Journal of Machine Learning Research () 1-00 Submitted 9/26; Published

Complete Identification of Deep ReLU Networks through Łukasiewicz Logic

Yani Zhang YANIZHANG@MINS.EE.ETHZ.CH
Chair for Mathematical Information Science
ETH Zurich
Sternwartstr. 7, 8092 Zürich, Switzerland

Helmut Bölcskei HBOELCSKEI@ETHZ.CH
Chair for Mathematical Information Science
ETH Zurich
Sternwartstr. 7, 8092 Zürich, Switzerland

Editor:

Abstract

Two deep ReLU networks can have entirely different architectures and parameters, yet realize the same function. We provide a complete characterization of this nonuniqueness, by building a symbolic calculus for deep ReLU networks, equivalence and simplification of networks becoming derivation of formulae, in close parallel to Shannon's analysis of switching circuits through Boolean logic. Two non-degenerate ReLU networks realize the same function on the unit cube if and only if one is obtained from the other by finitely many applications of the axioms of many-valued (MV) logic for integer weights and biases, of divisible MV logic for rational ones, and of Riesz MV logic for real ones. These axioms characterize all symmetries of ReLU networks, single-layer ones, the only kind for tanh networks, and deep ones spanning three or more layers. Our framework consists of three steps, extraction of a substitution graph whose represented formula has the network's input–output map as its truth function, a completeness theorem making functionally equivalent formulae interderivable, and construction returning from graphs to networks. The substitution graph encodes the network uniquely and induces a new normal form for MV logic, *compositional* rather than flat, with three local operations realizing every derivation.

Keywords: neural network identifiability, deep ReLU networks, Łukasiewicz logic, MV algebras, normal forms

1 Introduction

1.1 Background and problem formulation

A deep neural network is specified by its architecture and weights, but the function it computes depends on these quantities only through a many-to-one map. Two networks with different weights, or even different architectures, can realize the same input–output function—and for the ReLU activation $\rho\colon x \mapsto \max\{0, x\}$, such coincidences are pervasive. This redundancy has direct consequences for the theory of deep learning. It determines when two learned models are genuinely distinct, governs the geometry of loss landscapes, and constrains what can be inferred about a neural network from its input–output function alone.

Characterizing this redundancy amounts to identifying all symmetries, i.e., transformations of network representations that preserve the realized input–output function. Vlačić and Bölcskei [21] settled this question for deep networks with tanh activation, where all symmetries are single-layer (acting within a single pair of adjacent layers). For ReLU networks, deep symmetries exist that span at least three layers, and the corresponding question has remained open; the present paper resolves it.

More precisely, given a class $\mathfrak{N}$ of ReLU networks and a domain $X \subset \mathbb{R}^n$, we call a set of symmetries $\mathcal{P}$ *complete for the identification of* $\mathfrak{N}$ *over* X if

$$\mathcal{N}_1 \sim_X \mathcal{N}_2 \implies \mathcal{N}_1 \overset{\mathcal{P}}{\sim} \mathcal{N}_2$$

for all $\mathcal{N}_1, \mathcal{N}_2 \in \mathfrak{N}$, where $\mathcal{N}_1 \sim_X \mathcal{N}_2$ means $\langle \mathcal{N}_1 \rangle(x) = \langle \mathcal{N}_2 \rangle(x)$ for all $x \in X$, with $\langle \mathcal{N} \rangle$ denoting the input–output function realized by $\mathcal{N}$, and $\mathcal{N}_1 \overset{\mathcal{P}}{\sim} \mathcal{N}_2$ designates derivability in finitely many steps via modifications induced by $\mathcal{P}$. Since every modification induced by $\mathcal{P}$ preserves the realized function, the reverse implication holds as well, and completeness yields

$$\mathcal{N}_1 \sim_X \mathcal{N}_2 \iff \mathcal{N}_1 \overset{\mathcal{P}}{\sim} \mathcal{N}_2,$$

i.e., functional equivalence on X coincides with derivability via $\mathcal{P}$. In this case, network representations are identifiable up to $\mathcal{P}$ from the input–output map.

Three types of symmetry have been documented in the literature [21, 23]. *Affine symmetries* [21], defined for a general activation function φ, are induced by identities of the form

$$\sum_{s \in I} \alpha_s \varphi(\beta_s x + \gamma_s) = \xi,$$

where $\alpha_s \neq 0$ and $\beta_s \neq 0$ for all $s \in I$, and $\xi \in \mathbb{R}$. Instantiated locally, such identities can replace one node by several or, read in reverse, merge several into one, changing the number of nodes while preserving the realized function. *Scaling symmetries* [15, 18], a special case of affine symmetries, arise from positive homogeneity. For ReLU and any positively homogeneous activation, the identity

$$\rho(x) = \frac{1}{\lambda} \rho(\lambda x), \qquad \lambda > 0,$$

effects a rescaling of the incoming weights and bias of a node by λ and its outgoing weights by $1/\lambda$; the weights change, but both the network architecture and the realized function are preserved. *Permutation symmetries*, which apply to any activation function, reorder neurons within a hidden layer and apply the same permutation to the corresponding incoming and outgoing weights; they likewise preserve both the network architecture and the realized function. Permutation symmetries are subsumed within affine symmetries, as permuting neurons merely reorders the terms in the affine symmetry sum. Affine symmetries are single-layer symmetries in the sense of acting within a single pair of adjacent layers, leaving all other weights unchanged [21, 23]. In particular, single-layer modifications preserve the number of hidden layers.

It was shown in [21] that for tanh networks of any architecture, all affine symmetries are generated by the single identity $\tanh(t) = -\tanh(-t)$ and constitute the only source of

redundancy, yielding complete identification over $\mathbb{R}^n$. The complete set of symmetries of tanh networks therefore consists exclusively of single-layer modifications.

For ReLU networks, however, deep symmetries exist—modifications not confined to a single pair of adjacent layers and that cannot be generated by any composition of single-layer modifications [15, 21].

Proposition 1 (cf. [21, Sec. II-B])**.** *For $n, m \in \mathbb{N}$, let $\mathfrak{N}$ be the class of ReLU networks with n input and m output nodes. The set of affine symmetries is not complete for the identification of $\mathfrak{N}$ over $\mathbb{R}^n$.*

Complete identification therefore requires going beyond single-layer modifications, as deep symmetries can span many layers and involve large subnetworks. Since deep symmetries change the structure of the network across several layers when applied, their action must be described through rewriting on objects capable of expressing such cross-layer structural changes—the substitution graphs introduced in Section 3. The present paper shows that such symmetries are exhaustively described through a fundamental correspondence between ReLU networks and many-valued (MV) logic [8], a logical calculus that generalizes Boolean logic from the two-element truth domain $\{0, 1\}$ to the continuous interval $[0, 1]$. Specifically, ReLU networks with integer weights and biases correspond to Łukasiewicz formulae [8] via the identification of the function realized by the network on $[0, 1]^n$ with the *truth function* of the formula—the function $[0, 1]^n \to [0, 1]$ obtained by evaluating the formula under the MV operations (defined in Section 1.3)—a correspondence that extends to rational and real weights and biases[1]—and Chang's completeness theorem guarantees that any two such formulae with the same truth function are related by rewriting via the MV axioms (the equational axioms of MV algebras). This rewriting is carried out on the substitution graphs representing the networks (Section 4). The MV axioms thereby generate all symmetries of ReLU networks—affine (single-layer) symmetries in the sense of [21] and, additionally, deep symmetries that affine symmetries cannot reach—and two networks are functionally equivalent if and only if their MV formulae are related by such rewriting.

1.2 Why affine symmetries are insufficient

We start by formally defining ReLU networks.

Definition 1 (ReLU network)**.** A *ReLU network* $\mathcal{N}$ of depth $L \in \mathbb{N}$ and architecture $(d_0, \ldots, d_L) \in \mathbb{N}^{L+1}$, $d_L = 1$, is a tuple $(A_1, \ldots, A_L)$ of affine maps $A_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$, $A_\ell(x) = W_\ell x + b_\ell$, $\ell = 1, \ldots, L$, with weight matrices $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ and bias vectors $b_\ell \in \mathbb{R}^{d_\ell}$; it *realizes* the continuous, piecewise linear function

$$\langle \mathcal{N} \rangle = A_L \circ \rho \circ A_{L-1} \circ \rho \circ \cdots \circ \rho \circ A_1 \ : \ \mathbb{R}^{d_0} \to \mathbb{R},$$

where $\rho(x) = \max\{x, 0\}$ is the ReLU nonlinearity applied component-wise. For $L = 1$, $\langle \mathcal{N} \rangle = A_1$ is a purely affine map.

