Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Universality of Büchi Automata: Analysis with Graph Neural Networks

CHRISTOPHE STAMMET[1], ULRICH ULTES-NITSCHE[1], AND ANDREAS FISCHER.[1,2]
[1]University of Fribourg, Switzerland (e-mail: firstname.lastname@unifr.ch)
[2]University of Applied Sciences and Arts Western Switzerland, Switzerland (e-mail: andreas.fischer@hefr.ch)

Corresponding author: Christophe Stammet (e-mail: christophe.stammet@unifr.ch).

ABSTRACT The universality check of Büchi automata is a foundational problem in automata-based formal verification, closely related to the complementation problem, and is known to be PSPACE-complete. This article introduces a novel approach for creating labelled datasets of Büchi automata concerning their universality. We start with small automata, where the universality check can still be algorithmically performed within a reasonable timeframe, and then apply transformations that provably preserve (non-)universality while increasing their size. This approach enables the generation of large datasets of labelled Büchi automata without the need for an explicit and computationally intensive universality check.
We subsequently employ these generated datasets to train Graph Neural Networks (GNNs) for the purpose of classifying automata with respect to their (non-)universality. The classification results indicate that such a network can learn patterns related to the behaviour of Büchi automata that facilitate the recognition of universality. Additionally, our results on randomly generated automata, which were not constructed using the transformation techniques, demonstrate the network's potential in classifying Büchi automata with respect to universality, extending its applicability beyond cases generated using a specific technique.

INDEX TERMS Automata, Computational complexity, Formal verification, Graph neural networks, Machine Learning

I. INTRODUCTION

IN 1962, J.R. Büchi introduced automata on infinitely long words [1] and showed the equivalence between the languages accepted by these so-called Büchi automata and ω-regular languages. In order to verify properties on systems, Büchi automata are used to model these systems, as the properties of infinitely long inputs nicely represent the indefinitely long running time of concurrent systems.

One of the main problems and algorithmic bottlenecks in automaton-based formal verification of systems is the complementation of Büchi automata [2], which is shown to be PSPACE-complete, with the complemetation algorithm that requires the state growth of at least $O((0.76n)^n)$ w.r.t. the number of nodes[1]. With a different approach to automaton-theoretic verification focussing on language containment [3], one can, instead of complementing an automaton, focus on universality checking [4] of Büchi automata.

The goal of this article will be to show the progress made in our current research work, with the overarching goal of analysing if Graph Neural Networks (GNNs) are able to derive properties (beyond strictly structural graph-related properties like e.g. node reachability) from Büchi automata which are encoded as directed labelled graphs with additional state labels. The classification property in question will be the aforementioned (non-)universality of the input automaton, a property directly linked to the languages that are accepted by the automaton. This research extends our previous work [5], which showed that GNN are able to derive structural patterns from automaton structures for classification of structurally and computationally trivial properties.

[1]with n being the number of nodes of the automaton to be complemented.

In the next section, we will present an overview of the required concepts and their respective definitions and how they will be used in this article. The third section will lay the formal foundations of our proposed transformation constructions allowing to generate labelled data with respect to universality. The fourth section will focus on the challenges of encoding Büchi automata as data elements for GNNs and on the creation of the different datasets used in the experimentation. In the fifth section, we will give details about the various experimental setups, both on a GNN level and on a dataset choice level and present experimental results. We will finish by discussing the results and propose future work in a concluding final section.

II. PRELIMINARIES

A. BÜCHI AUTOMATA

The literature defines multiple types of automata on infinite structures [6], but this article will focus on non-deterministic Büchi automata (NBWs):

Definition 1. *Let NBW $\mathcal{A}$ be defined as a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where*

- *Q represents a finite set of states,*
- *Σ represents a finite set of symbols (called an alphabet),*
- *$\delta : Q \times \Sigma \longrightarrow 2^Q$ represents the transition function*[2]*,*
- *$q_0 \in Q$ represents the initial state and*
- *$F \subseteq Q$ represents the set of accepting states.*

We will define the behaviour of this automaton over infinitely long sequences of symbols from the alphabet Σ called ω-words. Such a word will produce one or more so-called runs over a given automaton. Before we continue, let us formally define ω-words, runs over NBWs, and the concept of infinitely many occurrences in an infinite sequence:

Definition 2. *Given a NBW $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, an ω-word w is defined as*

$$w = w(0)w(1)w(2)\ldots \text{, where } w(i) \in \Sigma, \forall i \in \mathbb{N}.$$

Next, a run r over w on $\mathcal{A}$ is defined as an infinite sequence of states

$$r = r(0)r(1)r(2)\ldots \text{, where } r(0) = q_0 \text{ and}$$
$$\forall i \in \mathbb{N} : r(i) \in Q \text{ and } r(i+1) \in \delta(r(i), w(i)).$$

Let the set of infinitely often visited states in a run, i.e. in an infinite sequence of states, be defined as

$$\mathit{inf}(r) = \{q \in Q | \exists^\infty i \in \mathbb{N} : r(i) = q\},$$

where $\exists^\infty i$ denotes the existence of infinitely many i.

Before we can illustrate how the behaviour of a NBW can be expressed as a set of ω-words, we will need to define the concept of acceptance of a run:

Definition 3. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a NBW and let r be a run on $\mathcal{A}$. Then r is called accepting if and only if*

$$\mathit{inf}(r) \cap F \neq \emptyset.$$

An ω-word is accepted by $\mathcal{A}$ if there exists an accepting run r of $\mathcal{A}$ on w.

Büchi automata can be represented graphically as state machines, with states illustrated by circles, accepting states by double-lined circles, the initial state with an unlabelled arrow pointing to it and transitions by labelled arrows. We can now look at the example automaton $\mathcal{A}_{ex1}$ from Fig. 1 and its behaviour on two example ω-words:

[2]where 2^Q denotes the powerset of the set of states Q

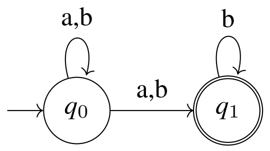


FIGURE 1. Example automaton $\mathcal{A}_{ex1}$, with $\Sigma = \{a, b\}$.

$$w_1 = abababab\ldots = (ab)^\omega,$$
$$w_2 = bbbbbbbb\ldots = b^\omega.$$

We can see that over w_1 there is only one possible run on $\mathcal{A}_{ex1}$, which is the run $r_{w_1} = q_0q_0q_0q_0\ldots$, the one staying in q_0 forever. Thus, from $\mathit{inf}(r_{w_1}) = \{q_0\}$ and $F = \{q_1\}$, we can conclude that r_{w_1} is non-accepting, and since it is the only run over w_1, by consequence w_1 is not accepted by $\mathcal{A}_{ex1}$. Over w_2 there are infinitely many runs on $\mathcal{A}_{ex1}$, depending on the non-deterministic jump from q_0 to q_1. Even though a non-accepting run is possible (staying in q_0 forever) this time we can also find accepting runs, e.g. $r_{w_2} = q_0q_0q_1q_1q_1\ldots$, concluding that w_2 is accepted by $\mathcal{A}_{ex1}$.

Definition 4. *Given a NBW $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, we define the language $\mathcal{L}_\omega(\mathcal{A})$ accepted by the automaton $\mathcal{A}$ as*

$$\mathcal{L}_\omega(\mathcal{A}) = \{w \in \Sigma^\omega | w \text{ accepted by } \mathcal{A}\}.$$

Following from [1], given any NBW $\mathcal{A}$, the language $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular.

Looking back at the example from Fig. 1, we can now reason about all the words that are accepted by this automaton, i.e. we can see that these words form the language

$$\mathcal{L}_\omega(\mathcal{A}_{ex1}) = \{w \in \Sigma^\omega | w \text{ contains finitely many } a\}$$
$$= (a|b)^* b^\omega.$$

This article will focus on the universality property of Büchi automata, an important property in automaton-based verification [4], defined as follows:

Definition 5. *Given a NBW $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, we say that $\mathcal{A}$ is called universal if and only if*

$$\mathcal{L}_\omega(\mathcal{A}) = \Sigma^\omega.$$

Contrarily, $\mathcal{A}$ is called empty if and only if

$$\mathcal{L}_\omega(\mathcal{A}) = \emptyset.$$

As mentioned in the introduction, the automata-theoretic approach to verification reduces the property satisfaction problem to language containment [3], where complementing a NBW becomes the computational bottleneck. There are many approaches to complementing NBW [7], [8].

The complementation problem requires two automata as input, and then checks whether one is the complement of the other. In order to avoid having two automata as input, we are considering in this article the closely related problem of universality, that is computationally equally hard, i.e. PSPACE-complete [9]. In the universality problem, one checks whether or not a Büchi automaton given as input is universal. The relation to complementation stems from the relation: A is universal if and only if its complement Büchi-accepts the empty ω-language.

B. GRAPH NEURAL NETWORKS

In the past decade, the advent of deep neural networks has led to performance breakthroughs for numerous applications of machine learning and pattern recognition, including image classification [10], speech recognition [11], translation [12], and bioinformatics [13], to name just a few. Depending on the type of data representation, different network architectures have been proposed, notably convolutional neural networks (CNN) [14] for images, recurrent neural networks (RNN) [15] and transformers [16] for sequences, and graph neural networks (GNN) [17] for graphs. The latter are a natural choice for analyzing and classifying Büchi automata, since the automata can be represented in a straight-forward manner by means of labelled graphs (see Section IV-A). More specifically, in this article we focus on Relational Graph Convolutional Networks (RGCN) [18], which are ideally suited for graphs with a discrete set of edge labels (in our case: the symbols in Σ). The layers of an RGCN are described in more detail in the following section.

