An Automata-Theoretic Approach to Synthesizing Binarized Neural Networks

Ye Tao[1][0009−0007−1478−9144], Wanwei Liu[1⋆][0000−0002−2315−1704], Fu Song[2,3,4][0000−0002−0581−2679], Zhen Liang[5][0000−0002−1171−7061], Ji Wang[5][0000−0003−0637−8744], and Hongxu Zhu[1]

[1] College of Computer Science and Technology, National University of Defense Technology
`{taoye0117,wwliu,zhuhongxu}@nudt.edu.cn`
[2] School of Information Science and Technology, ShanghaiTech University
`songfu@shanghaitech.edu.cn`
[3] Institute of Software, Chinese Academy of Sciences & University of Chinese Academy of Sciences
[4] Automotive Software Innovation Center
[5] Institute for Quantum Information & State Key Laboratory for High Performance Computing, National University of Defense Technology
`{liangzhen,wj}@nudt.edu.cn`

Abstract. Deep neural networks, (DNNs, a.k.a. NNs), have been widely used in various tasks and have been proven to be successful. However, the accompanied expensive computing and storage costs make the deployments in resource-constrained devices a significant concern. To solve this issue, quantization has emerged as an effective way to reduce the costs of DNNs with little accuracy degradation by quantizing floating-point numbers to low-width fixed-point representations. Quantized neural networks (QNNs) have been developed, with binarized neural networks (BNNs) restricted to binary values as a special case. Another concern about neural networks is their vulnerability and lack of interpretability. Despite the active research on trustworthy of DNNs, few approaches have been proposed to QNNs. To this end, this paper presents an automata-theoretic approach to synthesizing BNNs that meet designated properties. More specifically, we define a temporal logic, called BLTL, as the specification language. We show that each BLTL formula can be transformed into an automaton on finite words. To deal with the state-explosion problem, we provide a tableau-based approach in real implementation. For the synthesis procedure, we utilize SMT solvers to detect the existence of a model (i.e., a BNN) in the construction process. Notably, synthesis provides a way to determine the hyper-parameters of the network before training. Moreover, we experimentally evaluate our approach and demonstrate its effectiveness in improving the individual fairness and local robustness of BNNs while maintaining accuracy to a great extent.

⋆ Corresponding Author

1 Introduction

Deep Neural Networks (DNNs) are increasingly used in a variety of applications, from image recognition to autonomous driving, due to their high accuracy in classification and prediction tasks [27,29]. However, two critical challenges emerge, high-cost and a lack of trustworthiness, that impede their further development.

On the one hand, a modern DNN typically contains a large number of parameters which are typically stored as 32-bit floating-point numbers (e.g., GPT-4 contains about 100 trillion parameters [14]), thus an inference often demands more than a billion floating-point operations. As a result, deploying a modern DNN requires huge computing and storage resources, thus it is challenging for resource-constrained embedding devices. To tackle this issue, quantization has been introduced, which compresses a network by converting floating-point numbers to low-width fixed-point representations, so that it can significantly reduce both memory and computing costs using fixed-point arithmetic with a relatively small side-effect on the network's accuracy [23].

On the other hand, neural networks are known to be vulnerable to input perturbations, namely, slight input disturbance may dramatically change their output [12,3,28,4,35,5,6,7]. In addition, NNs are often treated as black box [17], and we are truly dearth of understanding of the decision-making process inside the "box". As a result, a natural concern is whether NNs can be trustworthy, especially in some safety-critical scenarios, where erroneous behaviors might lead to serious consequences. One promising way to tackle this problem is formal verification, which defines properties that we expect the network to satisfy and rigorously checks whether the network meets our expectations. Numerous verification approaches have been proposed recently aiming at this purpose [17]. Nevertheless, these approaches in general ignore rounding errors in quantized computations, making them unable to apply for quantized neural networks (QNNs). It has been demonstrated that specifications that hold for a floating-point numbered DNN may not necessarily hold after quantizing the inputs and/or parameters of the DNN [3,13]. For instance, a DNN that is robust to given input perturbations might become non-robust after quantization. Compared to DNN verification [17,18,20,21,19,36,15], verifying QNN is truly a more challenging and less explored problem. Evidences show that the verification problem for QNNs is harder than DNNs [16], and only few works are specialized for verifying QNNs [1,8,13,16,24,26,32,33,34,31].

In this paper, we concentrate on BNNs (i.e., binarized neural networks), a special type of QNN. Although formal verification has been the primary explored approach to verifying (quantized) neural networks, we pursue another promising line, synthesizing the expected binarized neural networks directly. In other words, we aim to construct a neural network that satisfies the expected properties we specify, rather than verifying an existing network's compliance with those properties. To achieve this, we first propose, BLTL, an extension of LTL$_f$ (namely, LTL defined on finite words), as the specification language. This logic can conveniently describe data-related properties of BNNs. We then provide an approach to converting a BLTL formula to an equivalent automaton. The synthesis task is then boiled down to find a path from an initial state to an accepting state in the automaton.

Unfortunately, such a method suffers from the state-exploration problem. To mitigate this issue, we observe that it is not necessary to synthesize the entire BNN since the desired properties are only related to some specific hyper-parameters of the network. To this end, we propose a tableau-based approach: To judge whether a path is successfully detected, we check the satisfiability of the associated BLTL formulas, and convert the problem into an IDL-solving problem, which can be efficiently solved. Besides, we prove the existence of a tracing-back threshold, which allows us to do backtracking earlier to avoid doing trace searching that is unlikely to lead to a solution. The solution given by the solver provides the hyper-parameters of the BNN, including the length of the network and crucial input-output relations of blocks. Afterwards, one can perform a block-wise training to obtain a desired BNN.

We implement a prototype synthesizing tool and evaluate our approach on local robustness and individual fairness. The experiments demonstrate that our approach can effectively improve the network's reliability compared to the baseline, especially for individual fairness.

The main contributions of this work are summarized as follows:

- We present a new temporal logic, called BLTL, for describing properties of BNNs, and provide an approach to transforming BLTL formulas into equivalent finite-state automata.
- We propose an automata-theoretic synthesis approach that determines the hyper-parameters of a BNN model before training.
- We implement a prototype synthesis tool and evaluate the effectiveness on two concerning properties, demonstrating the feasibility of our method.

Related Work. For BNNs, several verification approaches have been proposed. Earlier work reduces the BNN verification problem to hardware verification (i.e., verifying combinatorial circuits), for which SAT solvers are harnessed [8]. Following this line, [24] proposes a direct encoding from the BNN verification problem into the SAT problem. [25] studies the effect of BNN architectures on the performance of SAT solvers and uses this information to train SAT-friendly BNNs. [1] provides a framework for approximately quantitative verification of BNNs with PAC-style guarantees via approximate SAT model counting. Another line of BNN verification encodes a BNN and its input region into a binary decision diagram (BDD), and then one can verify some properties of the network by analyzing BDD. [26] proposes an Angluin-style learning algorithm to compile a BNN on a given input region into a BDD, and utilize a SAT solver as an equivalence oracle to query. [32] has developed a more efficient BDD-based quantitative verification framework by exploiting the internal structure of BNNs. Few work has been dedicated to QNN verification so far. [13] shows that the properties guaranteed by the DNN are not preserved after quantization. To resolve this issue, they introduce an approach to verifying QNNs by using SMT solvers in bit-vector theory. Later, [16] proves that verifying QNN with bit-vector specifications is **PSPACE**-Hard. More recently, [34,31] reduce the verification problem into integer linear constraint solving which are significantly more efficient than the SMT-based one.

Outline. The rest of the paper is organized as follows: In Section 2, we introduce preliminaries. We present the specification language BLTL in Section 3. In Section 4, we show how to translate a BLTL formula into an equivalent automaton, which is the basic of tableau-based approach for synthesis, and technical details are given in Section 5. The proposed approach is implemented and evaluated in Section 6. We conclude the paper in Section 7.

2 Preliminaries

We denote by $\mathbb{R}$, $\mathbb{N}$, and $\mathbb{B}$ the set of real numbers, natural numbers, and Boolean domain $\{0,1\}$, respectively. We use $\mathbb{R}^n$ and $\mathbb{B}^n$ to denote the set of real number vectors and binary vectors with n elements, respectively. For $n \in \mathbb{N}$, let $[n]$ be the set $\{0,1,2,\ldots,n-1\}$. We will interchangeably use the terminologies 0-1 vector and binary vector in this paper. For a binary vector $\boldsymbol{b}$, we use $\mathsf{dec}(\boldsymbol{b})$ to denote its corresponding decimal number, and conversely let $\mathsf{bin}(d)$ be the corresponding binary vector which encodes the number d. For example, let $\boldsymbol{b} = (0,1,1)^{\mathrm{T}}$, then we have $\mathsf{dec}(\boldsymbol{b}) = 3$. Note that $\mathsf{bin}(\mathsf{dec}(\boldsymbol{b})) = \boldsymbol{b}$ and $\mathsf{dec}(\mathsf{bin}(d)) = d$. For two binary vectors $\boldsymbol{a} = (a_0,\ldots,a_{n-1})^{\mathrm{T}}$ and $\boldsymbol{b} = (b_0,\ldots,b_{n-1})^{\mathrm{T}}$ with the same length, we denote by $\boldsymbol{a} \sim \boldsymbol{b}$ if $a_i \sim b_i$ for all $i \in [n]$, otherwise $\boldsymbol{a} \not\sim \boldsymbol{b}$, where $\sim \in \{>,\geq,<,\leq,=\}$. Note that $\boldsymbol{a} \neq \boldsymbol{b}$ if $a_i \neq b_i$ for some $i \in [n]$.

A (vectorized) Boolean function takes a 0-1 vector as input and returns another 0-1 vector. Hence, it is essentially a mapping from integers to integers when each 0-1 vector $\boldsymbol{b}$ is viewed as an integer $\mathsf{dec}(\boldsymbol{b})$. We denote by $\boldsymbol{I}_n$ the identity function such that $\boldsymbol{I}_n(\boldsymbol{b}) = \boldsymbol{b}$, for any $\boldsymbol{b} \in \mathbb{B}^n$, where the subscript n may be dropped when it is clear from the context. We use *composition* operation $\circ$ to represent the function composition among Boolean functions.