1. For rational weights and biases the relevant algebraic structure is the divisible MV (DMV) algebra, which extends the MV axioms by division operators; the analogous completeness theorem—same truth function implies interderivability—holds in this extended system [13]. For real weights and biases the appropriate framework is that of Riesz MV algebras, which extend the MV axioms by scalar multiplication operators; the analogous completeness theorem holds in this extended system [11].

The following example illustrates the effect of affine symmetries on a concrete ReLU network. Consider the network $\mathcal{N}$ realizing

$$\langle \mathcal{N} \rangle(x) = -\rho(\rho(-x) + \rho(2x)) + \rho(\rho(-x) + 1). \tag{1}$$

Figure 1 illustrates how $\mathcal{N}$ can be modified by affine symmetries—a permutation, a scaling, and a proper affine symmetry—to yield the functionally equivalent networks $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$, respectively. The affine maps in the decomposition $\langle \mathcal{N} \rangle = A_3 \circ \rho \circ A_2 \circ \rho \circ A_1$ are

$$A_1(x) = \begin{pmatrix} -1 \\ 2 \end{pmatrix} x, \qquad A_2(z) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} z + \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad A_3(z) = \begin{pmatrix} -1 & 1 \end{pmatrix} z, \tag{2}$$

where $x \in \mathbb{R}$ and $z \in \mathbb{R}^2$. $\mathcal{N}_1$ replaces (A_1, A_2) by (PA_1, A_2P), with P the 2×2 swap permutation matrix; $\mathcal{N}_2$ replaces (A_1, A_2) by $(D_{1/2}A_1, A_2D_{1/2}^{-1})$ with $D_{1/2} = \mathrm{diag}(1, \frac{1}{2})$; $\mathcal{N}_3$ replaces the node computing $\rho(2x)$ in the first hidden layer of $\mathcal{N}$ using $\rho(2x) = \rho(2x-1) - \rho(-2x+1) + \rho(-2x) + 1$, a consequence of the affine symmetry $\rho(2x-1) - \rho(-2x+1) - \rho(2x) + \rho(-2x) = -1$, expanding it into three nodes. The functional equivalences $\langle \mathcal{N} \rangle = \langle \mathcal{N}_1 \rangle = \langle \mathcal{N}_2 \rangle = \langle \mathcal{N}_3 \rangle$ are all accounted for by affine symmetries; the next example shows that affine symmetries are not complete for identification. Consider the network $\mathcal{N}'$ in Figure 2 realizing $\langle \mathcal{N}' \rangle(x) = \rho(1 - \rho(1-x))$, and the network $\mathcal{N}''$ realizing $\langle \mathcal{N}'' \rangle(x) = \rho(x) - \rho(x-1)$. These two networks are functionally equivalent: the identity $1 - \rho(t) = \min\{1, 1-t\}$, applied with $t = 1 - x$, yields

$$\rho\big(1 - \rho(1-x)\big) = \max\big\{0, \min\{x, 1\}\big\} = \rho(x) - \rho(x-1), \quad x \in \mathbb{R}.$$

Since affine symmetries are single-layer and thus preserve the number of hidden layers, and $\mathcal{N}'$ has two hidden layers while $\mathcal{N}''$ has one, affine symmetries cannot connect them, thereby proving Proposition 1; the equivalence $\langle \mathcal{N}' \rangle = \langle \mathcal{N}'' \rangle$ requires a symmetry that spans more than a single pair of adjacent layers; its precise character within the taxonomy of network symmetries is identified in Section 1.3.

1.3 Complete identification via many-valued logic

The central idea of this paper is to characterize the nonuniqueness of ReLU networks through Łukasiewicz infinite-valued logic—henceforth Łukasiewicz logic. With truth values in the real interval $[0, 1]$, it generalizes Boolean logic from the two-element domain $\{0, 1\}$ to the continuum. The input-output maps of ReLU networks with integer weights and biases are expressed as Łukasiewicz formulae, and syntactic manipulation of those formulae, guided by the logic axioms, relates the networks realizing the same map; Chang's completeness theorem then guarantees that this manipulation is exhaustive. We first review the relevant concepts.

Definition 2. A *Łukasiewicz formula* is a finite string formed from propositional variables $x_1, x_2, \ldots$ and the constants $0, 1$ using the connectives $\oplus$ (disjunction), $\odot$ (conjunction), and $\neg$ (negation), generalizing their Boolean counterparts to $[0, 1]$.

For instance, $\neg x_1 \oplus (x_2 \odot \neg x_1)$ is a Łukasiewicz formula with variables x_1 and x_2.

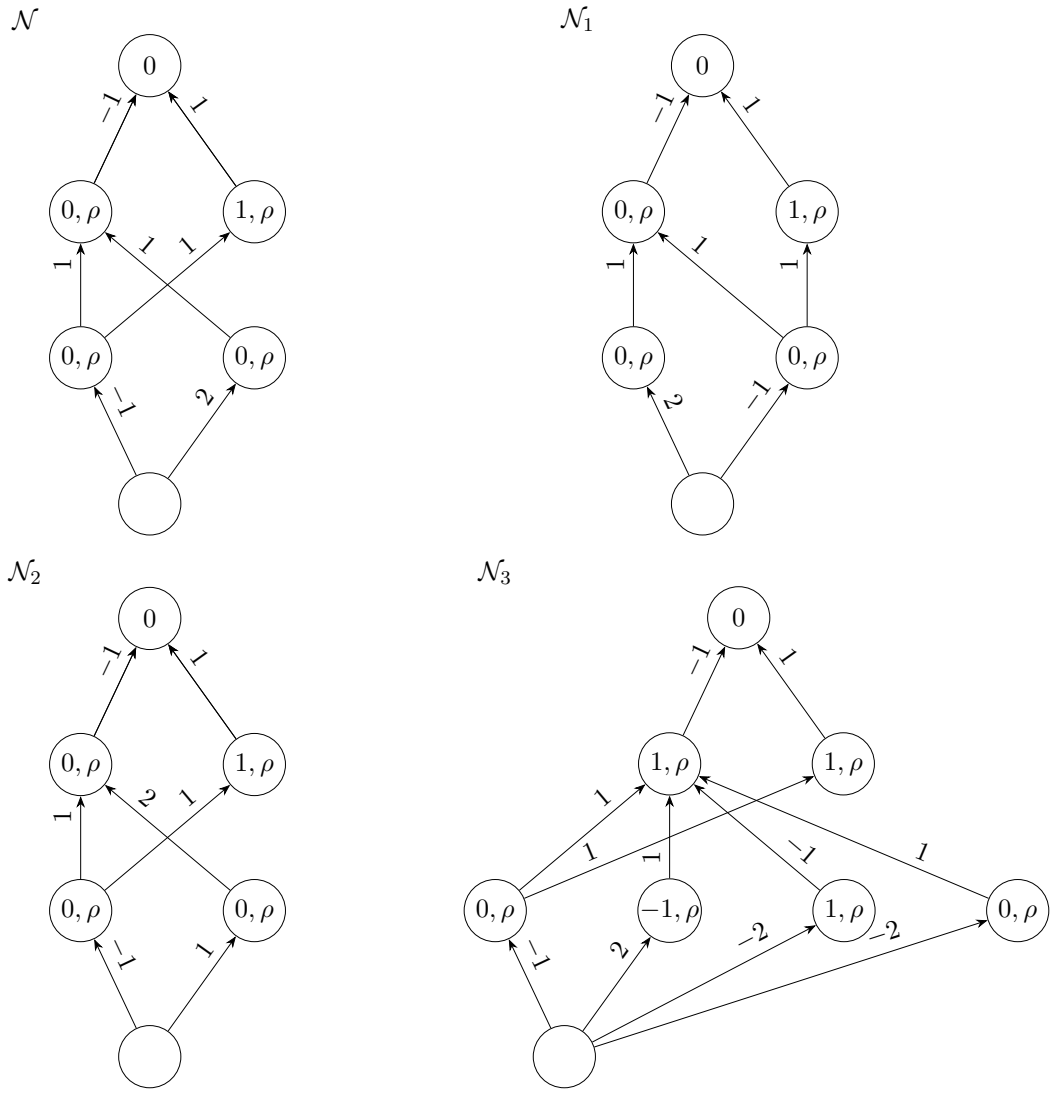

Figure 1: The network $\mathcal{N}$ and three functionally equivalent networks $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$, derived by affine modifications: a permutation, a scaling, and a proper affine symmetry, respectively. Edge labels denote weights; node labels denote biases and activation functions. Zero-weight edges are omitted.