1) Relational Graph Convolutional Layers

The following notation will be used for the RGCN input data. Let a graph $G = (V, E, \mathcal{R})$ be represented by a set of edge relations $\mathcal{R}$, a set of nodes V (with $n = |V|$ the number of nodes) and a set of edges E (with $m = |E|$ the number of edges). Let $(v_i, r, v_j) \in E$, for $v_i, v_j \in V$ and $r \in \mathcal{R}$, be a directed edge pointing from v_i to v_j belonging to relation r. We define the neighbourhood of a node $v \in V$ for an edge label $r \in \mathcal{R}$ as follows:

$$N_r(v) = \{u \in V | (v, r, u) \in E\}.$$

Let $\mathbf{X}_l \in \mathbb{R}^{n\times d^{(l)}}$ be the node feature matrix, with $d^{(l)}$ denoting the number of node features at layer l.

An RGCN layer updates each node's feature representation by aggregating information from its neighbours in the graph. Let $\mathbf{h}_v^{(l)} \in \mathbb{R}^{d^{(l)}}$ be the feature vector of node v at layer l. The RGCN layer computes a new feature vector $\mathbf{h}_v^{(l+1)} \in \mathbb{R}^{d^{(l+1)}}$ for node v at layer $l+1$ as follows:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\left(\sum_{r\in\mathcal{R}} \frac{1}{c_{r,v}} \sum_{u\in N_r(v)} \mathbf{W}_r^{(l)}\mathbf{h}_u^{(l)}\right) + \mathbf{W}_0^{(l)}\mathbf{h}_v^{(l)}\right), \tag{1}$$

where σ is an activation function, $\mathbf{W}_r^{(l)} \in \mathbb{R}^{d^{(l+1)}\times d^{(l)}}$ is the weight matrix for relation type r at layer l, $\mathbf{W}_0^{(l)} \in \mathbb{R}^{d^{(l+1)}\times d^{l}}$ is a self-loop weight matrix of a special relation type accounting for the node's own feature representation, and $c_{r,v}$ is a normalization constant.

2) Graph level classification

With $G = (V, E, \mathcal{R})$ being our graph, let $\mathbf{h}_v^{(L)}$ be the final feature vector of node v after L layers of the RGCN. The global pooling operation computes a graph-level representation $\mathbf{h}_G \in \mathbb{R}^{d^{(L)}}$ by applying an aggregation function g to the node representations:

$$\mathbf{h}_G = g\left(\mathbf{h}_v^{(L)} | v \in \mathcal{V}\right) \tag{2}$$

This feature vector may be further processed using a final linear transformation to obtain a vector of probabilities for each class. Specifically, we use a linear layer to map the graph-level feature vector $\mathbf{h}_G$ to a vector $\mathbf{z} \in \mathbb{R}^s$:

$$\mathbf{z} = \mathbf{W_f}\mathbf{h}_G + \mathbf{b}, \tag{3}$$

where $\mathbf{W_f} \in \mathbb{R}^{s\times d^L}$ is the final weight matrix, $\mathbf{b} \in \mathbb{R}^s$ is a bias vector and s is the number of classes in the dataset. The output vector $\mathbf{z}$ can be interpreted as the confidence of the graph belonging to each class, with a normalizing softmax function yielding the probability distribution vector $\hat{y} \in \mathbb{R}^s$ as follows:

$$\hat{y} = \left[\frac{e^{\mathbf{z}_1}}{\sum_{j=1}^{s} e^{\mathbf{z}_j}}, \frac{e^{\mathbf{z}_2}}{\sum_{j=1}^{s} e^{\mathbf{z}_j}}, \ldots, \frac{e^{\mathbf{z}_s}}{\sum_{j=1}^{s} e^{\mathbf{z}_j}}\right] \tag{4}$$

3) Network training

The goal of the training of an RGCN model is to optimize the trainable parameters ($\mathbf{W}_r^{(l)}, \mathbf{W}_0^{(l)}, \mathbf{W_f}$ and $b, \forall r \in \mathcal{R}$ and $l \in \mathbb{N} : 0 \leq l \leq L$) of the RGCN layers by minimizing the difference between the predicted graph-level output $\hat{y}$ and the ground-truth label y.

To do this, we define the cross-entropy loss function L (where, for an $i \in \mathbb{N}$, y_i denotes the i^{th} element of vector y) as follows:

$$L = -\sum_{c=1}^{s} y_c \log \hat{y}_c, \tag{5}$$

which captures the discrepancy between the predicted label and the true label. To reduce this difference between predicted and true label of the training graph, we will compute the gradient of the loss function for each parameter as follows:

$$\frac{\partial L}{\partial \mathbf{W}_r^{(l)}} = -\sum_{c=1}^{s} \frac{y_c}{\hat{y}_c} \frac{\partial \hat{y}_c}{\partial \mathbf{W}_r^{(l)}}. \tag{6}$$

The gradients for the other learnable parameters $\mathbf{W}_0^{(l)}$, $\mathbf{W_f}$ and b are computed similarly and are omitted here. Finally, an optimization algorithm using e.g. stochastic gradient descent is applied to update the parameters using the following rule (again only for $\mathbf{W}_r^{(l)}$):

$$\mathbf{W}_r^{(l+1)} = \mathbf{W}_r^{(l)} - \eta \frac{\partial L}{\partial \mathbf{W}_r^{(l)}}, \tag{7}$$

where η represent the learning rate, a parameter defining the step size of the learning, i.e. the ratio of parameter update with respect to its gradient.

III. NBW TRANSFORMATIONS

This section will introduce the concepts of preserving transformations on Büchi automata that do not violate universality (resp. non-universality), i.e. if such a transformation is applied on an universal automaton $\mathcal{A}$, then the universality of the automaton will be preserved after that transformation. We define these transformations as follows:

Definition 6. *Let BA be the set of all Büchi automata and P be the set of all transformation parameters. Then in general, a preserving transformation*

$$\mathfrak{T} : BA \times 2^P \rightarrow BA$$

*is a function that performs a transformation with the given parameters on the given Büchi automaton and outputs the transformed automaton (output automaton will be universal **if and only if** the input automaton was universal).*

There also exist transformations that only preserve universality or non-universality, which leads to the following definitions: For transformations preserving universality, we get

$$\mathfrak{T}^u : BA \times 2^P \rightarrow BA.$$

Similarly, the transformations preserving non-universality can be defined as follows:

$$\mathfrak{T}^n : BA \times 2^P \rightarrow BA.$$

The following subsections introduce preserving transformations and prove that they preserve (non-)universality. Here is an overview of all the transformations:

- Transformations preserving universality:
  - **Add state:** A new non-accepting state is added to the given automaton.
  - **Add transition:** A new transition is added to the given automaton.
  - **Make state accepting:** A non-accepting state is being made accepting.
- Transformations preserving non-universality[3]:
  - **Remove transition:** An existing transition is removed from the given automaton.
  - **Remove acceptance from state:** An accepting state is being made non-accepting.
- Preserving transformations:
  - **Split self-loop:** A self-loop transition loops through a newly introduced state and back.
  - **Expand self-loop:** New states are added to the automaton and a self-loop is replaced by connecting its state to the added states.
  - **Duplicate strongly connected component:** Copies an existing SCC and its behaviour and adds an interim accepting state.
  - **Duplicate transition behaviour:** Copies destination state of an existing transition and copies the destination state's outgoing transition behaviour.

[3]We do not consider removing a state as we are interested in transformations that at least preserve the number of states.

A. TRANSFORMATIONS PRESERVING UNIVERSALITY

To start off, we will introduce transformations that are guaranteed to preserve universality. The goal here is simple, add elements to the automaton without changing the states and transitions that are already present. This guarantees that all words that are accepted by the original automaton will also be accepted by the transformed one, i.e. preserving universality.

Definition 7. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton*

- *Let $q_+ \notin Q$ be a newly introduced state. The **TPU "adding a state"** $\mathfrak{T}^u_s$ is defined as follows:*

$$\mathfrak{T}^u_s(\mathcal{A}, \{q_+\}) = (Q \cup \{q_+\}, \Sigma, \delta, q_o, F)$$

- *Let $q_{src}, q_{dest} \in Q$ be states of $\mathcal{A}$ and $c \in \Sigma$. Then the **TPU "adding a transition"** $\mathfrak{T}^u_t$ is defined as follows:*

$$\mathfrak{T}^u_t(\mathcal{A}, [q_{src}, q_{dest}, c]^4) = (Q, \Sigma, \delta_+, q_0, F),$$

where

$$\delta_+(q, a) = \begin{cases} \delta(q, a) \cup \{q_{dest}\} & \textit{if } q = q_{src}, a = c \\ \delta(q, a) & \textit{otherwise} \end{cases}$$

- *Let $q_a \in Q$ be a state of $\mathcal{A}$. Then the **TPU "make accepting"** $\mathfrak{T}^u_a$ is defined as follows:*

$$\mathfrak{T}^u_a(\mathcal{A}, \{q_a\}) = (Q, \Sigma, \delta, q_o, F \cup \{q_a\})$$

All 3 transformations that strictly preserve universality are relatively straightforward, and the proofs that all of them indeed preserve universality are trivial[5], so the following lemma will be given without proof:

Lemma 1. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a universal Büchi automaton. Then*

- *$\mathfrak{T}^u_s(\mathcal{A}, \{q_+\})$ is universal, for $q_+ \notin Q$.*
- *$\mathfrak{T}^u_t(\mathcal{A}, [q_{src}, q_{dest}, c])$ is universal, $\forall q_{src}, q_{dest} \in Q$, $\forall c \in \Sigma$.*
- *$\mathfrak{T}^u_a(\mathcal{A}, \{q_a\})$ is universal, $\forall q_a \in Q$.*

B. TRANSFORMATIONS PRESERVING NON-UNIVERSALITY

The transformations in this subsection will guarantee that a non-universal input automaton cannot be made universal after applying these transformations, so as long as it is impossible that new words are added to the language accepted by the input automaton, these transformations are guaranteed to be preserving.