A *binarized neural network* (BNN) is a feed-forward neural network, composed of several internal blocks and one output block [26,32]. Each internal block is comprised of 3 layers and can be viewed as a mapping $f : \{-1,1\}^n \to \{-1,1\}^m$. Slightly different from internal blocks, the output block outputs the classification label to which the highest activation corresponds, thus, can be seen as a mapping $\mathsf{out} : \{-1,1\}^n \to \mathbb{R}^p$, where p is the number of classification labels of the network.

Since the binary values -1 and $+1$ can be represented as their Boolean counterparts 0 and 1 respectively, each internal block can be viewed as a Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ [32]. Therefore, ignoring the slight difference in the output block, an n-block BNN $\mathcal{N}$ can be encoded via a series of Boolean functions $f_i : \mathbb{B}^{\ell_i} \to \mathbb{B}^{\ell_{i+1}}$ $(i = 0,1,\ldots,n-1)$, and $\mathcal{N}$ works as the combination of these Boolean functions, namely, it corresponds to the function,

$$f_{\mathcal{N}} = f_{n-1} \circ f_{n-2} \circ \cdots \circ f_1 \circ f_0.$$

Integer difference logic (IDL) is a fragment of linear integer arithmetic, in which atomic formulas must be of the form $x - y \sim c$ where x and y are integer variables, and c is an integer constant, $\sim \in \{\leq, \geq, <, >, =, \neq\}$. All these atomic formulas can be transformed into constraints of the form $x - y \leq c$ [2]. For example, $x - y = c$ can be transformed into $x - y \leq c \wedge x - y \geq c$.

The task of an IDL-problem is to check the satisfiability of an IDL formula in conjunctive normal form (CNF)

$$(x_1 - y_1 \leq c_1) \wedge \cdots \wedge (x_n - y_n \leq c_n),$$

which can be in general converted into the cycle detection problem in a weighted, directed graph with $O(n)$ nodes and $O(n)$ edges, and solved by e.g., Bellman-Ford or Dijkstra's algorithm, in $O(n^2)$ time [22]. IDL can be generalized to Boolean combinations of atomic formulas of the form $x - y \sim c$.

3 The Temporal Logic BLTL

3.1 Syntax and Semantics of BLTL

Let us fix a signature $\boldsymbol{\Sigma}$, consisting of a set of desired Boolean functions and 0-1 vectors. Particularly, let $\boldsymbol{\Sigma}_{\mathrm{V}}$ be the subset of $\boldsymbol{\Sigma}$ containing only 0-1 vectors.

Terms of BLTL are described via BNF as follows:

$$\boldsymbol{t} ::= \boldsymbol{b} \mid f\,(\boldsymbol{t}) \mid \rhd^k \boldsymbol{t}$$

where $\boldsymbol{b} \in \boldsymbol{\Sigma}_{\mathbf{V}}$ is a 0-1 vector, called *vector constant*, $f \in \boldsymbol{\Sigma} \setminus \boldsymbol{\Sigma}_{\mathbf{V}}$ is a Boolean function, and $k \in \mathbb{N}$ is a constant, and $\rhd^k$ in $\rhd^k \boldsymbol{t}$ denotes k placeholders for k consecutive blocks of a BNN (i.e., k Boolean functions) to be applied onto the term $\boldsymbol{t}$. We remark that $\rhd^0 \boldsymbol{t} = \boldsymbol{t}$.

BLTL formulas are given via the following grammar:

$$\psi ::= \top \mid \boldsymbol{t} \sim \boldsymbol{t} \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi$$

where $\sim \in \{\leq, \geq, <, >, =\}$, X is the *Next* operator and U is the *Until* operator.

We define the following derived Boolean operators, quantifiers with finite domain, and temporal operators:

$$\begin{array}{rcc} \psi_1 \wedge \psi_2 \overset{\text{def}}{=} \neg(\neg\psi_1 \vee \neg\psi_2) & \mathsf{F}\psi \overset{\text{def}}{=} \top \mathsf{U} \psi & \mathsf{G}\psi \overset{\text{def}}{=} \neg \mathsf{F} \neg \psi \\ \psi_1 \rightarrow \psi_2 \overset{\text{def}}{=} (\neg\psi_1) \vee \psi_2 & \psi_1 \mathsf{R} \psi_2 \overset{\text{def}}{=} \neg(\neg\psi_1 \mathsf{U} \neg\psi_2) & \overline{\mathsf{X}}\psi \overset{\text{def}}{=} \neg \mathsf{X} \neg \psi \\ \forall \boldsymbol{x} \in \mathbb{B}^k . \psi \overset{\text{def}}{=} \bigwedge_{\boldsymbol{b} \in \mathbb{B}^k \cap \boldsymbol{\Sigma}_{\mathbf{V}}} \psi[\boldsymbol{x}/\boldsymbol{b}] & \exists \boldsymbol{x} \in \mathbb{B}^k . \psi \overset{\text{def}}{=} \neg \forall \boldsymbol{x} \in \mathbb{B}^k . \neg\psi & \end{array}$$

where $\psi[\boldsymbol{x}/\boldsymbol{b}]$ denotes the BLTL formula obtained from ψ by replacing each occurrence of $\boldsymbol{x}$ with $\boldsymbol{b}$.

The semantics of BLTL formulas is defined w.r.t. a BNN $\mathcal{N}$ given by the composition of Boolean functions $f_{\mathcal{N}} = f_{n-1} \circ f_{n-2} \circ \cdots \circ f_1 \circ f_0$, and a position $i \in \mathbb{N}$. We first define the semantics of terms, which is given by the function $[\![\bullet]\!]_{\mathcal{N},i}$, inductively:

- $[\![\boldsymbol{b}]\!]_{\mathcal{N},i} = \boldsymbol{b}$ for each vector constant $\boldsymbol{b}$;

- $[\![f\,(\boldsymbol{t})]\!]_{\mathcal{N},i} = f\,([\![\boldsymbol{t}]\!]_{\mathcal{N},i})$;
- $[\![\rhd^k \boldsymbol{t}]\!]_{\mathcal{N},i} = \begin{cases} (f_{i+\mathsf{slen}(\boldsymbol{t})+k-1} \circ \cdots \circ f_{i+\mathsf{slen}(\boldsymbol{t})})([\![\boldsymbol{t}]\!]_{\mathcal{N},i}), & \text{if } k \geq 1; \\ [\![\boldsymbol{t}]\!]_{\mathcal{N},i}, & \text{if } k = 0; \end{cases}$

  where f_i is the identity Boolean function $\boldsymbol{I}$ if $i \geq n$, $\mathsf{slen}(\boldsymbol{b}) = 0$, $\mathsf{slen}(f(\boldsymbol{t})) = \mathsf{slen}(\boldsymbol{t}) + 1$ and $\mathsf{slen}(\rhd^k \boldsymbol{t}) = \mathsf{slen}(\boldsymbol{t}) + k$.

Note that we assume the widths of Boolean functions and their argument vectors are compatible.

Proposition 1. *We have:* $[\![\rhd^k \rhd^{k'} \boldsymbol{t}]\!]_{\mathcal{N},i} = [\![\rhd^{k+k'} \boldsymbol{t}]\!]_{\mathcal{N},i}$.

Subsequently, the semantics of BLTL formulas is characterized via the *satisfaction* relation $\models$, inductively:

- $\mathcal{N}, i \models \top$ always holds;
- $\mathcal{N}, i \models \boldsymbol{t}_1 \sim \boldsymbol{t}_2$ iff $[\![\boldsymbol{t}_1]\!]_{\mathcal{N},i} \sim [\![\boldsymbol{t}_2]\!]_{\mathcal{N},i}$;
- $\mathcal{N}, i \models \neg\varphi$ iff $\mathcal{N}, i \not\models \varphi$;
- $\mathcal{N}, i \models \varphi_1 \vee \varphi_2$ iff $\mathcal{N}, i \models \varphi_1$ or $\mathcal{N}, i \models \varphi_1$;
- $\mathcal{N}, i \models \mathsf{X}\psi$ iff $i < n - 1$ and $\mathcal{N}, i + 1 \models \psi$;
- $\mathcal{N}, i \models \psi_1 \mathsf{U} \psi_2$ iff there is j such that $i \leq j < n$, $\mathcal{N}, j \models \psi_2$ and $\mathcal{N}, k \models \psi_1$ for each $i \leq k < j$;

We may write $\mathcal{N} \models \psi$ in the case of $i = 0$. In the sequel, we denote by $\mathscr{L}(\psi)$ the set of BNNs $\{\mathcal{N} \mid \mathcal{N} \models \varphi\}$ for each formula φ, and denote by $\psi_1 \equiv \psi_2$ if $\mathcal{N}, i \models \psi_1 \Leftrightarrow \mathcal{N}, i \models \psi_2$ for every BNN $\mathcal{N}$ and i.

Proposition 2. *The following statements hold:*

1. $\mathsf{G}\psi \equiv \bot\mathsf{R}\psi$;
2. $\mathsf{F}\psi \equiv \psi \vee \mathsf{X}\mathsf{F}\psi$;
3. $\mathsf{G}\psi \equiv \psi \wedge \overline{\mathsf{X}}\mathsf{G}\psi$;
4. $\psi_1\mathsf{U}\psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \mathsf{X}(\psi_1\mathsf{U}\psi_2))$;
5. $\psi_1\mathsf{R}\psi_2 \equiv \psi_2 \wedge (\psi_1 \vee \overline{\mathsf{X}}(\psi_1\mathsf{R}\psi_2))$.

For a BLTL formula φ and a BNN $\mathcal{N}$, the *model checking* problem w.r.t. φ and $\mathcal{N}$ is to decide whether $\mathcal{N} \models \varphi$ holds.

With the above derived operators, together with the patterns $\neg\neg\psi \equiv \psi$ and $\neg(\boldsymbol{t}_1 \sim \boldsymbol{t}_2) \equiv \boldsymbol{t}_1 \not\sim \boldsymbol{t}_2$, BLTL formulas can be transformed into *negation normal form* (NNF) by pushing the negations ($\neg$) inward, till no the negations are involved.

Given two sets of formulas Γ and Γ' in NNF, we say that Γ' is a *proper closure* of Γ, if the following conditions hold:

- $\Gamma \subseteq \Gamma'$.
- $\psi_1 \wedge \psi_2 \in \Gamma'$ implies that both $\psi_1 \in \Gamma'$ and $\psi_2 \in \Gamma'$.
- $\psi_1 \vee \psi_2 \in \Gamma'$ implies that either $\psi_1 \in \Gamma'$ or $\psi_2 \in \Gamma'$.
- $\psi_1\mathsf{U}\psi_2 \in \Gamma'$ implies $\psi_2 \vee (\psi_1 \wedge \mathsf{X}(\psi_1\mathsf{U}\psi_2)) \in \Gamma'$.
- $\psi_1\mathsf{R}\psi_2 \in \Gamma'$ implies $\psi_2 \wedge (\psi_1 \vee \overline{\mathsf{X}}(\psi_1\mathsf{R}\psi_2)) \in \Gamma'$.