Definition 3 (Chang [6])**.** The standard Łukasiewicz algebra $\mathbb{I} = ([0,1], \oplus, \odot, \neg, 0, 1)$ is defined by $x \oplus y = \min\{1, x + y\}$, $x \odot y = \max\{0, x + y - 1\}$, $\neg x = 1 - x$. When restricted to $\{0, 1\}$, these operations reduce to Boolean disjunction, conjunction, and negation, respectively.

The *truth function* of a Łukasiewicz formula τ in variables $x_1, \ldots, x_n$ is the function $\tau^{\mathbb{I}} : [0,1]^n \to [0,1]$ obtained by evaluating τ under the operations of $\mathbb{I}$—the continuous-domain analogue of a Boolean truth table.

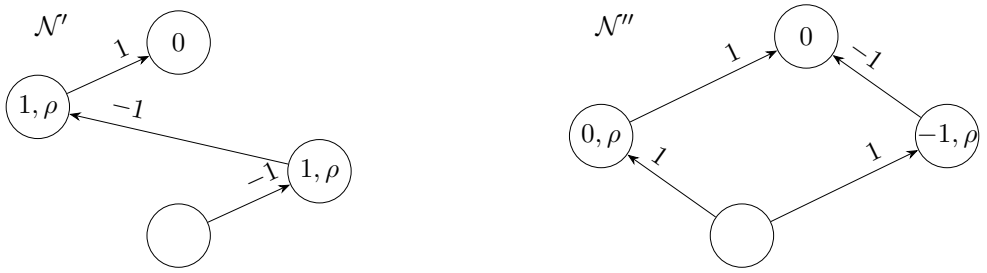

Figure 2: Functionally equivalent networks $\mathcal{N}'$ and $\mathcal{N}''$ that cannot be connected by affine symmetries.

McNaughton's theorem characterizes the truth functions of Łukasiewicz formulae.

Theorem 1 (McNaughton [16])**.** A function $f\colon [0,1]^n \to [0,1]$ is the truth function of some Łukasiewicz formula if and only if f is continuous and piecewise linear with integer coefficients, i.e., there exist $\ell \in \mathbb{N}$ and linear polynomials $p_1, \ldots, p_\ell$ of the form $p_j(x) = m_{j1}x_1 + \cdots + m_{jn}x_n + b_j$, $m_{ji}, b_j \in \mathbb{Z}$, such that $f(x) = p_j(x)$ for some $j \in \{1, \ldots, \ell\}$ at every $x \in [0,1]^n$. Functions of this form are called *McNaughton functions*.

The Boolean analogue asserts that every $f\colon \{0,1\}^n \to \{0,1\}$ is the truth function of a Boolean formula; on the Łukasiewicz domain $[0,1]^n$, by contrast, only the continuous piecewise linear functions with integer coefficients are truth functions (McNaughton's theorem). It was established in [22] that the $[0,1]$-valued input-output maps of ReLU networks on $[0,1]^n$ with integer weights and biases are precisely the McNaughton functions. Conversely, every McNaughton function arises as the input-output map of a ReLU network with integer weights and biases, via a construction algorithm [22] that maps any Łukasiewicz formula to such a network whose input-output map equals the formula's truth function on $[0,1]^n$. Furthermore, for every ReLU network $\mathcal{N}$ with integer weights and biases realizing a $[0,1]$-valued function and satisfying a mild non-degeneracy condition, an extraction algorithm [22] produces a Łukasiewicz formula whose truth function equals $\langle \mathcal{N} \rangle$ on $[0,1]^n$. This correspondence is the starting point of the identification approach developed in this paper, which represents the input-output map of each network as a Łukasiewicz formula via extraction, manipulates the formula syntactically using the axioms of MV algebra, and recovers an equivalent network via construction.

The operations of $\mathbb{I}$ satisfy the following axioms, which define the class of MV algebras.

Definition 4. [6] A *many-valued (MV) algebra* is a structure $\mathbb{M} = (M, \oplus, \odot, \neg, 0, 1)$, with M a non-empty set, $\oplus$ and $\odot$ binary operations on M, $\neg$ a unary operation on M, and $0, 1 \in M$ distinguished constants, satisfying the following axioms:

Ax. 1. $x \oplus y = y \oplus x$	Ax. 1′. $x \odot y = y \odot x$
Ax. 2. $x \oplus (y \oplus z) = (x \oplus y) \oplus z$	Ax. 2′. $x \odot (y \odot z) = (x \odot y) \odot z$
Ax. 3. $x \oplus \neg x = 1$	Ax. 3′. $x \odot \neg x = 0$
Ax. 4. $x \oplus 1 = 1$	Ax. 4′. $x \odot 0 = 0$
Ax. 5. $x \oplus 0 = x$	Ax. 5′. $x \odot 1 = x$
Ax. 6. $\neg(x \oplus y) = \neg x \odot \neg y$	Ax. 6′. $\neg(x \odot y) = \neg x \oplus \neg y$
Ax. 7. $x = \neg(\neg x)$	Ax. 8. $\neg 0 = 1$
Ax. 9. $(x \odot \neg y) \oplus y = (y \odot \neg x) \oplus x$	Ax. 9′. $(x \oplus \neg y) \odot y = (y \oplus \neg x) \odot x$

Boolean algebras are the MV algebras additionally satisfying $x \oplus x = x$ for all x [8, Corollary 1.5.5]; in $\mathbb{I}$ this becomes $\min\{1, 2x\} = x$, which forces $x \in \{0, 1\}$, recovering classical Boolean logic as a special case of MV logic.

The MV axioms, instantiated in $\mathbb{I}$ and expressed in terms of ρ, give rise to three tiers of symmetries; the DMV and Riesz extensions of Section 6 add a fourth tier of scaling symmetries. A complete listing of all four tiers can be found in Appendix A. *Tier 1 — Degenerate symmetries* (Ax. 3, 3′, 4, 4′, 5, 5′, 6, 6′, 7, 8): these hold by direct arithmetic—the argument of every ρ either cancels to a constant or lies permanently in the linear or clipped-negative regime, so no clipping transition occurs—and correspond to trivial structural network simplifications (replacing subnetworks with constant output by a single node realizing that constant value, or eliminating identity operations) rather than nontrivial rearrangements as in Tiers 2 and 3 below; Ax. 6 and 6′ are the sole exceptions, their ρ-arguments crossing zero; here each axiom's two sides expand to the same ρ-expression, so its two syntactically different formulae realize the same ReLU network. *Tier 2 — Permutation symmetries* (Ax. 1, 1′): each axiom reorders neurons within a hidden layer and applies the same permutation to the corresponding incoming and outgoing weights. *Tier 3 — Deep symmetries* (Ax. 2, 2′, 9, 9′): the primitive axioms in this tier span at least three network layers, with at least one inner ρ clipping on $[0, 1]^n$; further genuine deep symmetries, spanning more than three layers, can be composed from these and Tier 1 identities. *Pseudo-deep symmetries* (derived): identities that are syntactically multi-layer but in which no inner ρ clips on $[0, 1]^n$, making the nestings redundant. They arise from Tier 1 identities. The equivalence $\langle \mathcal{N}' \rangle(x) = \rho(1 - \rho(1 - x)) = \rho(x) - \rho(x - 1) = \langle \mathcal{N}'' \rangle(x)$ from Section 1.2 is of this type: the inner $\rho(1 - x) = 1 - x$ for all $x \in [0, 1]$ (since $1 - x \geq 0$, so no clipping occurs), reducing the identity to a combination of Tier 1 identities (Ax. 4′, 5′, and 7). The two-hidden-layer structure of $\mathcal{N}'$ is thus operationally void. Pseudo-deep symmetries can of course be composed with symmetries from any other tier—genuine deep, permutation, or scaling—producing compound transformations that combine redundant nestings with additional structural changes. Identifying the pseudo-deep components in such a compound allows reducing a syntactically multi-layer subnetwork to a shallower one, yielding a simpler equivalent network. *Tier 4 — Scaling symmetries* (Ax. DMV-n, Riesz-r): these arise only in the DMV and Riesz extensions (Section 6), each involving a single ρ on each side with no nesting and hence of the same nature as the Tier 2 permutation symmetries—both are affine symmetries in the sense of [21]. The local action differs though: whereas Tier 2 reorders neurons within a layer, Tier 4 rescales the weights and biases of a single node.