Definition 8. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton*

- *Let $q_{src}, q_{dest} \in Q$ be states of $\mathcal{A}$ and $c \in \Sigma$. Then the **TPN "removing a transition"** $\mathfrak{T}^n_t$ is defined as follows:*

$$\mathfrak{T}^n_t(\mathcal{A}, [q_{src}, q_{dest}, c]) = (Q, \Sigma, \delta_-, q_0, F),$$

where

$$\delta_-(q, a) = \begin{cases} \delta(q, a) \setminus \{q_{dest}\} & \textit{if } q = q_{src}, a = c \\ \delta(q, a) & \textit{otherwise} \end{cases}$$

[4]Here, [...] denotes an ordered set of elements.

[5]Every single run on any word that is present in the original automaton is also present in the transformed automaton.

- *Let $q_a \in Q$ be a state of $\mathcal{A}$. Then the **TPN "remove accepting"** $\mathfrak{T}_a^n$ is defined as follows:*

$$\mathfrak{T}_a^n(\mathcal{A}, \{q_a\}) = (Q, \Sigma, \delta, q_o, F \setminus \{q_a\})$$

Similarly to the case of the universal automata, both these transformations can easily be shown to preserve non-universality, proving the following lemma:

Lemma 2. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a non-universal Büchi automaton. Then*

- *$\mathfrak{T}_t^n(\mathcal{A}, [q_{src}, q_{dest}, c])$ is non-universal, $\forall q_{src}, q_{dest} \in Q$, $\forall c \in \Sigma$*
- *$\mathfrak{T}_a^n(\mathcal{A}, \{q_a\})$ is non-universal, $\forall q_a \in Q$*

C. PRESERVING TRANSFORMATIONS

We will now introduce transformations that are even language preserving, i.e. the language accepted by the transformed automaton is identical to the language accepted by the input automaton. From this follows that these transformations are preserving both universality and non-universality. The goal here is to change components or add components to the automata, while assuring that all runs that are added resp. changed do not change the behaviour of the automaton. To start, we will have a transformation that splits self-loops over states:

Definition 9. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton, let $q_{sp} \in Q$ be a state of $\mathcal{A}$ and let $q_+ \notin Q$ be a newly introduced state. Then the **TP "split loop"** $\mathfrak{T}_l$ is defined as follows:*

$$\mathfrak{T}_l(\mathcal{A}, \{q_{sp}\}) = (Q \cup \{q_+\}, \Sigma, \delta_{sp}, q_0, F_{sp}),$$

where

$$F_{sp} = \begin{cases} F & \text{if } q_{sp} \notin F \\ (F \setminus \{q_{sp}\}) \cup \{q_+\} & \text{if } q_{sp} \in F \end{cases}$$

and, for all $q \in Q \cup \{q_+\}$ and $a \in \Sigma$:

$$\delta_{sp}(q, a) = \begin{cases} \delta(q, a) \cup \{q_+\} & \text{if } q \neq q_{sp} \wedge q_{sp} \in \delta(q, a) \\ (\delta(q, a) \setminus \{q_{sp}\}) \cup \{q_+\} & \text{if } q = q_{sp} \wedge q_{sp} \in \delta(q, a) \\ \delta(q_{sp}, a) & \text{if } q = q_+ \\ \delta(q, a) & \text{else} \end{cases}$$

The effects of this transformation can be seen in Fig. 2.

Let us now prove that this transformation is indeed preserving. The following Theorem will show language equality of the original automaton and the transformed one (by corollary also proving its preserving nature).

Theorem 1. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton. For a $q_{sp} \in Q$, let $\mathcal{A}_l = \mathfrak{T}_l(\mathcal{A}, \{q_{sp}\})$*

Then
$$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}_l)$$

Proof: To show ω-language equality, we will show both $\mathcal{L}_\omega(\mathcal{A}) \subseteq \mathcal{L}_\omega(\mathcal{A}_l)$ and $\mathcal{L}_\omega(\mathcal{A}) \supseteq \mathcal{L}_\omega(\mathcal{A}_l)$:

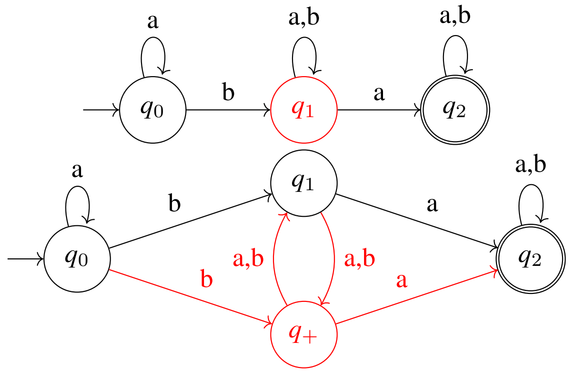


FIGURE 2. With $\Sigma = \{a, b\}$, $\mathcal{A}$ on the top and $\mathcal{A}_l = \mathfrak{T}_l(\mathcal{A}, \{q_1\})$ on the bottom.

- $\mathcal{L}_\omega(\mathcal{A}) \subseteq \mathcal{L}_\omega(\mathcal{A}_l)$

  Let $w = w_0 w_1 w_2 ... \in \mathcal{L}_\omega(\mathcal{A})$ and let $r_w = r_w(0) r_w(1) r_w(2) r_w(3) ...$ be an accepting run of $\mathcal{A}$ over w. We will show that from r_w we can construct an accepting run r'_w of $\mathcal{A}_l$ over w.
  Let r'_w be the run such that $r'_w(i) = r_w(i), \forall i \in \mathbb{N}$, such that $r_w(i) \neq q_{sp}$. For handling the occurrences of q_{sp} in r_w, we will separate two cases, maximal subruns of only q_{sp} in a run and a maximal suffix of only q_{sp}. For all the occurrences of either of these two cases in r_w:

  1) let $i \in \mathbb{N}$ such that $\forall k \geq i$: $r_w(k) = q_{sp}$ Then by construction, let

  $$\forall k \geq i : r'_w(k) = \begin{cases} q_+ & \text{if } k - i \text{ is even} \\ q_{sp} & \text{if } k - i \text{ is odd} \end{cases}$$

  2) let $i, j \in \mathbb{N}$ such that $\forall k \in [i, j]$: $r_w(k) = q_{sp}$ Then by construction, let

  $$\forall k \in [i, j] : r'_w(k) = \begin{cases} q_+ & \text{if } k - i \text{ is even} \\ q_{sp} & \text{if } k - i \text{ is odd} \end{cases}$$

  The resulting run r'_w is, by construction, a run of $\mathcal{A}_l$ over w. Furthermore, from

  $$q \in inf(r'_w) \Leftrightarrow q \in inf(r_w), \forall q \in Q \setminus \{q_{sp}\}$$

  and
  $$q_+ \in inf(r'_w) \Leftrightarrow q_{sp} \in inf(r_w)$$

  follows that $F_{sp} \cap inf(r'_w) \neq \emptyset$, since if an accepting state other than q_{sp} is visited infinitely often by r_w, it will so too by r'_w and if only q_{sp} is accepting and is visited infinitely often by r_w, the accepting state q_+ will also be visited infinitely often by r'_w.

- $\mathcal{L}_\omega(\mathcal{A}) \supseteq \mathcal{L}_\omega(\mathcal{A}_l)$

  Let $w = w_0 w_1 w_2 ... \in \mathcal{L}_\omega(\mathcal{A}_l)$ and let be $r_w = r_w(0) r_w(1) r_w(2) ...$ be an accepting run of $\mathcal{A}_l$ over w. By definition, $\exists r'_w$ run of $\mathcal{A}$ over w s.t. $\forall i \in \mathbb{N}$:

  $$r'_w(i) = \begin{cases} r_w(i) & \text{if } r_w(i) \neq q_+ \\ q_{sp} & \text{if } r_w(i) = q_+ \end{cases}$$

From $q_+ \in F_{sp} \Rightarrow q_{sp} \in F$ follows that r'_w is guaranteed to visit an accepting state infinitely often and thus $w \in \mathcal{L}_\omega(\mathcal{A})$.

■

Let us now continue with a transformation that replaces a chosen state q_c with a self-loop by a number of states that will have the same ingoing and outgoing transitions as q_c but that will create a SCC with the self-loop transitions in order to grow the size of the automaton but without changing the behaviour. Formally:

Definition 10. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton, let $q_c \in Q$ be an state of $\mathcal{A}$. Let $x \in \mathbb{N}$ be the number of states to be added and*

$$Q_+ = \{q_c\} \cup \{q_+^i | 1 \le i \le x\}$$

be the set consisting of the chosen q_c and of newly added states, i.e. $Q_+ \cap Q = \{q_c\}$. Let Σ_{sl} the symbols that create a self-loop on q_c, i.e.

$$\Sigma_{sl} = \{a \in \Sigma | q_c \in \delta(q_c, a)\}.$$

Now, let

$$\delta_+ : Q_+ \times \Sigma_{sl} \longrightarrow Q_+$$

be a given complete and deterministic transition function for the states in Q_+ on Σ_{sl}.