We denote by $\mathsf{Cl}(\Gamma)$ the set consisting of all proper closures of Γ (note that $\mathsf{Cl}(\Gamma)$ is a family of formula sets.) We also denote by $\mathsf{Sub}(\psi)$ the set of the subformulas of ψ except that

– if $\psi_1 \mathsf{U} \psi_2 \in \mathsf{Sub}(\psi)$, then $\psi_2 \vee (\psi_1 \wedge \underline{\mathsf{X}}(\psi_1 \mathsf{U} \psi_2)) \in \mathsf{Sub}(\psi)$;
– if $\psi_1 \mathsf{R} \psi_2 \in \mathsf{Sub}(\psi)$, then $\psi_2 \wedge (\psi_1 \vee \overline{\mathsf{X}}(\psi_1 \mathsf{R} \psi_2)) \in \mathsf{Sub}(\psi)$.

3.2 Illustrating Properties Expressed by BLTL

In this section, we demonstrate the expressiveness of BLTL. Since BLTL has the ability to express Boolean logic and arithmetic operations, we can see that many concerning properties can be specified using BLTL.

We can partition a vector into segments of varying widths, and then define a Boolean function, denoted by e_i, to extract the i-th segment with width of n, namely, $e_i : \mathbb{B}^m \to \mathbb{B}^n$, where m is the width of vector $\boldsymbol{b}$. We use $\boldsymbol{b}[i]$ to refer to $e_i(\boldsymbol{b})$ in the case that $e_i(\boldsymbol{b}) \in \mathbb{B}$.

Local Robustness. Given a BNN $\mathcal{N}$ and a n-width input $\boldsymbol{u}$, $\mathcal{N}$ is robust w.r.t. $\boldsymbol{u}$, if all inputs in the region $B(\boldsymbol{u}, \epsilon)$, are classified into the same class as $\boldsymbol{u}$ [1]. Here, we consider $B(\boldsymbol{u}, \epsilon)$ as the set of vectors that differ from $\boldsymbol{u}$ in at most ϵ positions, where ϵ is the maximum number of positions at which the values differ from those of $\boldsymbol{u}$. The local robustness can be described as follows:

$$\forall \boldsymbol{x} \in \mathbb{B}^n. \sum_{i=1}^{|\boldsymbol{u}|} (\boldsymbol{x}[i] \oplus \boldsymbol{u}[i]) \leq \epsilon \to \mathcal{N}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{u})$$

Individual Fairness. In the context of a BNN $\mathcal{N}$ with an input of t attributes and n-width, where the s-th attribute is considered sensitive, $\mathcal{N}$ is fair w.r.t the s-th attribute, when no two input vectors in its domain differ only in the value of the s-th attribute and yield different outputs [30,37]. The individual fairness can be formulated as:

$$\forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{B}^n. (\neg (e_s(\boldsymbol{a}) = e_s(\boldsymbol{b})) \wedge \forall i \in [t] - \{s\}. e_i(\boldsymbol{a}) = e_i(\boldsymbol{b})) \to \mathcal{N}(\boldsymbol{a}) = \mathcal{N}(\boldsymbol{b})$$

where e_i denotes the extraction of the i-th attribute, $\mathbb{B}^n$ is the domain of $\mathcal{N}$, and $\boldsymbol{a}$, $\boldsymbol{b}$ are input vectors.

In practice, it is possible to select inputs in the $\mathbb{B}^n$, and modify the sensitive attribute to obtain the proper pairs, which only differ in the sensitive attribute. For any such pair $(\boldsymbol{b}, \boldsymbol{b}')$,we formulate the specification as $\mathcal{N}(\boldsymbol{b}) = \mathcal{N}(\boldsymbol{b}')$.

Specification for Internal Blocks. BLTL can specify block-level properties. For instance, the formula

$$\forall \boldsymbol{x} \in \mathbb{B}^4. \mathsf{F} (\boldsymbol{x} \geq \boldsymbol{a} \to \rhd \boldsymbol{x} = \boldsymbol{a})$$

states that there exists a block in the network that behaves as follows: for any 4-bit input whose value is greater than or equal to $\boldsymbol{a}$, the corresponding output is equal to $\boldsymbol{a}$.

4 From BLTL to Automata

In this section, we present both an explicit and an implicit construction that translate a BLTL formula into an equivalent finite-state automaton. We first show how to eliminate the placeholders $\rhd^k$ in terms $\rhd^k \boldsymbol{t}$ and atomic formulas $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$.

4.1 Eliminating Placeholders

To eliminate the placeholders $\rhd^k$ in terms $\rhd^k \boldsymbol{t}$, we define the *apply operator* $[\,] : \boldsymbol{T} \times \boldsymbol{\Sigma} \setminus \boldsymbol{\Sigma}_{\mathbf{V}} \to \boldsymbol{T}$, where $\boldsymbol{T}$ denotes the set of terms. $[\boldsymbol{t}, f]$, written as $\boldsymbol{t}[f]$, is called the *application* of the term $\boldsymbol{t}$ w.r.t. the Boolean function $f \in \boldsymbol{\Sigma}$, which instantiates the innermost placeholder of the term $\boldsymbol{t}$ by the Boolean function f. Below, we give a formal description of the application.

Let us fix a term $\boldsymbol{t}$. According to Proposition 1, $\boldsymbol{t}$ can be equivalently transformed into the following canonical form

$$\rhd^{\ell_k} g_{k-1}\left(\rhd^{\ell_{k-1}} g_{k-2}\left(\cdots g_0\left(\rhd^{\ell_0} \boldsymbol{b}\right)\cdots\right)\right)$$

where $\boldsymbol{b}$ is a vector constant, $\ell_0 \geq 0$ and $\ell_i > 0$ for each $i > 0$. Hereafter, we assume that $\boldsymbol{t}$ is in the canonical form, and let $\mathsf{len}(\boldsymbol{t}) = \sum_{i=0}^{k} \ell_i$.

When $\boldsymbol{t}$ is $\rhd$-free, i.e., $\mathsf{len}(\boldsymbol{t}) = 0$, we let $\boldsymbol{t}[f] = \boldsymbol{t}$. When $\mathsf{len}(\boldsymbol{t}) > 0$, we say that the Boolean function $f \in \boldsymbol{\Sigma}$ is *applicable* w.r.t. the term $\boldsymbol{t}$, if:

1. $\boldsymbol{b} \in \mathsf{dom}\, f$;
2. if $\ell_0 = 1$, then $\mathsf{ran}\, f = \mathsf{dom}\, g_0$.

Intuitively, the above two conditions ensure that $f(\boldsymbol{b})$ and $g_0 \circ f$ are well-defined.

If $f \in \boldsymbol{\Sigma}$ is applicable w.r.t. the term $\boldsymbol{t}$, we let $\boldsymbol{t}[f]$ be the term:

$$\boldsymbol{t}[f] = \begin{cases} \rhd^{\ell_k} g_{k-1}\left(\rhd^{\ell_{k-1}} g_{k-2}\left(\cdots g_0\left(\rhd^{\ell_0 - 1} \boldsymbol{b}'\right)\cdots\right)\right), & \text{if } \ell_0 > 1 \\ \rhd^{\ell_k} g_{k-1}\left(\rhd^{\ell_{k-1}} g_{k-2}\left(\cdots g_1\left(\rhd^{\ell_1} \boldsymbol{b}''\right)\cdots\right)\right), & \text{if } \ell_0 = 1 \end{cases}$$

where $\boldsymbol{b}' = f(\boldsymbol{b})$ and $\boldsymbol{b}'' = (g_0 \circ f)(\boldsymbol{b})$.

It can be seen that $\mathsf{len}(\boldsymbol{t}[f]) = \mathsf{len}(\boldsymbol{t}) - 1$. By iteratively applying this operator, the placeholders $\rhd^k$ in the term $\boldsymbol{t}$ can be eliminated. For convenience, we write $\boldsymbol{t}[f_0, f_1, \ldots, f_i]$ for the shorthand of

$$\boldsymbol{t}[f_0][f_1]\cdots[f_i],$$

provided that each Boolean function f_i is applicable w.r.t. $\boldsymbol{t}[f_0][f_1]\cdots[f_i]$. Likewise, we call $\boldsymbol{t}[f_0, f_1, \ldots, f_i]$ the *application* of $\boldsymbol{t}$ w.r.t. the Boolean functions $f_0, f_1, \cdots, f_i$.

In particular, the *collapsion* of term $\boldsymbol{t}$, denoted by $\boldsymbol{t}\downarrow$, is the term $\boldsymbol{t}[\underbrace{\boldsymbol{I}, \ldots, \boldsymbol{I}}_{\mathsf{len}(\boldsymbol{t})}]$, namely, $\boldsymbol{t}\downarrow$ is obtained from $\boldsymbol{t}$ w.r.t. $\mathsf{len}(\boldsymbol{t})$ identity functions.

We hereafter denote by $\mathsf{Cons}(\boldsymbol{\Sigma})$ the set of constraints $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$ over the signature $\boldsymbol{\Sigma}$ and lift the apply operator $[\,]$ from terms to atomic formulas $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$. For a constraint $\gamma = \boldsymbol{t}_1 \sim \boldsymbol{t}_2 \in \mathsf{Cons}(\boldsymbol{\Sigma})$, we denote by $\gamma[f]$ the constraint $\boldsymbol{t}_1[f] \sim \boldsymbol{t}_2[f]$; and by $\gamma\downarrow$ the constraint $\boldsymbol{t}_1\downarrow \sim \boldsymbol{t}_2\downarrow$. Note that the former implicitly assumes that the Boolean function f is applicable w.r.t. both terms $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ (in this case, we call that f is applicable w.r.t. γ), whereas the latter requires that the terms $\boldsymbol{t}_1\downarrow$ and $\boldsymbol{t}_2\downarrow$ have the same width (we call that $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are *compatible* w.r.t. collapsion). In addition, we let $\mathsf{len}(\gamma) = \max(\mathsf{len}(\boldsymbol{t}_1), \mathsf{len}(\boldsymbol{t}_2))$, and in the case that $\mathsf{len}(\gamma) = 0$, we let $\gamma[f] = \top$ (resp. $\gamma[f] = \bot$) for any Boolean function f if γ is evaluated to true (resp. false).