The complete identification program proceeds in three steps (Theorem 3):

(i) *Extraction*: For every non-degenerate ReLU network $\mathcal{N} \in \mathfrak{N}$ (Definition 14), an extraction algorithm (detailed in Sections 2 and 3) produces a substitution graph $\texttt{ext}(\mathcal{N})$. This graph inherits the layered structure of the network, each of its nodes carrying the Łukasiewicz formula the corresponding node computes. Layer-wise syntactic substitution of the node labels yields the Łukasiewicz formula $\tau = [\texttt{ext}(\mathcal{N})]$, satisfying $\tau^{\mathbb{I}} = \langle \mathcal{N} \rangle$ on $[0,1]^n$.

(ii) *Derivation*: Chang's completeness theorem (Theorem 2) guarantees that any two Łukasiewicz formulae with the same truth function are derivable from each other by application of MV axioms (in finitely many steps). Lemma 5 lifts this formula-level derivability to substitution graphs, and, by Proposition 4, a graph-level derivation is realized by a finite sequence of local operations—node rewrites (Definition 23), which change a single node formula, and collapses and expansions of layers (Definition 24), which change the graph structure. The network-level relation $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$ is defined to mean $\texttt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \texttt{ext}(\mathcal{N}_2)$ (Definition 22)—a pullback through extraction, necessary because the intermediate graphs of a derivation need not correspond to networks (see the discussion following Theorem 3).

(iii) *Construction*: The construction algorithm in Section 5, applied to any normal (Definition 15) substitution graph $\mathcal{G}$, returns a ReLU network $\texttt{constr}(\mathcal{G})$ with $\langle \texttt{constr}(\mathcal{G}) \rangle = [\mathcal{G}]^{\mathbb{I}}$; moreover, $\texttt{constr}$ is a left inverse of $\texttt{ext}$ on $\mathfrak{N}$, that is, $\texttt{constr}(\texttt{ext}(\mathcal{N})) = \mathcal{N}$ for all $\mathcal{N} \in \mathfrak{N}$ (Proposition 5).

These three steps establish a symbolic calculus for deep ReLU networks. A network is manipulated by rewriting the formula extracted from it, and two networks realize the same function precisely when their formulae are interderivable by application of the MV axioms, so that questions about networks, equivalence and simplification among them, become questions about formulae. We develop the required machinery, namely substitution graphs and their compositional normal form, the three local operations that realize every derivation, and the lifting of derivability from formulae to graphs and networks. This machinery applies unchanged for finite input domains and for rational and real weights and biases, as shown in Section 6.

This three-step structure has an antecedent in Shannon's approach to switching circuit design [20]. At the heart of Shannon's theory is a bidirectional correspondence between switching circuits and Boolean formulae: every switching circuit can be associated with a Boolean formula representing its functionality, and conversely, every Boolean formula determines a circuit implementing the underlying function. Two circuits are functionally equivalent—they compute the same Boolean function—if and only if their associated formulae are interderivable (in finitely many steps) by application of the axioms of Boolean algebra [20]; circuit equivalence testing and simplification can therefore be carried out purely algebraically.

A concrete example illustrates the approach. The switching circuit in Figure 3 has associated formula $x_3 \odot ((x_2 \odot \neg x_1) \oplus (\neg x_1 \odot x_2))$; applying Boolean axioms simplifies it:

$$\begin{aligned} x_3 \odot ((x_2 \odot \neg x_1) \oplus (\neg x_1 \odot x_2)) &= x_3 \odot ((x_2 \odot \neg x_1) \oplus (x_2 \odot \neg x_1)) \\ &= x_3 \odot (x_2 \odot \neg x_1). \end{aligned}$$

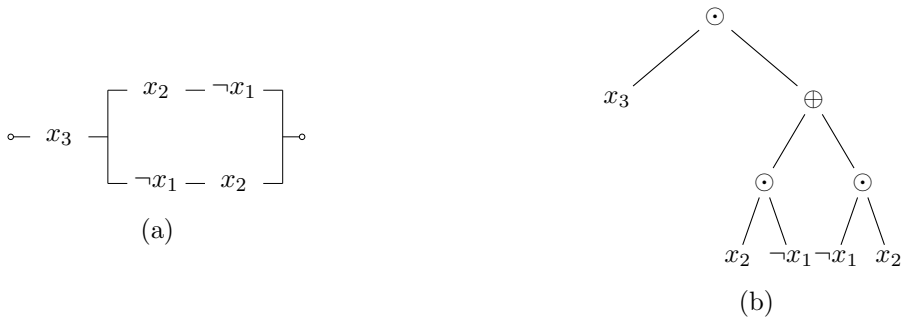

Figure 3: The Boolean formula $x_3 \odot ((x_2 \odot \neg x_1) \oplus (\neg x_1 \odot x_2))$. *(a)* Corresponding circuit. The symbol x_i denotes a make contact (closed when $x_i = 1$) and $\neg x_i$ a break contact (open when $x_i = 1$); series connections correspond to $\odot$ and parallel branches to $\oplus$. *(b)* Parse tree of the formula. Node labels are logical connectives ($\odot$, $\oplus$) or contacts (x_i, $\neg x_i$); the tree structure records the hierarchical nesting of series and parallel connections.

Figure 4: The circuit corresponding to the simplified formula $x_3 \odot (x_2 \odot \neg x_1)$, equivalent to Figure 3a; notation as in Figure 3.

The simplified formula $x_3 \odot (x_2 \odot \neg x_1)$ corresponds to the simpler circuit in Figure 4.

For Shannon's series-parallel circuits, steps (i) and (iii) present no difficulty (Shannon shows that non-series-parallel circuits such as the bridge circuit can be converted to equivalent series-parallel form [20], possibly at an increase in circuit size): the parse tree of a Boolean formula—the rooted tree recording the formula's recursive assembly from its parts—carries the same structural information as the series-parallel circuit diagram (Figure 3). The formula is read off the circuit by reading its series-parallel structure as a parse tree; the circuit is drawn from the formula's parse tree by interpreting each node as a series or parallel connection and each leaf as a relay contact. Construction from the parse tree recovers the circuit exactly, since the correspondence between parse trees and series-parallel circuits is bijective.

In the Boolean setting, step (ii) is elementary: for Boolean functions in n variables, the truth domain $\{0,1\}^n$ has exactly 2^n points. For each input $(a_1, \ldots, a_n) \in \{0,1\}^n$ at which the function equals 1, write the conjunction $x_1^{a_1} \odot \cdots \odot x_n^{a_n}$ where $x_i^1 = x_i$ and $x_i^0 = \neg x_i$; the disjunction $\oplus$ of these terms over all such inputs yields a canonical disjunctive normal form (DNF) [20]—uniquely determined by the function's truth table, with at most 2^n terms—and two functionally equivalent formulae share the same canonical DNF, making them interderivable by routing through it: any formula can be derived to its canonical DNF by finitely many applications of the Boolean axioms [20], and conversely.

Figure 5 illustrates the structural correspondence between Shannon's approach and the MV-logic approach to ReLU networks: extraction translates a circuit or network into

Figure 5: Structural correspondence between Shannon's switching circuit framework and the MV-logic framework for ReLU networks.

a formula or substitution graph, respectively; derivation relates equivalent formulae or substitution graphs; and construction translates back.

The present work is the MV-logic analogue of Shannon's program, as illustrated in Figure 5: the three-step structure is identical, and the philosophy is the same; what changes is the difficulty of each step.

In the Boolean setting, step (ii) relied on the finiteness of the truth domain; in the MV setting, the truth domain is the continuum $[0, 1]$. Two Łukasiewicz formulae are functionally equivalent when they agree at every point of the uncountable set $[0, 1]^n$, so the enumeration argument that produces the canonical DNF fails—there is no finite list of satisfying assignments to read off. Establishing that functionally equivalent Łukasiewicz formulae are interderivable (in finitely many steps) by application of MV axioms is a non-trivial algebraic fact, supplied by Chang's theorem.

