+1} acc) & \text{if } ic' \ominus_0 w \leq b \\ \bot \text{ else} \end{cases} \tag{6}$$

With $\bot$ being a special symbol representing the smallest/worst score sheet of all others.

Propagating extended score sheets

Propagating extended score sheets along an edge is done element-wise, score sheet per score sheet, we have

$$esc = esc' \odot_{B_w} (w, acc) = \{sc' \odot_{B_w} (w, acc) \min sc' \in esc'\}$$

by abuse of notation.

Lift operation and algorithm

The Lift operation is responsible for updating the extended score sheet of a specific vertex, taking into account the current progress measure. The core concept of this operation involves retaining the most favorable elements from all successors, which are propagated along their corresponding edges. It is important to note that the symbol $\bot$ remains invariant under propagation. It is defined as follows:

$$\text{Lift}_v(\rho)(u) = \begin{cases} \rho(v) & \text{if } u \neq v \\ \otimes(\{\rho(v') \odot_{B_w} (w, acc) \mid (v, w, acc, v') \in T\}) \end{cases} \tag{7}$$

Where:

- $\otimes$ denotes the operation of constructing a new Pareto front from all elements contained in a set of Pareto fronts. Again this should be monotone as it always considers all successors.
- ρ is a progress measure for B_w once a common fixed point for all vertices with respect to Lift is attained.

The measure is initialized by assigning to each state the extended score sheet $\{(0,0)\}$.

States that can be used as a starting point for a feasible run have a score sheet in their extended score sheet attaining the values described by the previous energy and acceptance conditions: a value of $(b+1)(N_{acc}+1)$ for cnt_b (a score sheet with a negative ic is maxed to 0 and a score sheet with an ic superior to b is impossible and directly set to the worst element, thus, verifying cnt_b suffices). The strategy to construct the word/run is more involved.