We subsequently extend the above notations to constraint sets. Suppose that $\Gamma \subseteq \mathsf{Cons}(\boldsymbol{\Sigma})$, we let $\Gamma[f] \overset{\text{def}}{=} \{\gamma[f] \mid \gamma \in \Gamma\}$, and let $\Gamma\downarrow \overset{\text{def}}{=} \{\gamma\downarrow \mid \gamma \in \Gamma\}$. Remind that the notation $\Gamma[f]$ makes sense only if the Boolean function f is *applicable* w.r.t. Γ, namely f is applicable w.r.t. each constraint $\gamma \in \Gamma$. Likewise, the notation $\Gamma\downarrow$ indicates that $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ is compatible w.r.t. collapsion for each constraint $\boldsymbol{t}_1 \sim \boldsymbol{t}_2 \in \Gamma$.

Theorem 1. *For a BNN $\mathcal{N}$ given by $f_{\mathcal{N}} = f_{n-1} \circ f_{n-2} \circ \cdots \circ f_1 \circ f_0$, and a constraint $\gamma \in \mathsf{Cons}(\boldsymbol{\Sigma})$, we have:*

1. *$\mathcal{N}, i \models \gamma$ iff $\mathcal{N}, i+1 \models \gamma[f_i]$ for each $i < n$.*
2. *$\mathcal{N}, i \models \gamma$ iff $\mathcal{N}, i \models \gamma\downarrow$ for each $i \geq n$.*

Indeed, since $\gamma\downarrow$ must have the form $\boldsymbol{b}_1 \sim \boldsymbol{b}_2$, where both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are Boolean constants, then the truth value of $\gamma\downarrow$ can always be directly evaluated.

4.2 Automata Construction

Given a BLTL formula φ in NNF, we can construct a finite-state automaton $\mathcal{A}_\varphi = (Q_\varphi, \boldsymbol{\Sigma}, \delta_\varphi, I_\varphi, F_\varphi)$, where:

– $Q_\varphi = \bigcup_{\Gamma \subseteq \mathsf{Sub}(\varphi)} \mathsf{Cl}(\Gamma)$. Recall that $\mathsf{Cl}(\Gamma) \subseteq 2^{\mathsf{Sub}(\varphi)}$ if $\Gamma \subseteq \mathsf{Sub}(\varphi)$, thus each state must be a subset of $\mathsf{Sub}(\varphi)$.
– For each $q \in Q_\varphi$, let $\mathsf{Cons}(q) \overset{\text{def}}{=} q \cap \mathsf{Cons}(\boldsymbol{\Sigma})$, let $q' = \{\psi \mid \mathsf{X}\psi \in q\}$ and let $q'' = \{\psi \mid \overline{\mathsf{X}}\psi \in q\}$. Then, for each Boolean function $f \in \boldsymbol{\Sigma}$, we have

$$\delta_\varphi(q, f) = \begin{cases} \emptyset, & \bot \in q \\ \mathsf{Cl}(q' \cup q'' \cup \mathsf{Cons}(q)[f]), & \bot \notin q \end{cases}.$$

– $I_\varphi = \{q \in Q_\varphi \mid \varphi \in q\}$ is the set of initial states.
– F_φ is the set of accepting states such that for every state $q \in Q_\varphi$, $q \in F_\varphi$ only if $\{\psi \mid \mathsf{X}\psi \in q\} = \emptyset$, $\bot \notin q$ and $\mathsf{Cons}(q)\downarrow$ is evaluated true.

For a BNN $\mathcal{N}$ given by $f_{\mathcal{N}} = f_{n-1} \circ f_{n-2} \circ \cdots \circ f_1 \circ f_0$, we denote by $\mathcal{N} \in \mathscr{L}(\mathcal{A}_\varphi)$ if the sequence of the Boolean functions $f_0, f_1, \cdots, f_{n-1}$, regarded as a finite word, is accepted by the automaton $\mathcal{A}_\varphi$.

Intuitively, $\mathcal{N}$ accepts an input word iff it has an accepting run $q_0, q_1 \cdots, q_n$, where q_i is constituted with a set of formulas that make the specification φ valid at the position i. In this situation, I_φ refers to the states involving φ and $q_0 \in I_\varphi$. For the transition $q_i \xrightarrow{f_i} q_{i+1}$, q_i' and q_i'' indicate the sets of formulas which should be satisfied in the next position $i+1$ according to the semantics of *next* (X) and *weak next* ($\overline{\mathsf{X}}$). Additionally, $\mathsf{Cons}(q_{i+1})$ is obtained by applying the Boolean function f_i to the constraints in q_i.

The following theorem reveals the relationship between φ and $\mathcal{A}_\varphi$.

Theorem 2. *Let $\mathcal{N}$ be a BNN given by a sequence of Boolean functions for a BLTL formula φ, we have:*

$$\mathcal{N} \models \varphi \textit{ if and only if } \mathcal{N} \in \mathscr{L}(\mathcal{A}_\varphi).$$

The proof is given in Appendix A.1 and an example of the construction is given in Appendix A.2.

4.3 Tableau-Based Construction

We have successfully provided a process for converting an BLTL formula into an automaton on finite words. At first glance, it seems that the model checking problem w.r.t. BNN can be immediately boiled down to a word-problem of finite automata. Nevertheless, a careful analysis shows that this would result in a prohibitively high cost. Actually, for a BLTL formula φ, the state set of $\mathcal{A}_\varphi$ is $\bigcup_{\Gamma \subseteq \mathsf{Sub}(\varphi)} \mathsf{Cl}(\Gamma) \subseteq 2^{\mathsf{Sub}(\varphi)}$, thus the number of states is exponential in the size of the length of φ. To avoid explicit construction, we provide an "on-the-fly" approach when performing synthesis.

Suppose the BLTL φ is given in NNF and the BNN $\mathcal{N}$ is given as a sequence of Boolean functions $f_0, f_1, \ldots, f_{n-1}$, using the following approach, we may construct a tree $\mathcal{T}_{\varphi,\mathcal{N}}$ which fulfills the followings:

– $\mathcal{T}_{\varphi,\mathcal{N}}$ is rooted at $\langle 0, \{\varphi\}\rangle$;
– For an internal node $\langle i, \Gamma\rangle$ with $i < n-1$, it has a child $\langle j, \Gamma'\rangle$ only if there is a tableau rule

$$\begin{array}{c|c} i & \Gamma \\ \hline j & \Gamma' \end{array}$$

where j is either i or $i+1$.
– A leaf $\langle i, \Gamma\rangle$ of $\mathcal{T}_{\varphi,\mathcal{N}}$ is a (Modal)-node with $i = n-1$, where nodes to which only the rule (Modal) can be applied are called (Modal)-nodes.

Tableau rules are listed in Figure 1. For the rule (Modal), we require that Γ consists of atomic formulas being of the form $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$. In the rules (True) and (False), we require that $\mathsf{len}(\boldsymbol{t}_1 \sim \boldsymbol{t}_2) = 0$ and it is evaluated to true and false, respectively.

Suppose $\langle n, \Gamma \cup \{\mathsf{X}\psi_1, \ldots, \mathsf{X}\psi_m\} \cup \{\overline{\mathsf{X}}\varphi_1, \ldots, \overline{\mathsf{X}}\varphi_k\}\rangle$ is a leaf of $\mathcal{T}_{\varphi,\mathcal{N}}$. We say it is *successful* if $m = 0$ and $\Gamma\downarrow$ is evaluated to true. In addition, we say a path

$$(\textsc{And})\quad \begin{array}{c|c} i & \Gamma,\varphi_1\wedge\varphi_2 \\ \hline i & \Gamma,\varphi_1,\varphi_2 \end{array} \qquad (\textsc{Or-}j)\quad \begin{array}{c|c} i & \Gamma,\varphi_1\vee\varphi_2 \\ \hline i & \Gamma,\varphi_j \end{array}\quad (j=1,2)$$

$$(\textsc{True})\quad \begin{array}{c|c} i & \Gamma,\boldsymbol{t}_1\sim\boldsymbol{t}_2 \\ \hline i & \Gamma,\top \end{array} \qquad (\textsc{False})\quad \begin{array}{c|c} i & \Gamma,\boldsymbol{t}_1\sim\boldsymbol{t}_2 \\ \hline i & \Gamma,\bot \end{array}$$

$$(\textsc{Until})\quad \begin{array}{c|c} i & \Gamma,\varphi_1\mathsf{U}\varphi_2 \\ \hline i & \Gamma,\varphi_2\vee(\varphi_1\wedge\mathsf{X}(\varphi_1\mathsf{U}\varphi_2)) \end{array}$$

$$(\textsc{Release})\quad \begin{array}{c|c} i & \Gamma,\varphi_1\mathsf{R}\varphi_2 \\ \hline i & \Gamma,\varphi_2\wedge(\varphi_1\vee\overline{\mathsf{X}}(\varphi_1\mathsf{R}\varphi_2)) \end{array}$$

$$(\textsc{Modal})\quad \begin{array}{c|c} i & \Gamma,\mathsf{X}\psi_1,\ldots,\mathsf{X}\psi_m,\overline{\mathsf{X}}\varphi_1,\ldots,\overline{\mathsf{X}}\varphi_k \\ \hline i+1 & \{\gamma[f_i]\mid\gamma\in\Gamma,\mathsf{len}(\gamma)>0\},\psi_1,\ldots,\psi_m,\varphi_1,\ldots,\varphi_k \end{array}$$

Fig. 1. Tableau rules for Automata Construction

of $\mathcal{T}_{\varphi,\mathcal{N}}$ is *successful* if it ends with a successful leaf, and no node along this path contains $\bot$.

In the process of the on-the-fly construction, we start by creating the root node, then apply the tableau rules to rewrite the formulas in the subsequent nodes. In addition, before the rule (Modal) or (Or-j) is applied, we preserve the set of formulas, which allows us to trace back and construct other parts of the automaton afterward. We exemplify how to achieve the synthesis task via the construction in Section 5.

Theorem 3. *$\mathcal{N}\models\varphi$ if and only if $\mathcal{T}_{\varphi,\mathcal{N}}$ has a successful path.*

Proof. Let $\mathcal{A}_\varphi$ be the automaton corresponding to φ. According to Theorem 2, it suffices to show that $\mathcal{N}\in\mathscr{L}(\mathcal{A}_\varphi)$ iff $\mathcal{T}_{\varphi,\mathcal{N}}$ has a successful path.