Steps (i) and (iii) likewise require non-trivial extensions. A ReLU network is not a series-parallel circuit but a computational graph, so the direct tree-traversal approach of the Boolean case—where parse trees and series-parallel circuits are in bijection—is no longer available; extraction must instead retain the full computational-graph topology of the network, and construction must exploit that topology to recover the network architecture exactly. Substitution graphs play the role of parse trees in the MV setting: they record the computational-graph topology of the network and serve as input to the construction algorithm.

Chang's theorem is the cornerstone of step (ii).

Theorem 2 (Chang [6, 7])**.** Let τ_1 and τ_2 be Łukasiewicz formulae. If $\tau_1^{\mathbb{I}} = \tau_2^{\mathbb{I}}$, then $\tau_1 \overset{\mathcal{MV}}{\sim} \tau_2$.

Here $\tau_1 \overset{\mathcal{MV}}{\sim} \tau_2$ denotes derivability of τ_2 from τ_1 in finitely many steps (Definition 20), each an application of an MV axiom. The converse implication is immediate as the application of MV axioms, by definition, leaves the truth function unchanged. For formulae extracted from ReLU networks, $\tau_1 = [\mathtt{ext}(\mathcal{N}_1)]$ and $\tau_2 = [\mathtt{ext}(\mathcal{N}_2)]$ with $\mathcal{N}_1, \mathcal{N}_2 \in \mathfrak{N}$, this is, by Definition 21, exactly $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$. How this derivability is reflected at the level of networks depends on the axioms applied in the rewrite. Each induced symmetry (Appendix A) maps a network directly to a functionally equivalent network. The symmetries of Tiers 2 and 4 in their entirety, together with part of Tier 1, are affine (single-layer) symmetries in the sense of [21]; the remaining Tier-1 symmetries, induced by identities outside the affine form

of Section 1.1, are single-layer as well—their structural effect, such as the removal of a constant subnetwork, decomposes into single-layer steps (Appendix A). A sequence of such symmetries yields a chain of networks. The deep symmetries of Tier 3, in contrast, are neither affine nor single-layer—they rearrange structure across at least three layers and cannot be generated by compositions of single-layer modifications (Section 1.1). Single-layer symmetries also alter the network's computational graph, each application of an axiom inserting, merging, or removing nodes within a single layer. In contrast, a deep symmetry couples several layers at once, so that describing its action requires rewriting on objects capable of expressing such cross-layer structural change—the substitution graphs (Section 4). Normal substitution graphs (Definition 15) are the graphs that the construction algorithm maps to networks (Section 5); rewriting involving deep symmetries, however, passes through intermediate graphs that are not normal, as will be shown in Section 4. We accordingly define the network-level relation $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$ as a pullback through extraction, that is, as $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$ (Definition 22). Individual derivation steps thus carry graph-level meaning only—the direct network action of an induced symmetry and the corresponding derivation step on graphs are distinct operations, and it is the latter that derivations compose; network-level statements are made at the endpoints, where the graphs are extracted from networks.

The main result of the present paper is the following.

Theorem 3. For $n \in \mathbb{N}$, let $\mathfrak{N}$ be the class of non-degenerate (Definition 14) ReLU networks with integer weights and biases realizing functions $f : [0,1]^n \to [0,1]$. For all $\mathcal{N}_1, \mathcal{N}_2 \in \mathfrak{N}$,

$$\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2 \implies \mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2.$$

Proof Step (i) (extraction): set $\tau_i = [\mathtt{ext}(\mathcal{N}_i)]$, $i = 1, 2$. Since $\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$, $\tau_1^{\mathbb{I}} = \langle \mathcal{N}_1 \rangle = \langle \mathcal{N}_2 \rangle = \tau_2^{\mathbb{I}}$. Step (ii) (derivation): since $\tau_1^{\mathbb{I}} = \tau_2^{\mathbb{I}}$, Lemma 5 yields $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$, which is $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$ by Definition 22. ■

The reverse implication $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2 \implies \mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$ is verified as follows. $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$ means $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$ (Definition 22), which by Definition 21 is $[\mathtt{ext}(\mathcal{N}_1)] \overset{\mathcal{MV}}{\sim} [\mathtt{ext}(\mathcal{N}_2)]$. Definition 20 then provides a finite sequence of formulae $[\mathtt{ext}(\mathcal{N}_1)] = \varphi_0, \varphi_1, \ldots, \varphi_T = [\mathtt{ext}(\mathcal{N}_2)]$, each obtained from its predecessor by applying an MV axiom. Each such application preserves the truth function, $\varphi_{t-1}^{\mathbb{I}} = \varphi_t^{\mathbb{I}}$; hence $[\mathtt{ext}(\mathcal{N}_1)]^{\mathbb{I}} = [\mathtt{ext}(\mathcal{N}_2)]^{\mathbb{I}}$. Since extraction preserves the realized function (Section 2), $\langle \mathcal{N}_1 \rangle = [\mathtt{ext}(\mathcal{N}_1)]^{\mathbb{I}}$ and $\langle \mathcal{N}_2 \rangle = [\mathtt{ext}(\mathcal{N}_2)]^{\mathbb{I}}$ on $[0,1]^n$, and therefore $\langle \mathcal{N}_1 \rangle = \langle \mathcal{N}_2 \rangle$, i.e., $\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$. Together, Theorem 3 and its converse give

$$\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2 \iff \mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2.$$

Section 6 carries this program to finite input domains, to rational weights and biases, and to real weights and biases, the last through the Riesz extension of MV algebra (Theorem 10). Neither the integrality of the weights nor the restriction to the unit box as input domain is therefore essential. For real weights, an affine change of variables normalizes any compact box to $[0,1]^n$ and the realized function to $[0,1]$, and both normalizations are absorbed into the real weights and biases (Section 6.3). Complete identification thus holds for non-degenerate

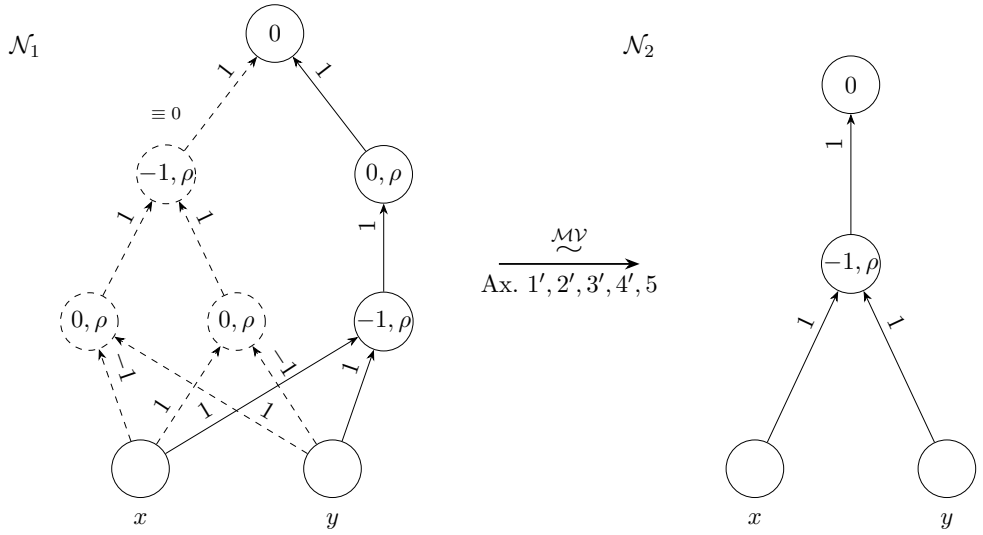

Figure 6: The network $\mathcal{N}_1$ realizes the truth function of $\tau_1 = (x \odot y) \oplus \big((\neg x \odot y) \odot (x \odot \neg y)\big)$ on $[0,1]^2$. Here $\rho(x+y-1) = x \odot y$, $\rho(-x+y) = \neg x \odot y$, and $\rho(x-y) = x \odot \neg y$, while the output node combines its inputs by the disjunction $\oplus$. The dashed branch computes $(\neg x \odot y) \odot (x \odot \neg y) \equiv 0$; the MV derivation rewrites τ_1 to $x \odot y$, establishing $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$, and pruning the dashed branch realizes the result as the single-node network $\mathcal{N}_2$.