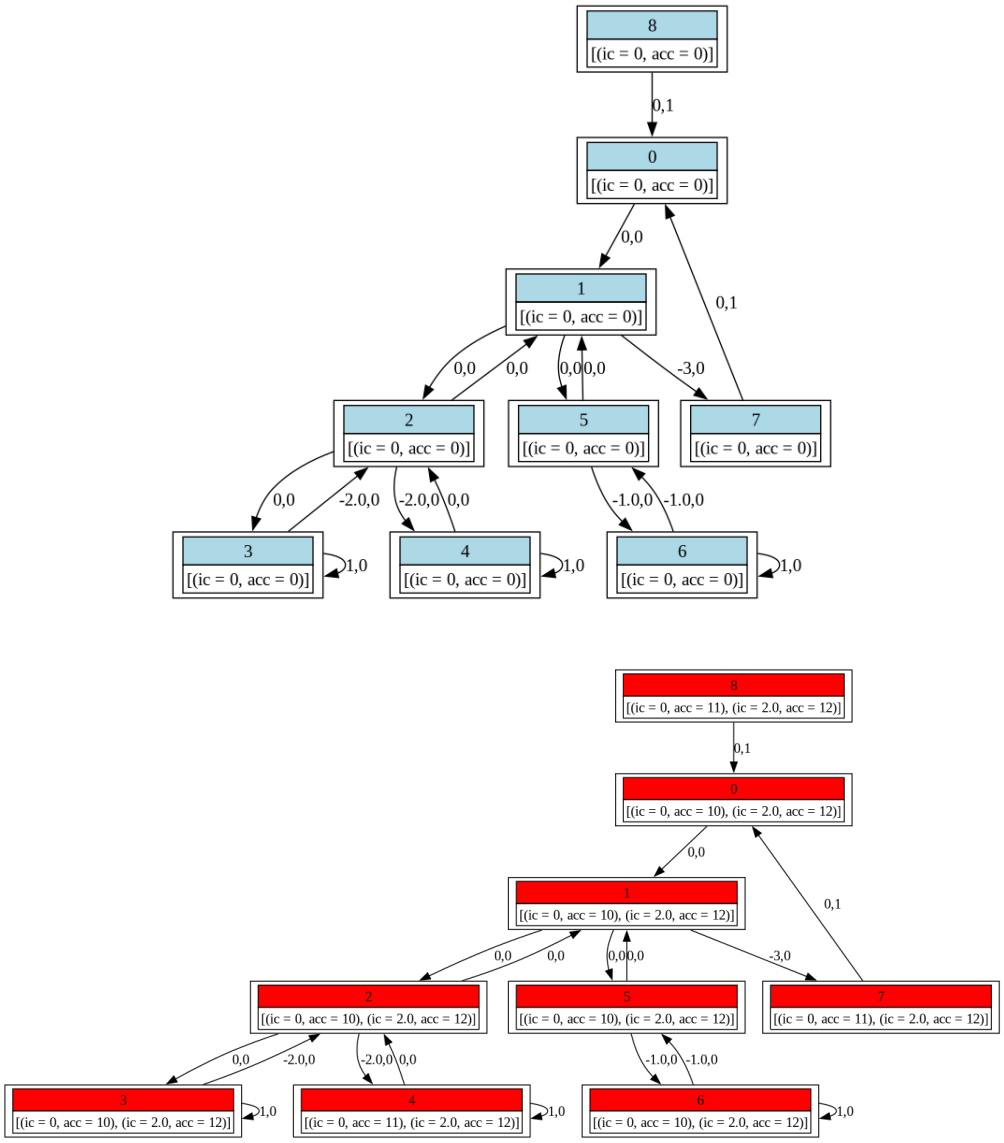

Figure 5: Extended score sheets method, initialization step, and fixed point

0.3.9 Implementation

Algorithm 3

```
PM = [[(0,0)], ..., [(0,0)]]
change = true
acceptanceBound = (wup + 1)(Nacc + 1)
while change do
    change = false
    for each v in V do
        for each (u, w, acc, v) in G do
            for each pmv in PM[v] do
                newAcc = min(acceptanceBound, pmv.acc + acc)
                newIc = max(0, pmv.ic − w)
                if newIc > wup then
                    newIc = +∞
                    newAcc = −∞
                end if
                accepted = update_Pareto_front()
                accepted = false
                for each pmu in PM[u] do
                    if newAcc > pmu.acc and newIc <= pmu.ic then
                        PM[u].remove(pmu)
                        accepted = true
                    end if
                    if newIc > pmu.ic and newAcc == pmu.acc then
                        PM[u].remove(pmu)
                        accepted = true
                    end if
                end for
                if accepted then
                    PM[u].add((newIc, newAcc))
                    change = true
                end if
            end for
        end for
    end for
end while
```

0.4 Discussion

The complexity of the extended score sheets algorithm for energy Büchi problems, for now, is proportional to the number of transitions and the weak upper bound as our fixed point is at $(b+1)(N_a cc+1)$ and since we keep incomparable score sheets in a Pareto structure if the weights on the transitions are small compared to b, we end up with a complexity approaching that of duplicating the automaton for different energy levels.

However, it is important to note the advantage of this method as it utilizes a lattice structure and the algorithm is thus guaranteed to converge to the same fixed point regardless of the back-propagation order. This feature enables parallelization, a powerful tool for segmented structures like graphs.

Furthermore, the algorithm capitalizes on the game-like interaction between the agent and the environment, making it adaptable to parity games as well. We are optimistic that further exploration of properties related to acceptance and energy will lead to a reduced bound and the development of a weak upper bound-independent algorithm that remains parallelizable.

0.5 Conclusion

In this report, we have discussed energy problems on ω-regular automata, explored the problem under a game theory lens, and presented an extended score sheet method to better adapt to generalized Büchi automata. Our method leverages a lattice structure to ensure convergence and supports parallelization, making it well-suited for large-scale and segmented structures like graphs.

The energy problem on generalized Büchi automata has been approached differently in other research. We cite the µ-calculus[8] approach[9], which is advantageous in terms of complexity as it isn't dependent on the weak upper bound.

On the other hand, our extended score sheet method is better suited for parallelization, especially in practical implementations involving large state spaces. Moreover, the integration with game-like environments allows for greater flexibility in handling diverse problem settings.

While our current method does not cover path retrieval yet, it leaves a global trace that is convenient for multiple acceptance conditions. Ongoing research aims to construct runs from this trace. Future work will focus on enhancing path retrieval capabilities and further optimizing the algorithm to maintain its parallelization and adaptability benefits.

Bibliography

[1] LRDE. *Spot - LRDE - epita*. https://www.lrde.epita.fr/wiki/Spot. Accessed: 2024-07-08.

[2] S. Dziadek, U. Fahrenberg, and P. Schlehuber-Caissier. "Regular Energy Problems". In: *Formal Aspects of Computing* 1.1 (Mar. 2024).

[3] M. Zimmermann, F. Klein, and A. Weinert. *Infinite Games Lecture Notes, Summer Term*. 2016.

[4] J. B. Nation. *Notes on Lattice Theory*. Accessed: 2024-07-08.

[5] Patricia Bouyer et al. "Infinite Runs in Weighted Timed Automata with Energy Constraints". In: *Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08)*. Saint-Malo, France, 2008, pp. 33–47. DOI: 10.1007/978-3-540-85778-5_4.

[6] Marcin Jurdziński. "Small Progress Measures for Solving Parity Games". In: *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2000)*. Ed. by Horst Reichel and Sophie Tison. Vol. 1770. Lecture Notes in Computer Science. This is a slightly revised version (July 2000). Lille, France: Springer-Verlag, Feb. 2000, pp. 290–301.

[7] K. Chatterjee and L. Doyen. "Energy Parity Games". In: *IST Austria (Institute of Science and Technology Austria), Austria; LSV, ENS Cachan & CNRS, France*. 2012.

[8] Yde Venema. *Lectures on the Modal μ-Calculus*. E-mail: y.venema@uva.nl. Science Park 107, NL–1098XG Amsterdam, 2020.

[9] Gal Amram et al. "Efficient Algorithms for Omega-Regular Energy Games". In: *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP 2024)*. Tel Aviv University, Tel Aviv, Israel; University of Leicester, Leicester, UK. Tel Aviv, Israel, 2024.