Suppose, $\mathcal{N}$ is accepted by $\mathcal{A}_\varphi$ with the run $q_0,q_1,\ldots,q_n$, we also create the root node $\langle 0,\Gamma_0=\{\varphi\}\rangle$. Inductively, we have the followings statements for each node $\langle i,\Gamma_j\rangle$ which is already constructed:

1) $\Gamma_j\subseteq q_i$;
2) $\mathcal{N},i\models\psi$ for each $\psi\in q_i$ (see the proof of Thm. 2)

Then, if $\langle i,\Gamma_j\rangle$ is not a leaf, we create a new node $\langle i',\Gamma'_j\rangle$ in the following way:

– $i'=i$ if $\langle i,\Gamma_j\rangle$ is not a (Modal)-node, otherwise $i'=i+1$;
– if rule (Or-k) ($k=1,2$) is applied to $\langle i,\Gamma_j\rangle$ to some $\varphi_1\vee\varphi_2\in\Gamma_j$, we require that $\varphi_k\in\Gamma_j$; for other cases, Γ'_j is uniquely determined by Γ_j and the tableau rule which is applied.

It can be checked that both Items 1) and 2) still hold at $\langle i',\Gamma'_j\rangle$. Then, we can see that the path we constructed is successful since q_n is an accepting state of $\mathcal{A}_\varphi$.

For the other way round, suppose that $\mathcal{T}_{\varphi,\mathcal{N}}$ involves a successful path

$$\langle 0,\Gamma_{0,0}\rangle,\langle 0,\Gamma_{0,1}\rangle,\ldots,\langle 0,\Gamma_{0,\ell_0}\rangle,\langle 1,\Gamma_{1,0}\rangle,\langle 1,\Gamma_{1,1}\rangle,\ldots,\langle 1,\Gamma_{1,\ell_1}\rangle,\ldots,$$
$$\langle i,\Gamma_{i,0}\rangle,\langle i,\Gamma_{i,1}\rangle,\ldots,\langle i,\Gamma_{i,\ell_i}\rangle,\ldots,\langle n,\Gamma_{n,0}\rangle,\langle n,\Gamma_{n,1}\rangle,\ldots,\langle n,\Gamma_{n,\ell_n}\rangle$$

then, the state sequence $q_0, q_1, \ldots, \ldots, q_n$ yields an accepting run of $\mathcal{A}_\varphi$ on $\mathcal{N}$, where $q_i = \bigcup_{j=0}^{\ell_i} \varGamma_{i,j}$. □

5 BNN Synthesis

Let us now consider a more challenging task: Given a BLTL specification φ, to find some BNN $\mathcal{N}$ such that $\mathcal{N} \models \varphi$. In the synthesis task, the parameters of the desired BNN are not given, even, we are not aware of the length (i.e., the number of blocks) of the network. To address this challenge, we leverage the tableau-based method (cf. Section 4.3) to construct the automaton for the given specification φ and check the existence of the desired BNN at the same time. But when performing the tableau-based rewriting, we need to view each block (i.e., a Boolean function) f_i as an unknown variable (called *block variable* in what follows).

The construction of the tableau-tree starts from the root node $\langle 0, \varphi \rangle$ in a depth-first search manner. During the construction, for each internal node $\langle i, \varGamma \rangle$, the following steps are taken: Firstly, rules other than (Or-1) and (Modal) are applied to $\varGamma$ until no further changes occur. Then rule (Or-j) is applied to the disjunctions in the formula set, and we always first try rule (Or-1) when the rewriting is performed. Lastly, rule (Modal) is applied to generate node $\langle i+1, \varGamma' \rangle$, which becomes the next node in the path, and the Boolean function f_i used in the rewriting is just a block variable. Particularly, we retain a stack of nodes on which either rule (Or-j) or (Modal) is applied for backtracing. A node is called a (Or-j) node if rule (Or-j) is applied onto it. Once an X-free (Modal)-node is reached, we verify the success of the path. However, since now the blocks are no longer concrete in this setting, an atomic formula of the form $\gamma[f_i, \ldots, f_{i+k}]$ cannot be immediately evaluated even if it is $\triangleright$-free. As a result, whether a path is *successful* cannot be evaluated directly.

To address this issue, we invoke an integer different logic (IDL) solver to examine the satisfiability of the atomic formulas in the (Modal)-nodes along the path, and we declare success if all of them are satisfiable and in addition, it ends up with an X-free (Modal)-node. Meanwhile, the model given by the solver would reveal hyper-parameters of the BNN, which then we adopt to obtain the expected BNN. For a node $\langle i, \varGamma \rangle$, we call i to be the *depth counter*. Once the infeasibility is reported by the IDL solver, or some specific depth counter (call it the *threshold*) is reached, a trace-back to the nearest (Or-1) node is required: all the nodes under the nearest (Or-1) node (including itself) are popped from the stack and then apply rule (Or-2) to that node (it becomes a (Or-2) node), but this time we do not push anything into the stack, because both choices for the disjunctive formula have been tried so far. If no (Or-1) nodes remains in the stack when doing trace-back, we declare the failure of the synthesis.

Now, there are two issues to deal with during that process. The first is, how to determine if the aforementioned 'threshold' is reached; second, how can we convert the satisfiability testing into IDL-solving.

5.1 The Threshold

There exists a naïve bound for the first problem, which is just the state number of $\mathcal{A}_\varphi$. However, this bound is in general not compact (i.e., doubly exponential in the size of the formula φ), and thus we provide a tighter bound.

We first introduce the following notion. Two modal nodes $\langle i, \Gamma\rangle$ and $\langle j, \Gamma'\rangle$ are *isomorphic*, denoted by $\langle i, \Gamma\rangle \cong \langle j, \Gamma'\rangle$, if Γ can be transformed into Γ' under a (block) variable bijection. The following lemma about isomorphic model nodes is straightforward.

Lemma 1. *If $\langle i, \Gamma\rangle \cong \langle j, \Gamma'\rangle$ and the node $\langle i, \Gamma\rangle$ could lead to a successful leaf (i.e., satisfiable leaf), then so does the node $\langle j, \Gamma'\rangle$.*

Thus, given φ, the threshold can be the number of equivalence classes w.r.t. $\cong$. To make the analysis clearer, we here introduce some auxiliary notions.

- We call an atomic constraint γ occurring in φ to be an *original constraint* (or, *non-padded constraint*); and call a formula being of the form $\gamma[f_i, \ldots, f_j]$ *padded constraint*, where $f_i, \ldots, f_j$ are block variables.
- A (padded or non-padded) constraint with length 0 (i.e., $\rhd$-free) is called *saturated*. In general, such a constraint is obtained from a non-padded constraint γ via applying k layer variables, where $k = \mathsf{len}(\gamma)$.

Theorem 4. *Let φ be a closed BLTL formula, and let*

- $c = \#(\mathsf{Cons}(\boldsymbol{\Sigma}) \cap \mathsf{Sub}(\varphi))$*, i.e., the number of (non-padded) constraints occurring in φ;*
- $k = \max\{\mathsf{len}(\gamma) \mid \gamma \in \mathsf{Cons}(\boldsymbol{\Sigma}) \cap \mathsf{Sub}(\varphi)\}$*, i.e., the maximum length of non-padded constraints occurring in φ;*
- *p be the number of temporal operators in φ*

then, $2^{(k+1)c+p} + 1$ is a threshold for synthesis.

The proof is shown in Appendix A.3.

5.2 Encoding with IDL Problem

Another problem is how to convert the satisfiability testing into IDL-solving. To tackle this problem, we present a method that transforms BLTL atomic formulas to IDL constraints.

We may temporarily view a Boolean function $g : \mathbb{B}^m \to \mathbb{B}^n$ as a (partial) integer function with domain $[2^m]$, namely, we equivalently view g maps $\mathsf{dec}(\boldsymbol{b})$ to $\mathsf{dec}(g(\boldsymbol{b}))$.

For a $\rhd$-free term $\boldsymbol{t} = (f_k \circ f_{k-1} \circ \cdots \circ f_0)(\boldsymbol{b})$, we say that $(f_i \circ f_{i-1} \circ \cdots \circ f_0)(\boldsymbol{b})$ is an *intermediate term* of $\boldsymbol{t}$ where $i \le k$. In what follows, we denote by $\boldsymbol{T}$ the set of all intermediate terms that may occur in the process of IDL-solving, which is a part of synthesis that check the satisfiability of atomic formulas in successful leaves.

Remind that in a term or an intermediate term, a symbol g may either be a fixed function or a variable that needs to be determined by the IDL-solver (i.e., *block variables*). To make it clearer, we in general use $g_0, g_1, \ldots$ to designate the former functions, whereas use f_0, f_1, etc for the latter cases.

The theory of IDL is limited to handling the Boolean combinations of the form $x - y \sim c$, where x, y are integer variables and c is an integer constant. However, since functions occur in the terms, they cannot be expressed using IDL. To this end, we note that we merely care about partial input-output relations of the functions, which consist of mappings among $\boldsymbol{T}$, and then the finite mappings can be expressed by integer constraints. Thus, for each intermediate term $\boldsymbol{t} \in \boldsymbol{T}$, we introduce an integer variable $v_{\boldsymbol{t}}$.

Then, all constraints describing the synthesis task are listed as follows.

(1) For each BLTL constraint $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$, we have a conjunct $v_{\boldsymbol{t}_1} \sim v_{\boldsymbol{t}_2}$.
(2) For each block variable $f : \mathbb{B}^n \to \mathbb{B}^m$ and each $f(\boldsymbol{t}) \in \boldsymbol{T}$, we add the bound constraints $0 \leq v_{f(\boldsymbol{t})}$ and $v_{f(\boldsymbol{t})} \leq 2^m$.
(3) For each block variable f and every pair of terms $\boldsymbol{t}_1, \boldsymbol{t}_2 \in \boldsymbol{T}$, we have the constraint: $v_{\boldsymbol{t}_1} = v_{\boldsymbol{t}_2} \to v_{f(\boldsymbol{v}_1)} = v_{f(\boldsymbol{v}_2)}$, which guarantees f to be a mapping.
(4) For every fixed function g, we impose the constraint $v_{g(\boldsymbol{t})} = \mathsf{dec}(g(\mathsf{bin}(v_{\boldsymbol{t}})))$ for every $\boldsymbol{t} \in \boldsymbol{T}$.