ReLU networks with arbitrary real weights and biases, over any compact box. The integer case is treated first because it is the sharpest—the axioms of MV logic alone suffice, no scaling symmetries arise (Appendix A), and the connection to Chang's completeness theorem is most direct.

The following example shows how a network can be pruned by application of MV axioms, without changing its input-output relation. Consider the network $\mathcal{N}_1$ in Figure 6, whose extracted formula becomes, after Tier-1 and Tier-2 rewrites, $\tau_1 = (x \odot y) \oplus \big((\neg x \odot y) \odot (x \odot \neg y)\big)$. Applying the MV axioms, we obtain

$$\begin{aligned}\tau_1 = (x \odot y) \oplus \big((\neg x \odot y) \odot (x \odot \neg y)\big) &\overset{\text{Ax. } 1',2'}{=} (x \odot y) \oplus \big((x \odot \neg x) \odot (y \odot \neg y)\big) \\ &\overset{\text{Ax. } 3'}{=} (x \odot y) \oplus (0 \odot 0) \overset{\text{Ax. } 4'}{=} (x \odot y) \oplus 0 \overset{\text{Ax. } 5}{=} x \odot y.\end{aligned}$$

Pruning the dead branch leaves the single-node network $\mathcal{N}_2$ of Figure 6; the derivation rewrites τ_1 into $x \odot y$ by application of MV axioms, establishing $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$.

Rewriting a structure-carrying object (here a substitution graph) by the axioms of an algebra (the MV algebra), under a semantics that forgets the structure (the truth function of the represented formula), and proving a completeness theorem that matches derivability to equality of what the objects denote (the functions the networks realize), has analogues in other fields. One is shared-subterm graphs in the implementation theory of functional languages, where graph reduction is sound with respect to term rewriting in general and complete for suitably regular rewrite systems [3]; another is string diagrams in categorical

algebra, the morphisms of PROPs—symmetric strict monoidal categories whose objects are the natural numbers—for which a complete equational characterization of equality of the denoted linear relations is known [4]. Two features distinguish the development here from [3] and [4]. The objects being rewritten encode neural networks, and normality singles out exactly those graphs that read back as networks, so that the admissible objects form a proper subclass of those the calculus manipulates, a subclass a derivation may have to leave and re-enter (Example 5); in [3] and [4] every object is admissible, so that a derivation cannot leave that subclass. A third feature sets the development apart, the normal form the graphs are required to be in being *compositional* (Section 3), its building blocks substituted into one another along the layered structure of the network rather than combined by a fixed pattern of connectives (Remark 1); no normal form of this kind appears in the Łukasiewicz literature. Moreover, to the best of our knowledge, the present paper is the first to develop a systematic symbolic calculus for deep ReLU networks.

Paper organization. Section 2 reviews the extraction algorithm of [22], presented here as a node-by-node reading of the network's computational graph (step (i)). Section 3 introduces the compositional normal form—a graphical representation of the network's formula that preserves the layered structure rather than collapsing to a string as in [22] (step (i)). In Section 4, we perform MV derivation on substitution graphs. A derivation starts from and returns to graphs in normal form, the extracted graphs of the two networks; it proceeds by local graph operations—node rewrites and layer collapses and expansions—through intermediate graphs that need not be normal (step (ii)). Section 5 develops the construction algorithm, which converts the substitution graph back to a ReLU network, closing the identification loop (step (iii)). Section 6 extends all results to finite domains, rational weights, and real weights.

2 Extracting formulae from ReLU networks

This section establishes step (i) of the proof of Theorem 3 by first reviewing the extraction algorithm of [22], which consists of three steps. Extraction, Step I converts the ReLU network, or *ρ-network*, to a clipped-ReLU network, or *σ-network*, with $\sigma(x) = \max\{\min\{x, 1\}, 0\}$. Step II associates an MV formula with each σ-node, and Step III composes the node formulae by substitution. The result of this composition is a single Łukasiewicz string. As discussed in Section 1.3, the identification program we establish needs the network's graph structure to be retained. We therefore introduce a representation that retains it, formed by the layered graph of the σ-network with each of its nodes labeled by the Łukasiewicz formula assigned to it by Step II, whose truth function is that node's local map. Throughout Sections 2 to 5, ReLU networks are assumed to have integer weights and biases and to realize functions $[0, 1]^{d_0} \to [0, 1]$; these are the hypotheses under which extraction operates, and they are not restated in the individual definitions and results. Section 6.1 retains integer weights and biases but considers finite input domains; Sections 6.2 and 6.3 replace integer by rational and by real weights and biases, respectively. The requirement that the network realize a $[0, 1]$-valued function on $[0, 1]^{d_0}$ is retained throughout.

2.1 ReLU networks as computational graphs

Formally, a ReLU network is a directed acyclic graph whose edges carry weights and whose nodes carry biases and activations (Definitions 5–8); we call the labeled graph underlying a neural network its *computational graph*.

Definition 5 (Directed acyclic graph)**.**

- A *directed graph* is an ordered pair (V, E) where V is a nonempty finite set of nodes and $E \subset V \times V \setminus \{(v, v) : v \in V\}$ is a nonempty set of directed edges. An edge $(v, \tilde{v})$ points from v to $\tilde{v}$.
- A *directed cycle* is a sequence $v_1, \ldots, v_k, v_1$ with $(v_j, v_{j+1}) \in E$ for $j < k$ and $(v_k, v_1) \in E$.
- A *directed acyclic graph* (DAG) is a directed graph with no directed cycles.

For a DAG (V, E), define the *parent set* of the node v as $\mathrm{par}(v) = \{\tilde{v} \in V : (\tilde{v}, v) \in E\}$, its *children* as $\{\tilde{v} \in V : (v, \tilde{v}) \in E\}$, and the *level* $\mathrm{lv}(v)$ recursively: $\mathrm{lv}(v) = 0$ if $\mathrm{par}(v) = \varnothing$; otherwise $\mathrm{lv}(v) = \max_{u \in \mathrm{par}(v)} \mathrm{lv}(u) + 1$.

Definition 6 (Layered graph)**.** A DAG (V, E) is *layered* if there exists $L \in \mathbb{N}$ such that

- $V = V^{(0)} \cup \cdots \cup V^{(L-1)} \cup \{v_{\mathrm{out}}\}$, where $V^{(j)} = \{v \in V : \mathrm{lv}(v) = j\}$ for $j = 0, \ldots, L-1$ and $\mathrm{lv}(v_{\mathrm{out}}) = L$;
- $E = \{(v, v') : \mathrm{lv}(v') = \mathrm{lv}(v) + 1\}$.

The integer L is the *depth* of the layered graph, the elements of $V^{(0)}$ are its *input nodes*, and v_{out} is its *output node*. With $d_j = |V^{(j)}|$ for $j = 0, \ldots, L-1$ and $d_L = 1$, the tuple $(d_0, \ldots, d_L)$ is its *architecture*.

Definition 7 (Neural network)**.** A *neural network* is a tuple $(V, E, \mathcal{W}, \mathcal{B}, \Psi)$ where (V, E) is a layered graph, $\mathcal{W} = \{w_{\tilde{v},v} \in \mathbb{R} : (v, \tilde{v}) \in E\}$ is the set of edge weights, $\mathcal{B} = \{b_v \in \mathbb{R} : v \in V \setminus V^{(0)}\}$ is the set of node biases, and $\Psi = \{\psi_v : \mathbb{R} \to \mathbb{R} : v \in V \setminus (V^{(0)} \cup \{v_{\mathrm{out}}\})\}$ is the set of activation functions associated with the *hidden nodes* $V \setminus (V^{(0)} \cup \{v_{\mathrm{out}}\})$. The *incoming weights* of a node v are $\{w_{v,u} : (u, v) \in E\}$ and its *outgoing weights* are $\{w_{\tilde{v},v} : (v, \tilde{v}) \in E\}$. The network is *shallow* if $V \setminus (V^{(0)} \cup \{v_{\mathrm{out}}\}) = \varnothing$, equivalently if (V, E) has depth 1, and *deep* otherwise.