*Then the **TP "expand self-loop"** $\mathfrak{T}_x$ is defined as follows:*

$$\mathfrak{T}_x(\mathcal{A}, \{q_c, x, \delta_+\}) = (Q_x, \Sigma, \delta_x, q_0, F_x),$$

where

$$Q_x = Q \cup Q_+,$$

$$F_x = \begin{cases} F & \text{if } q_c \notin F \\ F \cup Q_+ & \text{if } q_c \in F \end{cases}$$

and $\delta_x(q, a) =$

$$\begin{cases} \delta(q, a) \cup Q_+ & \text{if } q_c \in \delta(q, a) \text{ and } q \notin Q_+ \\ \delta_+(q, a) \cup (\delta(q_c, a) \setminus \{q_c\}) & \text{if } q \in Q_+ \text{ and } a \in \Sigma_{sl} \\ \delta(q_c, a) \setminus \{q_c\} & \text{if } q \in Q_+ \text{ and } a \notin \Sigma_{sl} \\ \delta(q, a) & \text{else} \end{cases}$$

In Fig. 3 [6] (where the double resp. wavy edges represent a same set of in- resp. outgoing edges), we can see a snippet from an arbitrary automaton to see how this transformation works. This transformation allows the runs that would stay in q_c (either finitely long or until infinity) to jump around between q_c and all the newly introduced states, who all share the same outgoing transitions, so the runs do not change the runs that are visiting q_c. Let us now formally prove that this transformation guarantees language equivalence:

Theorem 2. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton. For a $q_c \in Q$ and a transition function between these states*

[6] where $q_+(q_c, a) = q_+^1$, $q_+(q_c, b) = q_+^2$, $q_+(q_+^1, a) = q_c$, $q_+(q_+^1, b) = q_+^2$, $q_+(q_+^2, a) = q_+^1$, $q_+(q_+^2, b) = q_+^2$

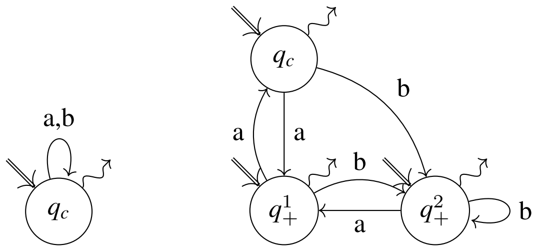


FIGURE 3. With $\Sigma = \{a, b\}$, $\mathcal{A}$ on the left and $\mathcal{A}_x = \mathfrak{T}_x(\mathcal{A}, \{q_c, 2, \delta_+\})$ on the right

δ_+, let $\mathcal{A}_x = \mathfrak{T}_x(\mathcal{A}, \{q_c, x, \delta_+\}) = (Q_x, \Sigma, \delta_x, q_0, F_x)$

Then

$$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}_x)$$

Proof: To show ω-language equality, we will show both $\mathcal{L}_\omega(\mathcal{A}) \subseteq \mathcal{L}_\omega(\mathcal{A}_x)$ and $\mathcal{L}_\omega(\mathcal{A}) \supseteq \mathcal{L}_\omega(\mathcal{A}_x)$:

- $\mathcal{L}_\omega(\mathcal{A}) \subseteq \mathcal{L}_\omega(\mathcal{A}_x)$
  Let $w = w_0 w_1 w_2 ... \in \mathcal{L}_\omega(\mathcal{A})$ and let $r_w = r_w(0) r_w(1) r_w(2) ...$ be an accepting run of $\mathcal{A}$ over w. We will show that from r_w we can construct an accepting run r'_w of $\mathcal{A}_x$ over w.
  For all $i \in \mathbb{N}$, we can construct a r'_w such that we have that

  $$r'_w(i) = \begin{cases} r_w(i) & \text{if } r_w(i) \neq q_c \\ q_+, \text{ where } q_+ \in Q_+ & \text{if } r_w(i) = q_c \end{cases}$$

  Proof of the second case of this case separation:
  There are two possibilities for the predecessor of $r_w(i)$:
  - $r_w(i-1) \neq q_c$. Then $r_w(i-1) \notin Q_+$ and since $r_w(i) = q_c$, by transition rule (1), we have that

    $$Q_+ \cap \delta_x(r_w(i-1), w(i-1)) \neq \emptyset.$$

  - $r_w(i-1) = q_c$. From this, from $w(i-1) \in \Sigma_{sl}$ and from δ_+ being complete and deterministic follows from transition rule (2) that

    $$Q_+ \cap \delta_x(r_w(i-1), w(i-1)) \neq \emptyset.$$

  To show that r'_w is accepting, it suffices to see that if an accepting state other than q_c is visited infinitely often by r_w over $\mathcal{A}$, it also will be by r'_w over $\mathcal{A}_x$.
- $\mathcal{L}_\omega(\mathcal{A}) \supseteq \mathcal{L}_\omega(\mathcal{A}_x)$

  Let $w = w_0 w_1 w_2 ... \in \mathcal{L}_\omega(\mathcal{A}_x)$ and let be $r_w = r_w(0) r_w(1) r_w(2) ...$ be an accepting run of $\mathcal{A}_x$ over w. By definition, $\exists r'_w$ run of $\mathcal{A}$ over w s.t. $\forall i \in \mathbb{N}$:

  $$r'_w(i) = \begin{cases} r_w(i) & \text{if } r_w(i) \notin q_+ \\ q_c & \text{if } r_w(i) \in Q_+ \end{cases}$$

  From, $\forall q_+ \in Q_+, q_+ \in F_x \Rightarrow q_c \in F$ follows that r'_w is guaranteed to visit an accepting state infinitely often and thus $w \in \mathcal{L}_\omega(\mathcal{A})$.

■

The next transformation will generate an exact copy of a strongly connected component of the automaton and a newly introduced accepting interim state that will be able to be visited only once, thus not changing the language of the automaton. An example will follow after the formal definition:

Definition 11. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton, let $C \subseteq Q$ be an SCC of $\mathcal{A}$, let $q_c \in C$ be a state in C, let $a_c \in \Sigma$ be a symbol with an outgoing transition from q_c, i.e. $\delta(q_c, a_c) \neq \emptyset$. We introduce a copy $\widehat{C}$*[7] *of the given SCC and an accepting interim state q_+ (with q_+ and none of the states from $\widehat{C}$ in Q). Then the **TP "duplicate SCC"** $\mathfrak{T}_d$ is defined as follows:*

$$\mathfrak{T}_d(\mathcal{A}, \{C, q_c, a_c\}) = (Q_d, \Sigma, \delta_d, q_0, F_d),$$

where

$$Q_d = Q \cup \widehat{C} \cup \{q_+\},$$

$$F_d = F \cup \{\widehat{q} | q \in C \cap F\} \cup \{q_+\}$$

and

$$\delta_d(q, a) = \begin{cases} \{\widehat{p} | p \in \delta(q, a)\} & \text{if } q \in \widehat{C} \\ \delta(q, a) \cup \{q_+\} & \text{if } q = q_c \text{ and } a = a_c \\ \overline{Q_c} \cup Q_c & \text{if } q = q_+ \\ \delta(q, a) & \text{else} \end{cases}$$

with

$$\overline{Q_c} = \{\widehat{q} | q \in \bigcup_{q_s \in \delta(q_c, a_c)} \delta(q_s, a) \cap C\}$$

$$Q_c = \{q | q \in \bigcup_{q_s \in \delta(q_c, a_c)} \delta(q_s, a) \cap (Q \setminus C)\}$$

Before we have a look at the preserving properties of this construction, let us illustrate the transformation with an example in Fig. 4.

This transformation is also a language preserving transformation, which we are going to prove now:

Theorem 3. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton. For a strongly connected component $C \subseteq Q$, $q_c \in C$ and $a_c \in \Sigma$, let $\mathfrak{T}_d(\mathcal{A}, \{C, q_c, a_c\}) = (Q_d, \Sigma, \delta_d, q_0, F_d)$*

Then

$$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}_d)$$

Proof: To show ω-language equality, we will show both $\mathcal{L}_\omega(\mathcal{A}) \subseteq \mathcal{L}_\omega(\mathcal{A}_x)$ and $\mathcal{L}_\omega(\mathcal{A}) \supseteq \mathcal{L}_\omega(\mathcal{A}_x)$:

- $\mathcal{L}_\omega(\mathcal{A}) \subseteq \mathcal{L}_\omega(\mathcal{A}_d)$
  Let $w = w_0 w_1 w_2 ... \in \mathcal{L}_\omega(\mathcal{A})$ and let $r_w = r_w(0) r_w(1) r_w(2) ...$ be an accepting run of $\mathcal{A}$ over w. Then, by construction, there exists a run $r'_w = r'_w(0) r'_w(1) r'_w(2) ...$ of $\mathcal{A}'$ over w such that

  $$r'_w(i) = r_w(i)\ , \forall i \in \mathbb{N}.$$

[7]with, for S any set of states: $\widehat{S} = \{\widehat{q} | q \in S\}$

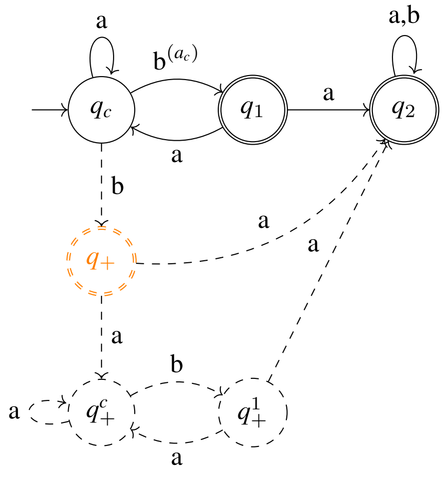


FIGURE 4. With $\Sigma = \{a, b\}$, $\mathcal{A}_d = \mathfrak{T}_d(\mathcal{A}, \{C = \{q_c, q_1\}, q_c, b\})$ is shown (with $\mathcal{A}$ composed by $Q = \{q_c, q_1, q_2\}$ and dashes mark added states and transitions).

  From this follows that $inf(r_w) = inf(r'_w)$ and with $inf(r_w) \cap F \neq \emptyset$ and $F_d \supset F$ (by construction) follows that $inf(r'_w) \cap F_d \neq \emptyset$.