Once the satisfiability is reported by the IDL-solver, we extract partial mapping information of f_i's from the solver's model, by analyzing equations of the form $v_{\boldsymbol{t}} = c$, where c is an integer called the value of $\boldsymbol{t}$. We iterate over the model and record the value of terms, when we encounter an equation in the form of $v_{f_i(\boldsymbol{t})} = c$, we query the value of $\boldsymbol{t}$, and obtain one input-output relation of f_i. Eventually, we get partial essential mapping information of such f_i's.

5.3 Utilize the Synthesis

A BNN that satisfies the specification can be obtained via block-wise training, namely, training each block independently to fulfill its generated input-output mapping relation, which is extracted by the IDL-solver during the synthesis process. Indeed, such training is not only in general lightweight but also able to reuse the pre-trained blocks.

Let us now consider a more general requirement that we have both high-level temporal specification (such as fairness, robustness) and data constraints (i.e., labels on a dataset), and is asked to obtain a BNN to meet all these obligations.

A straightforward idea is to express all data constraints with BLTL, and then perform a monolithic synthesis. However, such a solution seems to be infeasible, because the large amount of data constraints usually produces a rather complicated formula, and it makes the synthesis extremely difficult.

An alternative approach is to first perform the synthesis w.r.t. the high-level specification, then do a retraining upon the dataset. However, the second phase may distort the result of the first phase. In general, one need to conduct an iterative cycle composed of synthesis-training-verification, yet the convergence of such process cannot be guaranteed. Thus, we need make a trade-off between these two types of specifications.

More practically, synthesis can be used as an “enhancement” procedure. Suppose, we already have some BNN trained with the given dataset, then we are aware the hyper-parameters of that. This time, we have more information when doing synthesis, e.g., the threshold is replaced by the length of the network, and the shape (i.e., the width of input and output) of each block are also given. With this, we may perform a more effective IDL-solving process, and then retrain each block individually. Definitely, this might affect the accuracy of network, and some compromise also should be done.

6 Experimental Evaluation

We implement a prototype tool in Python, which uses Z3 [9] as the off-the-shelf IDL solver and PyTorch to train blocks and BNNs. To the best of our knowledge, few existing work on synthesizing BNN has been done so far. Hence, we mainly investigate the feasibility of our approach by exploring how much the trustworthiness of BNN can be enhanced, and the corresponding trade-off on accuracy degradation. The first two experiments focus on evaluating the effectiveness of synthesis in enhancing the properties of BNNs. We set BNNs with diverse architectures as baselines, and synthesize models via the “enhancement” procedure, wherein the threshold matches the length of the baselines, and the shape of blocks are constrained to maintain the same architecture as the baselines. Eventually, the blocks are retrained to fulfill the partial mapping, and the synthesized model is obtained through retraining on the dataset. We compare the synthesized models and their baselines on two properties: *local robustness* and *individual fairness*. Moreover, we also study the potential of our approach to assist in determining the network architecture.

Datasets. We train models and evaluate our approach over two classical datasets, MNIST [10] and UCI Adult [11]. MNIST is a dataset of handwritten digits, which contains 70,000 gray-scale images with 10 classes, and each image has 28×28 pixels. In the experiments, we downscale the images to 10×10, and binarize the normalized images, and then transform them into 100-width vectors. UCI Adult contains 48,842 entries with 14 attributes, such as age, gender, workclass and occupation. The classification task on the dataset UCI Adult is to predict whether an individual’s annual salary is greater than 50K. We first remove unusable data, retain 45,221 entries, and then transform the real-value data into 66-dimension binarized vectors as input.

Experimental Setup. In the block-wise training, different loss functions are employed for internal and output blocks: the MSE loss function for internal blocks and the cross-entropy loss function for output blocks. The training process entails a fixed number of epochs, with 150 epochs for internal blocks and 30 epochs for output blocks. The experiments are conducted on a 3.6G HZ CPU with 12 cores and 32GB RAM, and the blocks and BNNs are trained using a single GeForce RTX 3070 Ti GPU.

Table 1. BNN baselines.

Name	Arch	Acc	Name	Arch	Acc
R1	100-32-10	82.62%	**F1**	66-32-2	80.12%
R2	100-50-10	84.28%	**F2**	66-20-2	79.88%
R3	100-50-32-10	83.50%	**F3**	66-32-20-2	78.13%

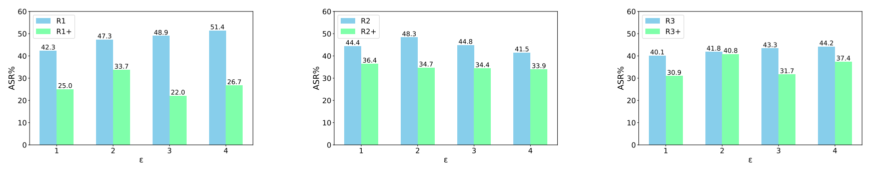
Baseline. We use six neural networks with different architectures as baselines, where three models **R1-R3** are trained on the MNIST for 10 epochs with a learning rate of 10^{-4} to study local robustness. For individual fairness, we train 3 models (**F1-F3**) on the UCI Adult for 10 epochs, with a learning rate of 10^{-3}, and split the dataset into a training set and a test set in a 4:1 ratio. The detailed information is listed in Table 1, Column (Name) indicates the name of BNNs, and Column (Arch) presents their architectures. The architecture of each network is described as by a sequence $\{n_i\}_{i=0}^{s}$, where s is the number of the blocks in the network, and n_i and n_{i+1} indicate the input and output dimensions of the i-th block. For instance, 100-32-10 indicates that the BNN has two blocks, the input dimensions of these blocks are 100 and 32 respectively, and the number of classification labels is 10. Column (Acc) shows the accuracy of the models on the test set.

6.1 Local Robustness

In this section, we evaluate the effectiveness of our approach for enhancing the robustness of models in different cases. We use the metric, called Adversarial Attack Success Rate (ASR), to measure a model's resistance to adversarial attacks. ASR is calculated as the proportion of perturbed inputs that leads to a different prediction result compared to the original input.

We choose 30 image vectors from the training set, and set the maximum perturbation to four levels, $\epsilon \in \{1, 2, 3, 4\}$. The value of ϵ indicates the maximum number of positions that can be modified in one image vector. One selected input vector, one maximum perturbation ϵ and one baseline model constitute a case, resulting in a total of 360 cases.

For each of the 360 cases, we make a synthesized model individually and compare its ASR with the corresponding baseline. For the local robustness property (cf. Section 3.2), since the input space is too large to enumerate, we need to sample inputs within $B(\boldsymbol{u}, \epsilon)$ when describing the specification, which is formulated as $\bigwedge_{i=1}^{k} (\mathcal{N}(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{b}_i))$, where each $\boldsymbol{b}_i$ is a sample and k is the number of samples. We here sample 100 points within the maximum perturbation limit ϵ. The specification is written as $\bigwedge_{i=1}^{k} (\rhd^n \boldsymbol{u} = \rhd^n \boldsymbol{b}_i)$, where n is the number of the block of the baseline. Subsequently, we use the block constraint (cf. Section 5.2), $0 \leq v_{f_i(\boldsymbol{t})} \leq 2^m$, to specify the range of output of each block. To make the bound tighter, we retain the maximal and minimal activations of each block



(a) Arch:100-32-10 (b) Arch:100-50-10 (c) Arch:100-50-32-10

Fig. 2. Results of local robustness.

using calibration data run on the baseline, and then take the recorded values as bounds. Eventually, the generated mappings are used in the block-wise training, and then the enhanced BNN is obtained through retraining on the MNIST dataset.

We also take 100 samples for each case and compare the ASR for baselines and their synthesized counterparts. The results are shown in Figure 2, where blue bars represent the baselines, while green bars represent synthesized models. We use the sign + to denote the synthesized models. Figure 2(a) (resp. Figure 2(b) and Figure 2(c)) depicts the percentage of average ASR of **R1** (resp. **R2** and **R3**) and the counterpart **R1+** (resp. **R2+** and **R3+**) (the vertical axis), with different ϵ (1, 2, 3, 4) (the horizontal axis). The results demonstrate a decrease in ASR by an average of 43.45%, 22.12%, and 16.95% for **R1**, **R2** and **R3**, respectively.

Whilst the models' robustness are enhanced, their accuracy are slightly decreased. Table 2 shows the results of the accuracy of the models, where Acc+ represents the average accuracy for synthesized models with the same architectures.

Table 2. The average accuracy of **R1**-**R3** and their synthesized models.

	R1	**R2**	**R3**
Acc	82.62%	84.28%	83.50%
Acc+	81.33%	81.72%	78.75%

6.2 Individual Fairness

In this section, we investigate the individual fairness w.r.t two sensitive features, namely, sex (Male and Female) and race (White and Black) on the UCI Adult dataset.

We consider **F1**-**F3** as baselines, and randomly select 1000 entries for both **F1** and **F2**, and 200 entries for **F3** from the training dataset, and then generate proper pairs by modifying the value of the sensitive attribute while keeping all other attributes the same. For example, we modify the value of Male to Female. After forming specifications using the approach mentioned in Section 3.2 with the pairs, we proceed with the "enhancement" procedure and retraining to obtain

Table 3. Results of individual fairness.

Model	Feature	Acc	Acc+	Fair	Fair+	Synthesis Time(s)
F1	sex	80.12%	74.53%	92.91%	99.94%	241.67
F1	race	80.12%	74.54%	92.92%	100%	216.46
F2	sex	79.88%	75.71%	95.68%	97.83%	215.61
F2	race	79.88%	75.18%	94.64%	98.47%	212.46
F3	sex	78.13%	74.48%	89.67%	99.83%	90.39
F3	race	79.88%	74.09%	89.16%	98.27%	95.75

the synthesized models. We then evaluate the models on the test dataset by measuring the fairness score. We count the number of the fair pairs (i.e., the pairs only differ in the sensitive attribute, and yield the same predication result): *fair num*, and compute the fairness score, $\frac{fair\ num}{test\ size}$, where *test size* is the size of the test set.

The results are listed in Table 3, where the baselines and the sensitive attributes are shown in Columns 1,2. Columns 3,4 (Acc/Acc+) demonstrate the accuracy of baselines and synthesized models, and Columns 5,6 (Fair/Fair+) show their fairness scores. The results show that all the models' individual fairness is significantly improved, some of which even reach 100% (e.g., Row 2, the fairness score increase from 92.92% to 100%). However, the enhancement is accompanied by the accuracy loss, Columns 3,4 show that all models suffer from a certain degree of accuracy decrease. Our tool efficiently synthesized the hyper-parameters within a few minutes, as shown in Column 7.