Each node v computes an affine combination of its parents' outputs, then applies its activation function. Formally, the *local map* $\langle v \rangle$ is defined as follows. For $v_i^{(0)} \in V^{(0)}$, $i \in \{1, \ldots, d_0\}$, $\langle v_i^{(0)} \rangle(x) = x_i$, $x \in \mathbb{R}^{d_0}$. For $v_i^{(j)} \in V^{(j)}$ with $j \in \{1, \ldots, L-1\}$, $i \in \{1, \ldots, d_j\}$,

$$\langle v_i^{(j)} \rangle(x) = \psi_{v_i^{(j)}} \left(\sum_{i'=1}^{d_{j-1}} w_{v_i^{(j)}, v_{i'}^{(j-1)}} x_{i'} + b_{v_i^{(j)}} \right), \quad x \in \mathbb{R}^{d_{j-1}}. \tag{3}$$

For v_{out}, $\langle v_{\mathrm{out}} \rangle(x) = \sum_{i'=1}^{d_{L-1}} w_{v_{\mathrm{out}}, v_{i'}^{(L-1)}} x_{i'} + b_{v_{\mathrm{out}}}$, $x \in \mathbb{R}^{d_{L-1}}$.

The local map of a node depends only on its immediate inputs, the outputs of the preceding layer. Composing these maps from the input layer onward expresses each node's output as a function of the network's input; the resulting map at the output node is the function the network realizes. We now define these global maps.

Definition 8 (Global map and input-output map)**.** The *global map* $\langle\!\langle v \rangle\!\rangle : \mathbb{R}^{d_0} \to \mathbb{R}$ is defined recursively: for input nodes, $\langle\!\langle v_i^{(0)} \rangle\!\rangle = \langle v_i^{(0)} \rangle$; for $v_i^{(j)}$ with $j \geq 1$,

$$\langle\!\langle v_i^{(j)} \rangle\!\rangle(x) = \psi_{v_i^{(j)}} \left(\sum_{i'=1}^{d_{j-1}} w_{v_i^{(j)}, v_{i'}^{(j-1)}} \langle\!\langle v_{i'}^{(j-1)} \rangle\!\rangle(x) + b_{v_i^{(j)}} \right).$$

The *input-output map* of $\mathcal{N}$, denoted $\langle \mathcal{N} \rangle$, is the global map of the output node, $\langle \mathcal{N} \rangle = \langle\!\langle v_{\text{out}} \rangle\!\rangle$.

Two specific classes of networks, distinguished by their activation function, are of interest here.

Definition 9 (ρ-networks and σ-networks)**.** A neural network $(V, E, \mathcal{W}, \mathcal{B}, \Psi)$ is a *ρ-network* if $\psi_v = \rho$ for every $v \in V \setminus (V^{(0)} \cup \{v_{\text{out}}\})$, no activation function being applied at the output node, and a *σ-network* if $\psi_v = \sigma$ for every $v \in V \setminus V^{(0)}$, the activation function thus being applied at the output node as well, so that

$$\langle v_{\text{out}} \rangle(x) = \sigma\Big(\sum_{i'=1}^{d_{L-1}} w_{v_{\text{out}}, v_{i'}^{(L-1)}} x_{i'} + b_{v_{\text{out}}} \Big), \quad x \in \mathbb{R}^{d_{L-1}}. \tag{4}$$

The following example illustrates Definitions 5–8.

Example 1. Consider the ρ-network of architecture $(2, 2, 2, 1, 1)$, whose computational graph is shown in Figure 7. It has depth 4, and for instance $\text{par}(v_1^{(2)}) = \{v_1^{(1)}, v_2^{(1)}\}$. Reading the edge weights, node biases, and activations off the labels gives the local maps; those of the level-1 nodes are

$$\langle v_1^{(1)} \rangle(x) = \rho(x_1 + 2x_2 - 1), \qquad \langle v_2^{(1)} \rangle(x) = \rho(-2x_1 + 1).$$

Composing them from the input layer onward, as in Definition 8, yields the global maps. At level 2,

$$\langle\!\langle v_1^{(2)} \rangle\!\rangle = \rho(a), \qquad \langle\!\langle v_2^{(2)} \rangle\!\rangle = \rho(-a), \qquad \text{with } a = 1 + \langle\!\langle v_2^{(1)} \rangle\!\rangle - \langle\!\langle v_1^{(1)} \rangle\!\rangle,$$

so $\langle\!\langle v_1^{(2)} \rangle\!\rangle + \langle\!\langle v_2^{(2)} \rangle\!\rangle = |a|$ and

$$\langle \mathcal{N} \rangle(x) = \langle\!\langle v_{\text{out}} \rangle\!\rangle(x) = \rho\big(1 - |1 + \rho(1 - 2x_1) - \rho(x_1 + 2x_2 - 1)|\big),$$

a McNaughton function on $[0, 1]^2$ depicted in Figure 8.

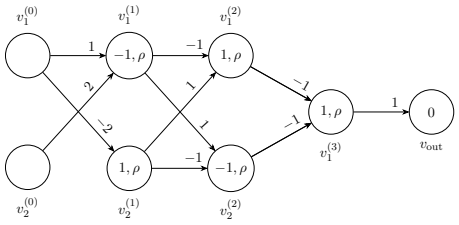

Figure 7: The computational graph of a neural network, with weights on the edges and biases and activation functions on the nodes; zero-weight edges are omitted.

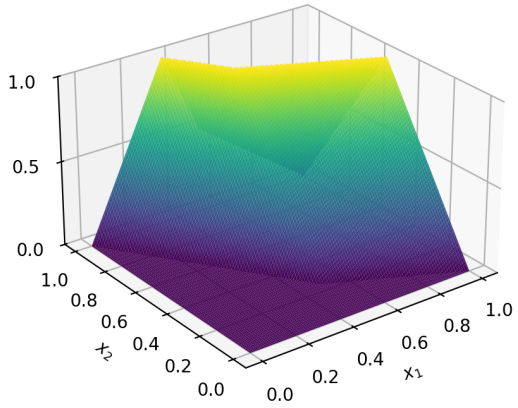

Figure 8: The function on $[0,1]^2$ realized by the network in Figure 7.

2.2 Extraction, Step I: Converting a ρ-network to a σ-network

We start from a ρ-network $\mathcal{N}$ whose realized function $\langle \mathcal{N} \rangle$ takes values in $[0,1]$. Since the network's domain is $[0,1]^{d_0}$ and every node has finitely many parents, each contributing a finite weight (Definitions 5 and 7), the input to every hidden node, the affine combination feeding its activation, is bounded. When the input to a ρ-node lies in an interval $[\ell, \mathcal{L}]$ with $\ell, \mathcal{L} \in \mathbb{R}$, the following identity lets it be replaced by one or more σ-nodes:

$$\rho(t) = \begin{cases} \sigma(t), & \mathcal{L} \leq 1, \\ \sigma(t) + \sigma(t-1) + \cdots + \sigma(t - \lceil \mathcal{L} \rceil + 1), & \mathcal{L} > 1. \end{cases} \tag{5}$$

We apply this substitution layer by layer, proceeding from the input layer to the output. Specifically, for each $v_i^{(j)} \in V^{(j)}$, $j = 1, \ldots, L-1$, we first compute the input interval $[\ell_{v_i^{(j)}}, \mathcal{L}_{v_i^{(j)}}]$ given by

$$\ell_{v_i^{(j)}} = b_{v_i^{(j)}} + \sum_{i'=1}^{d_{j-1}} \frac{w_{v_i^{(j)}, v_{i'}^{(j-1)}} - |w_{v_i^{(j)}, v_{i'}^{(j-1)}}|}{2}, \quad \mathcal{L}_{v_i^{(j)}} = b_{v_i^{(j)}} + \sum_{i'=1}^{d_{j-1}} \frac{w_{v_i^{(j)}, v_{i'}^{(j-1)}} + |w_{v_i^{(j)}, v_{i'}^{(j-1)}}|}{2}. \tag{6}$$

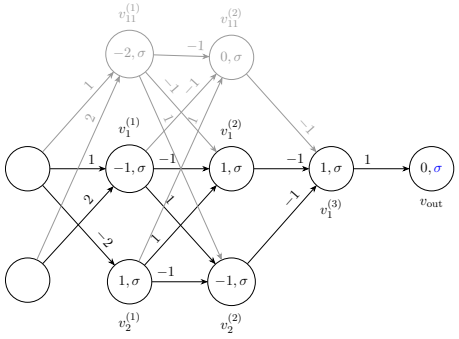

Figure 9: The σ-network obtained from Figure 7 by Extraction, Step I. The nodes introduced by the conversion, and their edges, are drawn in grey; the remaining nodes and edges are those of Figure 7.