- $\mathcal{L}_\omega(\mathcal{A}) \supseteq \mathcal{L}_\omega(\mathcal{A}_d)$

  Let $w = w_0 w_1 w_2 ... \in \mathcal{L}_\omega(\mathcal{A}_d)$ and r_w an accepting run on $\mathcal{A}_d$ over w. First off, we can state that if r_w does not visit q_+, by construction $\exists$ run r'_w on $\mathcal{A}$ over w that will be identical to r_w, from which follows that $w \in \mathcal{L}_\omega(\mathcal{A})$. Furthermore, by construction, for every accepting run visiting q_+, there also exists an accepting run that does not visit q_+ (this follows from transition rule 2 of δ_d), i.e. we can derive the previous conclusion that there will be an accepting run on $\mathcal{A}$ over w, i.e. $w \in \mathcal{L}_\omega(\mathcal{A})$.

■

The next transformation duplicates the behaviour of a transition by introducing a new state that can only be reached by passing the duplicating transition, meaning that the language accepted by the transformation remains unchanged. Formally:

Definition 12. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton, let $q_{src}, q_{dest} \in Q$ be states of $\mathcal{A}$ and let $a_c \in \Sigma$ be a symbol with a transition going from q_{src} to q_{dest}, i.e. $q_{dest} \in \delta(q_{src}, a_c)$. Let $q_+ \notin Q$ be a newly introduced state. Then the **TP "duplicate transition behaviour** $\mathfrak{T}_t$ is defined as follows:*

$$\mathfrak{T}_t(\mathcal{A}, [q_{src}, q_{dest}, a_c]) = (Q_t, \Sigma, \delta_t, q_0, F_t),$$

where

$$Q_t = Q \cup \{q_+\},$$

$$F_t = \begin{cases} F & \text{if } q_{dest} \notin F \\ F \cup \{q_+\} & \text{if } q_{dest} \in F \end{cases}$$

and

$$\delta_t(q,a) = \begin{cases} \delta(q,a) \cup \{q_+\} & \textit{if } q = q_{src} \textit{ and } a = a_c \\ \{q \mid q \in \bigcup\limits_{q_s \in \delta(q_{src},a_c)} \delta(q_s,a)\} & \textit{if } q = q_+ \\ \delta(q,a) & \textit{else} \end{cases}$$

Before proving preserving properties of this construction, let us again illustrate the transformation with the example from Fig. 5:

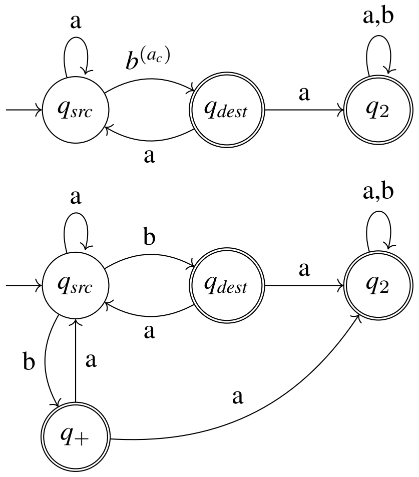


FIGURE 5. With $\Sigma = \{a, b\}$, $\mathcal{A}$ on top and $\mathcal{A}_t = \mathfrak{T}_t(\mathcal{A}, [q_{src}, q_{dest}, a_c])$ on the bottom.

Being very similar to the duplicating SCC transformation, as for every possible accepted word, it is possible to find at least one accepting run that will exist both in $\mathcal{A}$ and $\mathcal{A}_t$ (one not passing through q_+), we will omit the proof of the following language equivalence theorem:

Theorem 4. *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton. For two states $q_{src}, q_{dest} \in Q$ and $a_c \in \Sigma$, let $\mathfrak{T}_t(\mathcal{A}, [q_{src}, q_{dest}, a_c])$*

Then

$$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}_t)$$

IV. BÜCHI AUTOMATA DATASETS

After gaining familiarity with the different transformations used to modify NBW while preserving their universality property, the automaton structures will now need to be encoded as input for GNN and populate a range of datasets. This section describes the encoding process for GNN, the construction of transformation datasets and the generation of randomly generated labelled automata.

A. NBW AS RGCN INPUT DATA

This section will detail the process of encoding Büchi automata as RGCN input data (as described in Section II-B). Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton. The RGCN input graph $G = (V, E, \mathcal{R})$ is then defined as follows:

- set of nodes V is equal to the set of states of the automaton, i.e. $V = Q$.
- set of relations $\mathcal{R}$ will be the set of symbols Σ, i.e. $\mathcal{R} = \Sigma$.
- set of edges E will be constructed as follows. Let $(v_i, r, v_j) \in E$, for $v_i, v_j \in V$ and $r \in \mathcal{R}$ if and only if
$$v_j \in \delta(q_i, r).$$

The initial node feature matrix $\mathbf{X}_0$ (i.e. for layer $l = 0$) will, $\forall v \in V$, consist of the feature vector $\mathbf{h}_v^{(0)}$ defined as follows:

$$\begin{bmatrix} v^{init} \\ v^{acc} \end{bmatrix},$$

where

- $v^{init} = 1 \Leftrightarrow v = q_0$ encodes if v is the initial state,
- $v^{acc} = 1 \Leftrightarrow v \in F$ encodes if v is an accepting state.

B. TRANSFORMATION DATASETS

This section will give details about the process of generating labelled datasets using preserving transformations and the used functions, a process which can be consulted in Algorithm 1.

Algorithm 1 Transformed NBW Dataset Generation
```
1:  function GenDataset($d, n_{\min}, n_{\max}, \mathbb{A}^u, \mathbb{A}^{nu}, \mathbb{T}, W_{\mathbb{T}}$)
2:      dataset $\leftarrow \{\}$
3:      while size(dataset) $< d$ do
4:          $n \leftarrow$ random_integer($n_{min}, n_{max}$)
5:          if size(dataset) $\leq d/2$ then
6:              $\mathcal{A}_{\text{base}} \leftarrow$ random_choice($\mathbb{A}^u$)
7:          else
8:              $\mathcal{A}_{\text{base}} \leftarrow$ random_choice($\mathbb{A}^{nu}$)
9:          end if
10:         automaton $\leftarrow \mathcal{A}_{\text{base}}$
11:         while size(automaton) $< n$ do
12:             $\mathcal{T} \leftarrow$ random_weighted_choice($\mathbb{T}, W_{\mathbb{T}}$)
13:             automaton $\leftarrow$ apply_$\mathcal{T}$(automaton)
14:         end while
15:         automaton $\leftarrow$ prune(automaton)
16:         dataset $\leftarrow$ dataset $\cup$ {automaton}
17:     end while
18:     return dataset
19: end function
```

Given a number of automata $d \in \mathbb{N}$, a minimum and maximum size $n_{min}, n_{max} \in \mathbb{N}$, a set of labelled universal base automata $\mathbb{A}^u$, a set of labelled non-universal base automata $\mathbb{A}^{nu}$ and a set of transformations $\mathbb{T}$ with an accompanying set of weights $W_{\mathbb{T}} \subseteq \mathbb{N}$, the dataset generation procedure generates d-many different automata with a random generating bound between n_{min} and n_{max} by randomly choosing a universal base automaton $\mathcal{A}_{base} \in \mathbb{A}^u$ for the first $\frac{d}{2}$ automata and choosing a non-universal base automaton $\mathcal{A}_{base} \in \mathbb{A}^{nu}$ for the second half, then applying a randomly chosen transformation $\mathcal{T}$ from the transformation set $\mathbb{T}$ using the weights $W_{\mathbb{T}}$ until the bound of desired nodes is surpassed. To make all the states in the automaton relevant regarding its language acceptance, i.e. to remove structural noise for the GNN that doesn't change the behaviour of the automaton structure, the resulting automaton is pruned before being added to the dataset. These various functions from Algorithm 1 are defined as follows:

- The function 'random_integer(n_{min}, n_{max})' returns, for $n_{min}, n_{max} \in \mathbb{N}$, a random integer from $[n_{min}, n_{max}]$ with a uniform distribution.
- The functions 'random_choice($\mathbb{A}^u$)' and 'random_choice ($\mathbb{A}^{nu}$)' return a random automaton from the sets containing automata $\mathbb{A}^u$ resp. $\mathbb{A}^{nu}$ with a uniform distribution.
- The function 'random_weighted_choice($\mathbb{T}, W_{\mathbb{T}}$)' chooses a transformation from the set of transformations $\mathbb{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n\}$ with the probability
$$P(\mathcal{T}_i) = \frac{w_i}{\sum_{j=1}^{n} w_j}, \forall 1 \leq i \leq n,$$
where $W_{\mathbb{T}} = \{w_1, w_2, \ldots, w_n\}$ is the set of non-negative integer weights.
- The function 'apply_$\mathcal{T}$(automaton)' applies the transformation $\mathcal{T}$ to the given automaton. Depending on $\mathcal{T}$, the corresponding transformation parameters are also passed to the function, in accordance with the definitions from Section III.
- The function 'prune(automaton)' prunes the given automaton by removing all nodes which are not reachable from the initial state or which have no path to a self-reaching state[8]. An example of this procedure can be seen in Fig. 6.

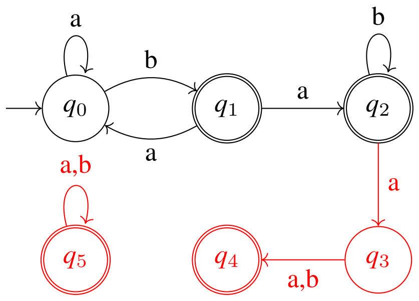


FIGURE 6. **Pruning procedure of the given NBW $\mathcal{A}$. To be removed states and transitions are highlighted in red.**

C. ERDÖS-RENYI DATASETS

Although the universality check for Büchi automata is computationally complex, it is still feasible for small automata. This allows us to generate correctly labelled datasets with randomly generated small automata, providing us with ground truth data that can be used as base automata for our transformation datasets, as testing and validation datasets for comparative analysis and to improve network performance through inclusion in training. In this section, we will first highlight how these small automata are randomly generated and then show the computational limits of the universality check with respect to the automaton size.