Furthermore, we examine the ability of our approach on helping determine the architecture of the BNNs. For both sex and race, we sample 200 entries in the training dataset to generate proper pairs, and formulate the specification without using the bound constraints or fixing the number of block, as follows,

$$\mathsf{F}(\bigwedge_i^k(\boldsymbol{x}_i = \boldsymbol{y}_i)) \wedge (\bigwedge_i^k(\boldsymbol{x}_i = \triangleright^2\boldsymbol{a}_i \wedge \boldsymbol{y}_i = \triangleright^2\boldsymbol{b}_i) \vee (\bigwedge_i^k(\boldsymbol{x}_i = \triangleright^3\boldsymbol{a}_i \wedge \boldsymbol{y}_i = \triangleright^3\boldsymbol{b}_i)))$$

where $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ is the proper pair, and k is the number of samples. The formula indicates the presence of consecutive blocks in the model, with a length of either 2 or 3. For each proper pair $(\boldsymbol{a}_i, \boldsymbol{b}_i)$, their respective outputs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ must be equal.

After synthesizing the partial input-output relation of block functions f_i's, we determine the length of the network by selecting the maximum i among the block functions f_i's. The dimensions of the blocks are set to the maximum input and output dimensions in the partial relation obtained for the corresponding f_i.

We make a slight adjustment to the synthesis framework, when finding a group of hyper-parameters, we continue searching for one more feasible group, resulting in two groups of hyper-parameters for sex and race. We showcase the synthesized models in Table 4. Column 1 indicates the sensitive attribute of interest, and Columns 2,3 give the architecture and the length of the BNNs respectively. Column 4 shows the number of partial mappings we obtained in the

Table 4. The synthesized models whose architectures are given by our tool.

Attr	**Arch**	**Len**	**#Mapping**	**Acc**	**Fair**
sex	66-10-10-2	3	1117	74.38%	99.51%
sex	66-8-2	2	559	74.69%	99.72%
race	66-9-8-2	3	952	74.38%	94.59%
race	66-8-2	2	567	74.13%	99.71%

synthesis task. Our tool successfully generates models with varying architectures and high individual fairness, which are presented in Columns 5,6 respectively.

7 Conclusion

In this paper, we have presented an automata-based approach to synthesizing binarized neural networks. Specifying BNNs' properties with the designed logic BLTL, the synthesis framework uses the tableau-based construction approach and the IDL-solver to determine hyper-parameters of BNNs and relations among some parameters. Subsequently, we may perform a block-wise training. We implemented a prototype tool and the experiments demonstrate the effectiveness of our approach in enhancing the local robustness and individual fairness of BNNs. Although our approach shows the feasibility of synthesizing trustworthy BNNs, there is still a need to further explore this line of work. In the future, beyond the input-output relation of BNNs, we plan to focus on specifying properties between the intermediate blocks. Additionally, we aim to extend the approach to handle the synthesis task of multi-bits QNNs.

Acknowledgements This work is partially supported by the National Key R & D Program of China (2022YFA1005101), the National Natural Science Foundation of China (61872371, 62072309, 62032024), CAS Project for Young Scientists in Basic Research (YSBR-040), and ISCAS New Cultivation Project (ISCAS-PYFX-202201).

References

1. Baluta, T., Shen, S., Shinde, S., Meel, K.S., Saxena, P.: Quantitative verification of neural networks and its security applications. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1249–1264 (2019)
2. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-lib standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK). vol. 13, p. 14 (2010)
3. Bu, L., Zhao, Z., Duan, Y., Song, F.: Taking care of the discretization problem: A comprehensive study of the discretization problem and a black-box adversarial attack in discrete integer domain. IEEE Trans. Dependable Secur. Comput. **19**(5), 3200–3217 (2022)

4. Chen, G., Chen, S., Fan, L., Du, X., Zhao, Z., Song, F., Liu, Y.: Who is real bob? adversarial attacks on speaker recognition systems. In: Proceedings of the 42nd IEEE Symposium on Security and Privacy (SP). pp. 694–711 (2021)
5. Chen, G., Zhang, Y., Zhao, Z., Song, F.: Qfa2sr: Query-free adversarial transfer attacks to speaker recognition systems. In: Proceedings of the 32nd USENIX Security Symposium (2023)
6. Chen, G., Zhao, Z., Song, F., Chen, S., Fan, L., , Wang, F., Wang, J.: Towards understanding and mitigating audio adversarial examples for speaker recognition. IEEE Trans. Dependable Secur. Comput. pp. 1–17 (2022)
7. Chen, G., Zhao, Z., Song, F., Chen, S., Fan, L., Liu, Y.: AS2T: Arbitrary source-to-target adversarial attack on speaker recognition systems. IEEE Trans. Dependable Secur. Comput. pp. 1–17 (2022)
8. Cheng, C.H., Nührenberg, G., Huang, C.H., Ruess, H.: Verification of binarized neural networks via inter-neuron factoring: (short paper). In: Verified Software. Theories, Tools, and Experiments: 10th International Conference, VSTTE 2018, Oxford, UK, July 18–19, 2018, Revised Selected Papers 10. pp. 279–290. Springer (2018)
9. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14. pp. 337–340. Springer (2008)
10. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
11. Dua, D., Graff, C.: UCI machine learning repository (2017), `http://archive.ics.uci.edu/ml`
12. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1625–1634 (2018)
13. Giacobbe, M., Henzinger, T.A., Lechner, M.: How many bits does it take to quantize your neural network? In: Tools and Algorithms for the Construction and Analysis of Systems: 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings, Part II 26. pp. 79–97. Springer (2020)
14. Gpt-4, `https://openai.com/product/gpt-4`
15. Guo, X., Wan, W., Zhang, Z., Zhang, M., Song, F., Wen, X.: Eager falsification for accelerating robustness verification of deep neural networks. In: Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering. pp. 345–356 (2021)
16. Henzinger, T.A., Lechner, M., Zikelic, D.: Scalable verification of quantized neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 3787–3795 (2021)
17. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020)
18. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification.

In: Static Analysis: 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings 26. pp. 296–319. Springer (2019)
19. Liang, Z., Ren, D., Liu, W., Wang, J., Yang, W., Xue, B.: Safety verification for neural networks based on set-boundary analysis. In: David, C., Sun, M. (eds.) Theoretical Aspects of Software Engineering. pp. 248–267. Springer Nature Switzerland, Cham (2023)
20. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for verifying deep neural networks. Foundations and Trends® in Optimization **4**(3-4), 244–404 (2021)
21. Liu, W.W., Song, F., Zhang, T.H.R., Wang, J.: Verifying relu neural networks from a model checking perspective. Journal of Computer Science and Technology **35**, 1365–1381 (2020)
22. Lösbrock, C.D.: Implementing an incremental solver for difference logic. Master's thesis, RWTH Aachen university (2018)
23. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., Van Baalen, M., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)
24. Narodytska, N., Kasiviswanathan, S., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)
25. Narodytska, N., Zhang, H., Gupta, A., Walsh, T.: In search for a SAT-friendly binarized neural network architecture. In: International Conference on Learning Representations (2020)
26. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by angluin-style learning. In: Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22. pp. 354–370. Springer (2019)
27. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
28. Song, F., Lei, Y., Chen, S., Fan, L., Liu, Y.: Advanced evasion attacks and mitigations on practical ml-based phishing website classifiers. Int. J. Intell. Syst. **36**(9), 5210–5240 (2021)
29. Fsd chip-tesla, `https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip`
30. Zhang, P., Wang, J., Sun, J., Dong, G., Wang, X., Wang, X., Dong, J.S., Dai, T.: White-box fairness testing through adversarial sampling. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 949–960 (2020)
31. Zhang, Y., Song, F., Sun, J.: QEBVerif: Quantization error bound verification of neural networks. In: Proceedings of the 35th International Conference on Computer Aided Verification. pp. 413–437 (2023)
32. Zhang, Y., Zhao, Z., Chen, G., Song, F., Chen, T.: BDD4BNN: a BDD-based quantitative analysis framework for binarized neural networks. In: Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33. pp. 175–200. Springer (2021)
33. Zhang, Y., Zhao, Z., Chen, G., Song, F., Chen, T.: Precise quantitative analysis of binarized neural networks: A bdd-based approach. ACM Trans. Softw. Eng. Methodol. **32**(3) (2023)
34. Zhang, Y., Zhao, Z., Chen, G., Song, F., Zhang, M., Chen, T., Sun, J.: QVIP: an ilp-based formal verification approach for quantized neural networks. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. pp. 1–13 (2022)

35. Zhao, Z., Chen, G., Wang, J., Yang, Y., Song, F., Sun, J.: Attack as defense: characterizing adversarial examples using robustness. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). pp. 42–55 (2021)
36. Zhao, Z., Zhang, Y., Chen, G., Song, F., Chen, T., Liu, J.: CLEVEREST: accelerating cegar-based neural network verification via adversarial attacks. In: Proceedings of the 29th International Symposium on Static Analysis. pp. 449–473 (2022)
37. Zheng, H., Chen, Z., Du, T., Zhang, X., Cheng, Y., Ji, S., Wang, J., Yu, Y., Chen, J.: Neuronfair: Interpretable white-box fairness testing through biased neuron identification. In: Proceedings of the 44th International Conference on Software Engineering. pp. 1519–1531 (2022)