1) Random generation of NBW

The random automata generation is based on the Erdős-Rényi graph model [19], where a graph $G(n, p)$ is defined as a graph with n nodes and all possible edges are included with probability p. We extended this approach to NBW, by counting all possible edges of the graph structure once for each symbol in Σ. In addition, a second probability p_{acc} is defined to determine for each state if they belong to F, i.e. are accepting.

Furthermore, we allow for more variety in the generation by letting the input be an upper and lower bound for node sizes, edge probabilities and acceptance probabilities (with n, p and p_{acc} being randomly chosen in between these bounds) and reduce the automaton structure to only include states that are self-reachable and reachable from the initial state by applying the same pruning procedure seen before.

2) Algorithmically checking universality

As it was already discussed in Section II, the complexity bottleneck of the universality check is the complementation construction of NBW, as it yields a node growth of $\mathcal{O}(0.76n^n)$. For our implementation, we used a simplified version of the optimal construction from [20], which is slightly slower in the worst case. The emptiness check of the complement can subsequently be done in linear time to determine universality of the original automaton.

TABLE 1. **Randomly Generated NBW Complementation Statistics.**

Num. nodes:	**6**		**8**		**10**	
	sec	**#**	**sec**	**#**	**sec**	**#**
Highest	385.69	4447	309.44	3491	9924.1	17004
Average	0.692	64	1.32	107	14.99	200
Median	0.035	33	0.098	54	0.102	54

Computation times and complement automaton sizes for 1000 constructions of non-deterministic Büchi automata with 6, 8, and 10 nodes.

Table 1 shows the computation times and complement automaton sizes for 1000 constructions of complement NBW with 6, 8, and 10 nodes. The slowest construction for each size of the starting NBW took a significant amount of time, with the slowest construction for an NBW with 10 nodes taking over 2 and a half hours to complete. The average computation times for each size of the starting NBW were 0.692 seconds, 1.32 seconds, and 14.99 seconds for NBW with 6, 8, and 10 nodes, respectively, which is consistent with the complexity of the algorithm. It is also noteworthy that the random generation of the input NBW influenced the worst case computation times, with the slowest computation time for NBW with 6 nodes being slower than the slowest time for NBW with 8 nodes, due to the random nature of the generated NBW.

[8]A self-reaching state in a NBW is a state that has a path back to itself.

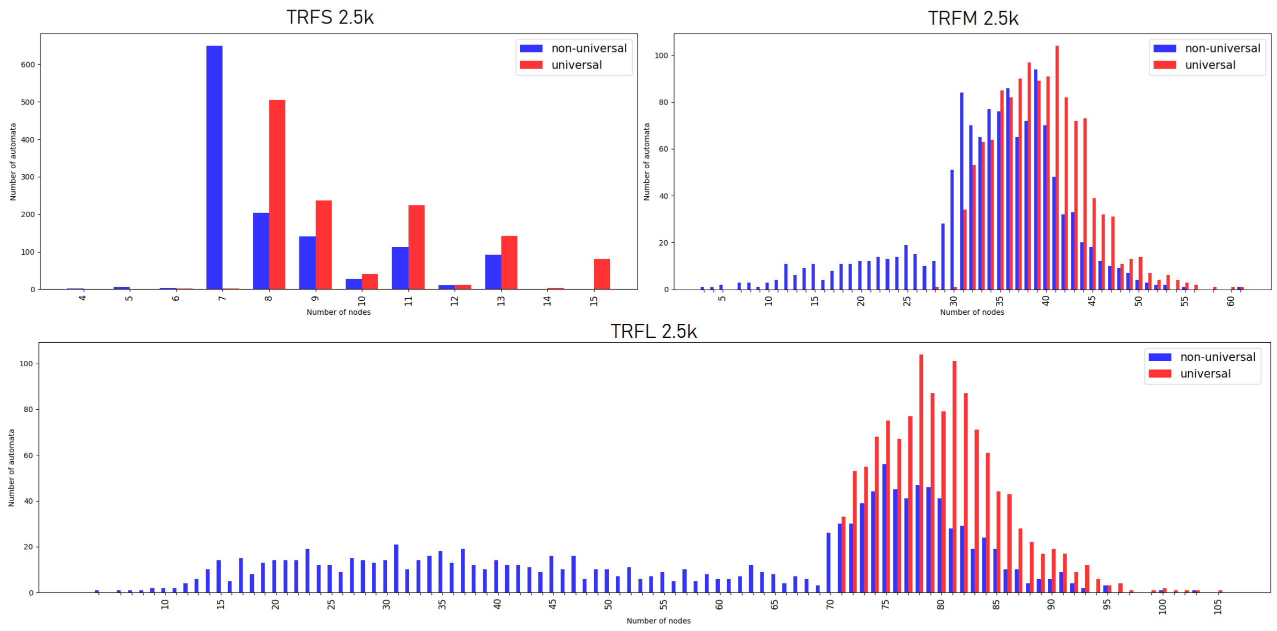


FIGURE 7. Distribution charts of automaton sizes (per class: blue are non-universal, red are universal) for the 3 transformed datasets containing 2500 small, medium or large automata.

V. EXPERIMENTAL RESULTS

We have conducted several experiments to investigate the potential of using GNN for predicting universality of Büchi automata. In the following, Section A describes the datasets of automata used, Section B presents the hyper-parameter optimization for the GNN, and Section C discusses the classification results obtained.

All the experiments regarding the construction and generation of datasets were conducted on a laptop with a Intel(R) Core i7 CPU processor with 16GB RAM. All RGCN models described in this section were trained on up to 8 NVIDIA V100 GPU cores with 32GB RAM each.

A. DATASETS

This section will start by presenting the datasets used in the experiments and establish consistent naming conventions throughout this section.

Let us start by defining the transformed **TRF** datasets, which were constructed using the transformation procedure outlined in Section IV-B. These datasets are primarily differentiated by the number of data elements they contain and by the size of the automata, determined by the parameters n_{min} and n_{max} for the dataset generation (given in parenthesis) and denoted by a **S**, **M**, **L** or **A** at the end of the dataset name abbreviation:

- small (7, 7) dataset name **TRFS**,
- medium (30, 40) dataset name **TRFM**,
- large (70, 80) dataset name **TRFL**,
- all (10, 80) dataset name **TRFA**.

Fig. 7 presents distribution charts depicting the sizes of automata and their class membership per automaton size for the datasets comprising 2500 automata (utilized as test sets throughout the experiments) across the three automaton sizes. It is interesting to note that the automata from the transformations procedure, due to the pruning of the transformed automata and the nature of the transformations, that are smaller are more frequently non-universal.

These datasets use only the preserving transformations[9] from Section III-C to expand both universal and non-universal base automata. This minimizes the bias towards transformation patterns that the transformations preserving (non)-universality would introduce into the structure of the automata. The generation parameters and dataset statistics of the transformed datasets can be examined in Table 2.

TABLE 2. Automaton size statistics for transformed (TRF) training datasets.

Dataset		Statistics		
Name	Size	Average	Min	Max
TRFS	1k	9.1	2	15
	5k	9.1	4	15
	10k	9.1	2	15
TRFM	1k	36.7	5	56
	5k	36.7	4	67
	10k	36.6	4	71
TRFL	1k	69.0	3	101
	5k	68.4	3	115
	10k	69.0	3	108
TRFA	300k	43.0	1	155

The Erdös-Renyi **ER** datasets containing randomly generated and labelled automata are mainly differentiated by the amount of data in the datasets (as Section IV-C showed, these datasets can only contain small automata due to computation

[9]In all experiments in this article, the weights for the random transformation choice will all be equal.

complexity restraints during their generation). Table 3 provides the dataset statistics.

TABLE 3. Automaton size separation statistics for Erdös-Renyi (ER) datasets.

Dataset		Statistics		
Name	Size	Average	Min	Max
	2.5k	5.7	2	9
ER	10k	5.8	1	9
	27k	6.0	2	9

With the aim of encompassing the variability introduced by our different data generation strategies, we combined the NBW from the training sets from each approach (adding up to approximately 75000 data elements). This **master** dataset[10] gives another data configuration exploring the impact the dataset creation strategy may have on the performance of the trained model. Fig. 8 gives an overview of the distribution of the number of nodes (per universality class) over the dataset.

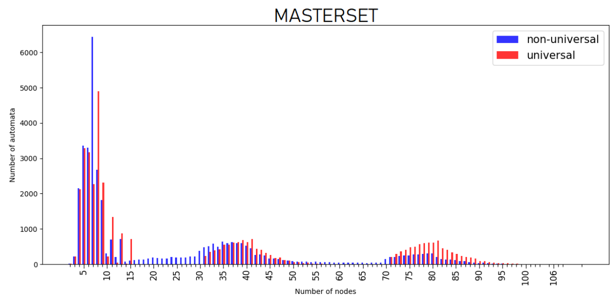


FIGURE 8. Master dataset automaton size distribution (per class: blue are non-universal, red are universal).

B. NEURAL NETWORK ARCHITECTURE AND PARAMETERS

In this section, we describe our choices regarding network architecture and hyper-parameters, which were optimized with respect to the classification accuracy on a validation dataset of 10k **ER** automata.

The choices of the optimizer, Adam [21], the activation function ($ReLU$) and the normalization constant $c_{i,r} = |\mathcal{N}_i^r|$ follow the experimental conventions detailed in the RGCN paper [18]. The selection of the remaining parameters is based on the following results:

- **Number of hidden layers L and their number of node features (i.e. $d^{(l)}$, for $1 \leq l \leq L$):** $L = 4$ and all $d^{(l)} = 128$.