A Appendix

A.1 Proof of Theorem 2

Proof. Still let $f_0, f_1, \cdots, f_{n-1}$ be the encoding of a BNN $\mathcal{N}$:
($\Rightarrow$): Suppose that $\mathcal{N} \in \mathscr{L}(\mathcal{A}_\varphi)$, and $q_0, q_1, \cdots, q_n$ be an accepting run of $\mathcal{A}_\varphi$ on $\mathcal{N}$, remind that each q_i is a formula set, by induction on both the index (in the backward way) and formulas' structure, we prove the following claim:

$$\mathcal{N}, i \implies \psi \text{ for ever } \psi \in q_i.$$

– The case is trivial if $\psi = \top$; and $\psi \neq \bot$ since $q_0, q_1, \ldots q_n$ is an accepting run.
– If $\psi = \boldsymbol{t}_1 \sim \boldsymbol{t}_2$, we need to distinguish two cases:
  1) When $i = n$, then $\psi \in \mathsf{Cons}(q_n)$. We have that $\psi\downarrow$ is evaluated to true, because $q_n \in F_\varphi$. Therefore, $\mathcal{N}, n \models \psi$ holds from Thm. 1.
  2) If $i < n$, then we have $\psi[f_i] \in q_{i+1}$ according to the automaton construction. Inductively, we have $\mathcal{N}, i+1 \models \psi[f_i]$, and we can then conclude that $\mathcal{N}, n \models \psi$ according to Thm. 1.
– If $\psi = \psi_1 \wedge \psi_2$, then we have $\psi_1 \in q_i$ and $\psi_2 \in q_i$, because q_i is some proper closure. Thus $\mathcal{N}, i \models \psi_j$ holds for $j = 1, 2$ by induction.
– The case of $\psi = \psi_1 \vee \psi_2$ are similar to the above.
– If $\psi = \psi_1 \mathsf{U} \psi_2$, then $\psi_2 \vee (\psi_1 \wedge \mathsf{X}(\psi_1 \mathsf{U} \psi_2)) \in q_i$, and subsequently either $\psi_2 \in q_i$ or both $\psi_1 \in q_i$ and $\mathsf{X}(\psi_1 \mathsf{U} \psi_2) \in q_i$. For the former case, we can ensure that $\mathcal{N}, i \models \psi_2$. For the latter case, since $\mathsf{X}(\psi_1 \mathsf{U} \psi_2) \in q_i$ we can guarantee that $q_i \notin F_\varphi$ and $i < n$, subsequently $\psi_1 \mathsf{U} \psi_2 \in q_{i+1}$. Therefore, we in this case have both $\mathcal{N}, i \models \psi_1$ and $\mathcal{N}, i+1 \models \psi_1 \mathsf{U} \psi_2$ by induction.
– If $\psi = \psi_1 \mathsf{R} \psi_2$ then $\psi_2 \wedge (\psi_1 \vee \overline{\mathsf{X}}(\psi_1 \mathsf{R} \psi_2)) \in q_i$, which indicates that either $\psi_1, \psi_2 \in q_i$ or $\psi_2, \overline{\mathsf{X}}(\psi_1 \mathsf{R} \psi_2) \in q_i$. For the former case, we can easily infer $\mathcal{N}, i \models \psi_j$ for $j = 1, 2$ by induction, which implies $\mathcal{N}, i \models \psi$ holds. For the latter case, first, we have $\mathcal{N}, i \models \psi_2$; in addition, we have $\overline{\mathsf{X}}(\psi_1 \mathsf{R} \psi_2) \in q_i$, and it could be distinguished by two cases:
  1) $i = n$, then $\mathcal{N}, i \models \overline{\mathsf{X}}(\psi_1 \mathsf{R} \psi_2)$ trivially holds in this case;
  2) $i < n$, then we have $\psi_1 \mathsf{R} \psi_2 \in q_{i+1}$, and we also have $\mathcal{N}, i \models \overline{\mathsf{X}}(\psi_1 \mathsf{R} \psi_2)$ by induction.

($\Leftarrow$): On the other way round, suppose that $\mathcal{N} \models \varphi$, then let

$$q_i = \{\psi \in \mathsf{Sub}(\varphi) \mid \mathcal{N}, i \models \psi\}$$

for each $i \leq n$.

We first show that that each q_i is some proper closure of some subset of $\mathsf{Sub}(\varphi)$. Therefore, we have $q_i \in Q_\varphi$ for each i.

- If $\psi_1 \wedge \psi_2 \in q_i$, then $\mathcal{N}, i \models \psi_j$ for $j = 1, 2$, thus both ψ_1 and ψ_2 are in q_i.
- If $\psi_1 \vee \psi_2 \in q_i$, then either $\mathcal{N}, i \models \psi_1$ or $\mathcal{N}, i \models \psi_2$, which implies that $\psi_1 \in q_i$ or $\psi_2 \in q_i$.
- Suppose that $\psi_1 \mathsf{U} \psi_2 \in q_i$, we can immediately infer that $\psi_2 \vee (\psi_1 \wedge \mathsf{X}(\psi_1 \mathsf{U} \psi_2)) \in q_i$ according to Proposition 2.
- Similar for the case for the formula $\psi_1 \mathsf{R} \psi_2$.

Thus, we can conclude that each $q_i \in Q_\varphi$. Next, we also need to show that $q_{i+1} \in \delta(q_i, f_i)$ for every $i < n$.

- First of all, since q_i consists of closed formulas which is satisfied by $\mathcal{N}$ at step i, we can conclude that $\bot \notin q_i$.
- For each constraint $\gamma \in \mathsf{Cons}(q_i)$, according to Theorem 1 and the construction, we have $\gamma(f_i) \in q_{i+1}$. Therefore, $\mathsf{Cons}(q_i)[f_i] \subseteq q_{i+1}$.
- For each $\mathsf{X}\psi \in q_i$, since $i < n$ and $\mathcal{N}, i \models \mathsf{X}\psi$, then we have $\psi \in q_{i+1}$ and subsequently $q_i' \subseteq q_{i+1}$ (cf. the automaton construction). Likewise, we can also infer that $q_i'' \subseteq q_{i+1}$.
- Therefore, $q_{i+1} \in \mathsf{Cl}(\mathsf{Cons}(q_i)[f_i] \cup q_i' \cup q_i'')$, because q_{i+1} must be some proper closure.

In addition, we have $\varphi \in q_0$ because $\mathcal{N} \models \varphi$, and we thus have $q_0 \in I_\varphi$. Moreover, we claim that $q_n \in F_\varphi$ due to $\mathcal{N}, n \models \psi$ for each $\psi \in q_n$, in detail:

- $\gamma \downarrow$ has to evaluated to true, if $\gamma \in \mathsf{Cons}(q_n)$, according to Theorem 1;
- q_n' must be $\emptyset$ according to the semantics definition on X operator.

Then, we can conclude that $q_0, q_1, \ldots, q_n$ is an accepting run of $\mathcal{A}_\varphi$ on $\mathcal{N}$.

A.2 Example of the Automata Construction

Consider the BLTL formula $\varphi = \mathsf{X}\left((\triangleright \boldsymbol{a} = \triangleright \boldsymbol{b}) \wedge \mathsf{X}\left(\boldsymbol{a} = \boldsymbol{c}\right)\right) \vee \triangleright \boldsymbol{a} \leq \boldsymbol{b}$. We exemplify the automata construction using φ. The constructed automaton is shown in Fig 3, where $q_1, q_3 \in I_\varphi$, and $q_2 \in F_\varphi$. It is easy to see that q_0, q_1, q_2 is an accepting path over the input f_1, f_2. We note that $q_4 \notin F_\varphi$, since the formula $f_1(a) \leq b$ is evaluated false.

A.3 Proof of Theorem 4

Proof. Observe that a counter remains unchanged a (Modal)-node is encountered, thus we concentrate to the slicing of the tableau consisting of (Modal)-nodes only. First of all, we have the following observations:

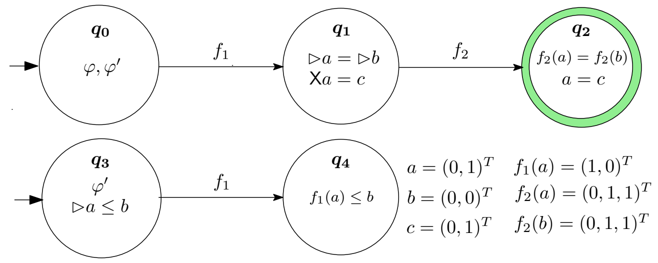


Fig. 3. The automaton for $\varphi = \mathsf{X}((\rhd \boldsymbol{a} = \rhd \boldsymbol{b}) \wedge \mathsf{X}(\boldsymbol{a} = \boldsymbol{c})) \vee \rhd \boldsymbol{a} \leq \boldsymbol{b}$, where $\varphi' = \mathsf{X}((\rhd \boldsymbol{a} = \rhd \boldsymbol{b}) \wedge \mathsf{X}(\boldsymbol{a} = \boldsymbol{c}))$.

- Suppose, $\langle i, \Gamma \rangle$ is a (Modal)-node, a (padded or non-padded, but not saturated) constraint $\gamma \in \Gamma$, then we ensure that $\gamma[f_i]$ must occur in the next (Modal)-node whose counter is $i + 1$.
- Thus, each padded constraint in a (Modal)-node must be of the form $\gamma[f_\ell, f_{\ell+1}, \ldots, f_{\ell+t}]$ where γ is a non-padded constraint, and $t < \mathsf{len}(\gamma)$. Namely, indices of the layer variables of a padded constraint must be successive. For such a constraint, we call ℓ and $\ell + t$ the *starting index* and the *ending index*, respectively.
- In a same (Modal)-node, all padded constraints share a same ending index, but their staring indices may be different. Thus, for a (Modal)-node, if the (common) ending index is w, then each starting index must be less than $w - k$. Since we are now concerned about the number of equivalent classes of $\cong$, according to Lemma 1, we may fix the ending index to be k, therefore starting indices belong to the set $\{1, 2, \ldots, k\}$.
- Call two padded constraints to be *homologous* if they are obtained from a same original constraint via applying different layer variable list. Note that homologous is also an equivalent relation, and each equivalent class must be of the form $\{\gamma[f_\ell, f_{\ell+1}, \ldots, f_k] \mid \ell \leq 1\}$ for each $\gamma \in \mathsf{Cons}(\boldsymbol{\Sigma}) \cap \mathsf{Sub}(\varphi)$, denoted that set as $H(\gamma)$.

Let us now count the upper bound of the equivalence class number of (Modal)-nodes. In a (Modal)-node, we categorize the formulas into two sets: the first consists of constraints, and the second one is constituted with X- and/or $\overline{\mathsf{X}}$-guarded formulas.

(1) For each original constraint γ, the first set may contain a subset of $H(\gamma) \cup \{\gamma\}$, hence this part has no more than $2^{(k+1)c}$ possibles.
(2) In a (Modal)-node, each X-guarded (resp. $\overline{\mathsf{X}}$-guarded) formula corresponds a subformula of φ, whose out-most operator is either X (resp. $\overline{\mathsf{X}}$) or U (resp. R). Thus, the number of such formulas occurring in the node is not more than p, and such part yields not more than 2^p subsets.

As a result, once the counter becomes $2^{(k+1)c+p} + 1$, we may declare that some isomorphic (Model)-node already exists in the current path, hence it could be a candidate value of threshold.