  To determine the number of layers L and the amount of node features per hidden layers, for all combinations of $L \in \{1, 4, 7\}$ and $d^{(l)} \in \{64, 128, 256, 512, 1024\}$, 3 models were trained on all transformed training datasets (sizes 1000, 5000 and 10000) and their averaged classification accuracy (to minimize the effect of the randomization of initial learnable weights) on the validation set was computed to determine the best performance.

  For the models with 4 hidden layers, the classification accuracy over all the different trained models was higher than for both 1 and 7 hidden layers, averaging at 78.89% accuracy (compared to 74.18% for one hidden layer and 75.09% for 7 hidden layers). Thus, after fixing the number of hidden layers to be 4, let us analyse in detail the classification accuracies for the various numbers of hidden node features, presented in Table 4.

TABLE 4. Average classification accuracies (in %, tested on validation set) over 3 models with 4 layers over various TRF training sets and numbers of node features.

Trainingset		Number of hidden nodes				
Name	Size	64	128	256	512	1024
	1k	78.99	80.84	76.29	80.85	77.88
TRFS	**5k**	80.11	79.1	78.71	78.58	78.66
	10k	80.84	81.13	79.26	78.66	78.96
	1k	79.18	80.24	79.31	79.24	79.62
TRFM	**5k**	79.88	79.79	77.97	78.35	77.3
	10k	80.04	81.62	80.78	80.22	79.56
	1k	77.91	77.34	76.2	76	75.79
TRFL	**5k**	77.36	77.55	74.35	77.2	74.66
	10k	78.76	81.72	83.9	84.05	75.09
	Average	79.23	**79.93**	78.53	79.24	77.5

  These results show that over the various training datasets, the highest averaged classification results were achieved by models containing 128 node features for each hidden layer.

- **Learning rate η:** 0.001.

  With the fixed layer parameters, we conducted several experiments regarding the learning rate. The results of these comparative experiments, where 3 models were trained for each of the transformed training sets using the following learning rates: 0.01, 0.005, 0.001, 0.0005. The average classification over the 3 models on the validation set for each configuration is represented in Fig. 9.

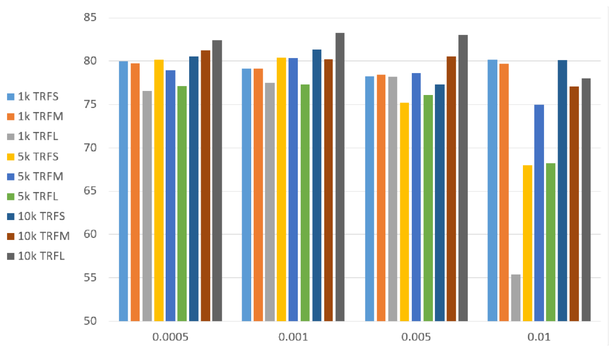


FIGURE 9. Averaged accuracy comparison over 3 models, each for various learning rates.

These results show that the learning rates of 0.01 and 0.005 perform worse on the trained models, but between

[10]**Master set size statistics:** Average: 26.6 Min: 2 Max: 112.

0.001 and 0.0005, the choice of the learning rate does not lead to very significant classification accuracy changes, so it is going to be fixed on 0.001 by having a slightly higher average classification accuracy (79.85% opposed to 79.63%), which incidentally is also the proposed choice of the paper introducing the Adam optimizer.

- **Number of epochs:** Flexible (maximum 100)

  The rolling average classification accuracy over the validation set for the past 5 epochs is calculated after each epoch. After 100 epochs, the model configuration after the epoch with the highest rolling average is output. During the training process, the models start overfitting to the training set. This is illustrated by Fig. 10, where in both cases after the initial rise in classification accuracy, the accuracy keeps to improve on the training data, but slowly converges on the validation set, which is why the model at the point in time of highest average accuracy over the validation set is output.

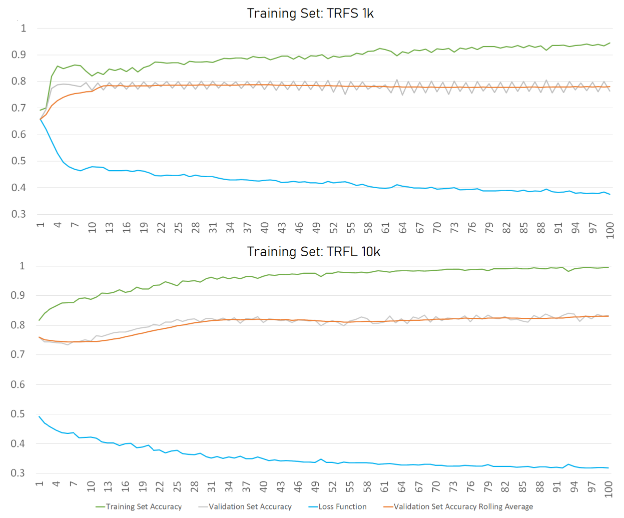


FIGURE 10. Example of training set and validation set classification accuracy and loss function evolution during training over 1k TRFS or 10k TRFL automata.

- **Network Architecture:** Relational GCN

  With the hyperparameters for the network training fixed, Table 5 will show the classification results when using two different layers for training, GCNConv and RGCN layers. For GCNConv, the only change in the automaton encoding stems from encoding the transition labels, as this architecture requires an edge feature vector of length $|\Sigma|$ one-hot encoding the transition symbol (whereas RGCN edges are labelled with a relation) where the i^{th} bit is set to one if and only if the transitions reads the i^{th} symbol in Σ.

  These results show that when trained on small automata, the GCNConv layer using the one-hot encoded edge labels perform similarly to the relational approach, but fall off more visibly when training on larger automata, where the need to infer structural reasoning is larger, as the verification set automata are also small.

TABLE 5. Average classification accuracies (in %, tested on validation set) over 3 models trained on two different architectures.

Training Sets		Layer architecture	
Name	Size	RGCN	GCNConv
TRFS	1k	80.84	80.07
	5k	79.10	78.09
	10k	81.13	80.44
TRFM	1k	80.24	78.49
	5k	79.79	77.23
	10k	81.62	77.79
TRFL	1k	77.34	73.04
	5k	77.55	71.47
	10k	81.72	71.96

C. CLASSIFICATION EXPERIMENTS

With the parameters of the learning process fixed, we can now present the classification results given various combinations of training and test sets. This result analysis allows us to show the impact the choice of training set (and the strategy used behind creating the data) has on the capabilities of a model to recognize patterns behind the (non-)universality of the given input NBW.

To start, let us look at the classification results of models being trained on all the different transformed dataset. The first results are given for test sets containing transformed datasets and can be consulted in Table 6.

TABLE 6. Average classification accuracies (in %, $\pm$ std. dev.) of 3 models trained and tested on various disjunct datasets.

Training sets		TRF Testsets, 2500 Automata		
Name	Size	TRFS	TRFM	TRFL
TRFS	1k	86.20 ± 1.59	82.61 ± 1.42	81.56 ± 1.57
	5k	86.47 ± 0.88	80.31 ± 0.99	77.79 ± 0.96
	10k	93.53 ± 3.16	87.52 ± 3.04	85.73 ± 3.06
TRFM	1k	84.03 ± 1.57	80.97 ± 2.09	80.44 ± 2.06
	5k	84.24 ± 0.34	80.80 ± 0.49	79.20 ± 0.69
	10k	98.09 ± 0.69	96.25 ± 0.69	95.71 ± 0.41
TRFL	1k	83.11 ± 3.43	84.36 ± 1.04	84.21 ± 0.50
	5k	84.16 ± 3.62	84.05 ± 2.44	83.29 ± 2.34
	10k	97.15 ± 0.61	97.16 ± 0.53	97.41 ± 0.46
TRFA	300k	99.22 ± 0.61	98.99 ± 0.35	98.38 ± 0.60

There are a few things to note from these results:

- Solid performance when testing on other transformed datasets. Larger training sets lead to better classification accuracy over most test sets.
- Good generalization for larger transformed automata. The models learning on small NBW are able to generalize the structures needed to derive (non-)universality well for automata containing more nodes.

Table 7 shows the classification results when tasking models trained on various transformed automata datasets to classifying Erdös-Renyi automata.

We can now add the following observations:

- Drop-off for classification accuracy on randomly generated NBW compared to the transformed automata test

TABLE 7. Average classification accuracies (in %, ± std. dev.) of 3 models tested on 2500 disjunct ER automata.

Training sets		ER Testset, 2500 Automata
Name	**Size**	
TRFS	**1k**	81.28 ± 1.50
	5k	82.97 ± 0.73
	10k	83.23 ± 0.69
TRFM	**1k**	81.17 ± 0.30
	5k	82.55 ± 0.15
	10k	81.79 ± 0.84
TRFL	**1k**	80.29 ± 1.58
	5k	79.53 ± 0.63
	10k	85.16 ± 0.09
TRFA	**300k**	84.94 ± 1.53

sets, especially when datasets are larger. The structurally less "predictable" randomly generated automata seem harder to classify, thus opening the conjecture that these models recognize more the transformation patterns and less the patterns connected to (non-)universality.

- Removing the segregation into datasets containing small, medium and large automata and by vastly augmenting the number of data elements that the model has access to to learn the automaton structures (i.e. training on 300k **TRFA** automata), we can see that the model strongly minimizes classification errors on the transformed test sets, but doesn't produce significantly better results on the randomly generated NBW.

Another observation that is in line with the data generation can be made by having a look at the distribution of the errors of the classifications (i.e. the distribution of misclassifications into true and false positives), which can be consulted in Table 8.

TABLE 8. Distribution (in %) of the misclassifications of various training sets when tested on 2500 ER automata.

Training Sets		Misclassification distribution	
Name	**Size**	**False positive %**	**False negative %**
TRFS	10k	18.1 %	81.9 %
TRFM	10k	15.2 %	84.8 %
TRFL	10k	27 %	73 %
ER	27k	40 %	60 %

Analysing these distributions show that the models that are trained on transformed automata are producing a bias towards non-universality when the automata are small, thus increasing the percentage of false negatives when tested on **ER** automata, which are exclusively smaller automata. This bias is a logical consequence of the size distribution as seen earlier in Fig. 7, which showed that small transformed automata are more likely to be non-universal, due to the generation process.

To continue, we will have a look at the performance of a model that is trained on the randomly generated NBW and how it performs when tasked to classify the various test sets. The results are presented in Table 9.

TABLE 9. Average classification accuracy (in %, ± std. dev.) and std. dev. of 3 models trained on 27000 ER automata.

Testsets, 2500 automata	Accuracy
TRFS	85.53 ± 1.18
TRFM	60.09 ± 1.30
TRFL	52.15 ± 0.88
ER	94.76 ± 0.56

Here we see that the models trained on randomly generated and algorithmically classified Erdös-Renyi automata get a strong classification accuracy on the testing set containing different randomly generated automata, with a weaker classification of the small transformed automata and a substantial decline for the medium and large transformed automata, showing a lack of generalization towards larger automata.

The final results will show the classification accuracy of the models trained on the master dataset when tasked to classify the various testing datasets. In Table 10, we can see that the training set containing a variety of differently constructed data elements leads to the best classification accuracies for the transformed automata, and reaching very strong classification results on the 2500 randomly generated **ER** automata. This shows that the most promising way to infer (non-)universality from a general structure of a given NBW is to train on sets containing a large variety of automata with respect to their generation.

TABLE 10. Average classification accuracies (in %, ± std. dev.) and std. dev. of 3 models trained on master dataset.

Testsets, 2500 automata	Accuracy
TRFS	99.79 ± 0.12
TRFM	99.56 ± 0.45
TRFL	99.46 ± 0.32
ER	93.58 ± 0.64

VI. CONCLUSION

This section will recap the results that were acquired previously, give a concluding overview of the contents of this article and present various ideas for future research work.

A. RESULTS AND CONCLUSION

Based on the achieved results, we can assert that our contribution is twofold and can be summarized as follows:

- The dataset generation approach with the transformations allows to quickly generate datasets for GNN training that lead the models to a basic understanding regarding the universality problem, with solid generalization results when applied to larger automata generated using the same methodology. The drawbacks in classification accuracy of randomly generated automata can be mitigated by creating larger training datasets incorporating a mix of transformed automata and randomly generated (algorithmically classified) ones, significantly improving classification accuracy.
- We showed that the RGCN architecture for classifying graphs can be applied to Büchi automata structures, a

special case of a graph. The experimental results show that the models, when training on variously constructed datasets containing NBW with the goal of learning about (non-)universality, behave the way that is expected when changing different parameters (both on the dataset construction and on the models themselves) and reach surprisingly strong classification accuracies on randomly generated Büchi automata when trained on a training set combining randomly generated automata with transformed automata.

B. FUTURE WORK

The results that were shown in this article are mainly a show of proof of concept by presenting GNN models that were trained on graph structures representing Büchi automata for a classification task that is shown to be computationally expensive. With this work, to our knowledge, being the first time in literature to classify a given Büchi automaton using GNNs regarding their universality, there is a lot of potential for follow-up work to improve both the data generation and the training process. In this section, we will propose several ideas to serve as starting points for future research endeavours.
Let us start with a few ideas in regards to data generation:

- Improve the algorithmic classification of randomly generated automata to create a bigger dataset containing more possible NBW as a baseline testing dataset.
- Analyse the effect on classification of manipulating the weights on the transformations, the impact of including the transformations that exclusively preserve (non-)universality or the removal of the pruning procedure of the automata.
- As more classification errors occur on universal automata in the presented results, experiment on training models on unbalanced datasets containing more universal automata in order to reach structural conclusions regarding universality.

The process of training GNN also presents different opportunities for further experimentation:

- Use easily checkable features of nodes (e.g. is contained in a SCC, has a self-loop, ...) to add more structural information to the base node features to facilitate the model's learning.
- Analyse in more detail the node features after the readout to learn from the information that the model is gathering during the message passing.
- Due to the rapid development of GNN architecture research, experiment with different models and compare results, e.g. transformers using self-attention.

The procedure of classifying NBW presented in this article could also be subject to comparative research with the results from applications from formal verification requiring a universality check. Generally speaking, we encourage subsequent experimentation on this topic, both in regards of the dataset generation and in the training and testing of the models, to enhance the performance of the datasets or the scope of the dataset generation.

REFERENCES

[1] J. R. Büchi, *On a Decision Method in Restricted Second Order Arithmetic*. New York, NY: Springer New York, 1990, pp. 425–435. [Online]. Available: https://doi.org/10.1007/978-1-4613-8928-6_23
[2] M. Y. Vardi, "The büchi complementation saga," in *STACS 2007*, W. Thomas and P. Weil, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 12–22.
[3] M. Vardi, "Automata-theoretic model checking revisited," 01 2007, pp. 137–150.
[4] L. Doyen and J.-F. Raskin, "Improved algorithms for the automata-based approach to model-checking," 03 2007, pp. 451–465.
[5] C. Stammet, P. Dotti, U. Ultes-Nitsche, and A. Fischer, "Analyzing büchi automata with graph neural networks," 2022.
[6] E. Grädel, W. Thomas, and T. Wilke, Eds., *Automata Logics, and Infinite Games: A Guide to Current Research*. Berlin, Heidelberg: Springer-Verlag, 2002.
[7] S. Safra, "On the complexity of omega -automata," in *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, 1988, pp. 319–327.
[8] O. Kupferman and M. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, pp. 291–314, 09 1999.
[9] A. R. Meyer and L. J. Stockmeyer, "The equivalence problem for regular expressions with squaring requires exponential space," in *Scandinavian Workshop on Algorithm Theory*, 1972. [Online]. Available: https://api.semanticscholar.org/CorpusID:206585190
[10] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640
[11] T. Bohnstingl, A. Garg, S. Woźniak, G. Saon, E. Eleftheriou, and A. Pantazi, "Towards efficient end-to-end speech recognition with biologically-inspired neural networks," 2021.
[12] F. Stahlberg, "Neural machine translation: A review," *CoRR*, vol. abs/1912.02047, 2019. [Online]. Available: http://arxiv.org/abs/1912.02047
[13] L. J. Lancashire, C. Lemetre, and G. R. Ball, "An introduction to artificial neural networks in bioinformatics—application to complex microarray and mass spectrometry datasets in cancer studies," *Briefings in Bioinformatics*, vol. 10, no. 3, pp. 315–329, 03 2009. [Online]. Available: https://doi.org/10.1093/bib/bbp012
[14] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.
[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762
[17] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *CoRR*, vol. abs/1901.00596, 2019. [Online]. Available: http://arxiv.org/abs/1901.00596
[18] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*. Springer, 2018, pp. 593–607.
[19] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hungary. Acad. Sci.*, vol. 5, pp. 17–61, 1960.
[20] J. D. Allred and U. Ultes-Nitsche, "A simple and optimal complementation algorithm for büchi automata," in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 46–55. [Online]. Available: https://doi.org/10.1145/3209108.3209138
[21] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

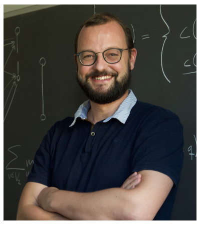
CHRISTOPHE STAMMET received his M.S. degree in Computer Science with a specialization into Logic from the University of Fribourg, Switzerland, in 2017. He is currently finishing his doctoral research in the research group 'Foundations of Dependable Systems', led by Prof. Dr. Ulrich Ultes-Nitsche.

After his doctoral contract ends, he will work at the University of Luxembourg as an Education Programme Specialist at the Computer Science Division of the Scienteens Lab.

The research of Christophe Stammet seeks to combine Machine Learning frameworks on graphs as inputs with problems rooted in theoretical Computer Science, most notably in Automata theory on infinitely long inputs. An analysis of proof-of-concept systems of his research was presented at the LearnAut workshop held at the ICALP 2022.

ULRICH ULTES-NITSCHE is a professor of computer science at the University of Fribourg, Switzerland. He received his PhD in 1992 from the University of Frankfurt, Germany. Before joining the University of Fribourg in 2003, he had appointments at the Universities of Southampton (UK), Zurich (Switzerland), Frankfurt (Germany) as well as with the German National Research Centre for Mathematics and Computer Science (now Fraunhofer).

His research interests include formal methods, automata theory, verification, logic, abstraction, dependability.



ANDREAS FISCHER received the M.S. and PhD degrees in Computer Science from the University of Bern, Switzerland, in 2008 and 2012, respectively. Afterwards, he conducted postdoctoral research in Montreal, Canada, at the Centre for Pattern Recognition and Machine Intelligence (CENPARMI) and at Polytechnique Montreal. Currently, he is Full Professor at the University of Applied Sciences and Arts Western Switzerland (HES-SO, HEIA Fribourg) and Lecturer at the University of Fribourg, Switzerland.

Andreas Fischer's research interests include pattern recognition, deep learning, graph-based methods, document analysis, and handwriting recognition. He has published over 100 peer-reviewed articles in international journals and conference proceedings on these topics.

Andreas Fischer is member of the IEEE, member of the governing board of the International Association of Pattern Recognition (IAPR), and Chair of the IAPR technical committee on reading systems (TC11).