Note that (6) relies on each level-$(j-1)$ parent outputting a value in $[0,1]$, which holds because those parents have already been converted to σ-nodes by the time $v_i^{(j)}$ is reached. The bounds (6) let the parent outputs range independently over $[0,1]$, though these are functions of the same network input, and may hence overestimate the true range of the affine combination. We then convert the node $v_i^{(j)}$ according to $\mathcal{L}_{v_i^{(j)}}$. If $\mathcal{L}_{v_i^{(j)}} \leq 1$, the activation function of $v_i^{(j)}$ is replaced by σ. If $\mathcal{L}_{v_i^{(j)}} > 1$, $v_i^{(j)}$ is replaced by $\lceil \mathcal{L}_{v_i^{(j)}} \rceil$ σ-nodes, each joined to the parents and children of $v_i^{(j)}$ by edges of the same weights, with biases $b_{v_i^{(j)}}, b_{v_i^{(j)}} - 1, \ldots, b_{v_i^{(j)}} - \lceil \mathcal{L}_{v_i^{(j)}} \rceil + 1$, according to (5). Each such substitution reproduces the local map of the node it replaces, preserving the network's input-output map. $\langle \mathcal{N} \rangle$ takes values in $[0,1]$ by assumption, on which σ acts as the identity. Associating σ with the output node, as required by Definition 9, therefore leaves $\langle \mathcal{N} \rangle$ unchanged and yields the corresponding σ-network $\mathcal{M}$.

Example 2. Continuing Example 1, the node $v_1^{(1)}$ has input interval $[-1,2]$, so it is replaced by two σ-nodes (biases -1 and -2 respectively). The node $v_2^{(1)}$ has input interval $[-1,1]$, so only its activation is swapped. Proceeding level by level yields the σ-network in Figure 9.

2.3 Extraction, Step II: Associating formulae with σ-nodes

Given the σ-network $\mathcal{M} = (V, E, \mathcal{W}, \mathcal{B}, \Psi)$ from Extraction, Step I, this step associates with each node $v \in V$ a Łukasiewicz formula, denoted $[v]$, whose truth function equals the local map of v.

For an input node $v_i^{(0)}$, $[v_i^{(0)}] = x_i$, with truth function $x_i^{\mathbb{I}} = \langle v_i^{(0)} \rangle$. For a non-input σ-node $v_i^{(j)}$ with local map $\sigma(f)$, where $f = m_1 x_1 + \cdots + m_n x_n + b$, formula extraction is based on the following result.

Lemma 1 ([19])**.** *Let $f(x) = m_1x_1 + \cdots + m_nx_n + b$ with $m_1, \ldots, m_n, b \in \mathbb{Z}$, and set $f_\circ = (m_1 - 1)x_1 + m_2x_2 + \cdots + m_nx_n + b$. If $m_1 \geq 1$, then on $[0,1]^n$,*

$$\sigma(f) = (\sigma(f_\circ) \oplus x_1) \odot \sigma(f_\circ + 1). \tag{7}$$

For all $m_1, \ldots, m_n, b \in \mathbb{Z}$, on $[0,1]^n$,

$$\sigma(f) = \neg\sigma(1 - f). \tag{8}$$

A proof, following [17], is given in Appendix B.

The formula associated with the node $v_i^{(j)}$ is $[v_i^{(j)}] = \texttt{EXTR}(\sigma(f))$, where EXTR is the algorithm that applies Lemma 1 recursively, lowering the coefficients m_i until the recursion terminates. On input $\sigma(f)$, let ℓ and $\mathcal{L}$ be the minimum and maximum of f over $[0,1]^n$. If $\ell \geq 1$, EXTR returns the constant 1; if $\mathcal{L} \leq 0$, it returns the constant 0. Otherwise $\ell < 1$ and $\mathcal{L} > 0$. Let k be the smallest index with $m_k \neq 0$ (thus $m_1 = \cdots = m_{k-1} = 0$), so x_k is the first variable present. As a nonzero integer, m_k is either ≥ 1 or ≤ -1, and EXTR eliminates x_k accordingly. Specifically, if $m_k \geq 1$, then by (7) (with x_k in the role of x_1) $\texttt{EXTR}(\sigma(f)) = (\tau_1 \oplus x_k) \odot \tau_2$, where τ_1 and τ_2 are the results of EXTR applied to $\sigma((m_k - 1)x_k + m_{k+1}x_{k+1} + \cdots + m_nx_n + b)$ and to the same argument with b replaced by $b + 1$. If $m_k \leq -1$, then by (8) $\texttt{EXTR}(\sigma(f)) = \neg\,\texttt{EXTR}(\sigma(1 - f))$, where $1 - f = -m_kx_k - \cdots - m_nx_n - b + 1$ has leading coefficient $-m_k \geq 1$, satisfying the assumption of Lemma 1. Each application of (7) lowers $\sum_i |m_i|$, and (8) makes the leading coefficient positive and so is followed by (7); hence EXTR terminates. Eliminating the first present variable is a convention—Lemma 1 applies with any present variable in the role of x_1, and the recursion terminates for every choice. Since every step preserves the truth function, the choice of the variable to be eliminated affects only the syntactic form of the resulting formula, not its truth function. EXTR, with the convention of eliminating the first variable present, is used throughout the paper; subsequent definitions and results are stated relative to this convention. The recursion may leave $0 \oplus \tau$ or $\tau \odot 1$ in the output; we identify these with τ (Ax. 1, 5, 5′). Each step of EXTR rewrites $\sigma(f)$ by one of the identities of Lemma 1, which hold for all $x \in [0,1]^n$, so that the formula returned has truth function $\sigma(f)$, that is, $[v_i^{(j)}]^{\mathbb{I}} = \langle v_i^{(j)} \rangle$.

Example 3. Consider a σ-node v with local map $\langle v \rangle = \sigma(x_1 - x_2 + x_3 - 1)$ on $[0,1]^3$. Its input interval has

$$\ell_v = \min_{x \in [0,1]^3} (x_1 - x_2 + x_3 - 1) = -2, \qquad \mathcal{L}_v = \max_{x \in [0,1]^3} (x_1 - x_2 + x_3 - 1) = 1,$$

so $\ell_v < 1$ and $\mathcal{L}_v > 0$. The first nonzero coefficient is that of x_1, equal to 1, so applying (7) eliminates x_1 according to

$$\sigma(x_1 - x_2 + x_3 - 1) = (\sigma(-x_2 + x_3 - 1) \oplus x_1) \odot \sigma(-x_2 + x_3).$$

EXTR is then applied to the two σ-terms. Since $\sigma(-x_2 + x_3 - 1)$ has $\mathcal{L} = 0$, EXTR returns 0. The term $\sigma(-x_2 + x_3)$ has input interval $[-1, 1]$ with first nonzero coefficient -1 (that of x_2), so by (8) the sign is flipped according to

$$\sigma(-x_2 + x_3) = \neg\sigma(x_2 - x_3 + 1),$$

and applying (7) then eliminates x_2 according to

$$\sigma(x_2 - x_3 + 1) = (\sigma(-x_3 + 1) \oplus x_2) \odot \sigma(-x_3 + 2),$$

where $\texttt{EXTR}(\sigma(-x_3 + 1)) = \neg x_3$ and $\texttt{EXTR}(\sigma(-x_3 + 2)) = 1$. Substituting back, the formula associated with v is

$$[v] = x_1 \odot \neg(\neg x_3 \oplus x_2).$$

2.4 Extraction, Step III: Composing by substitution

Throughout this step, $v_i^{(j)}$, d_j, and $\langle\!\langle \cdot \rangle\!\rangle$ refer to the σ-network $\mathcal{M}$ obtained in Extraction, Step I from the ρ-network $\mathcal{N}$. The Łukasiewicz formula corresponding to the full network is obtained by substituting, for each edge $(v_{i'}^{(j-1)}, v_i^{(j)})$, $i' = 1, \ldots, d_{j-1}$, $i = 1, \ldots, d_j$, $j = 1, \ldots, L$, the formula $[v_{i'}^{(j-1)}]$ into all occurrences of $x_{i'}$ in $[v_i^{(j)}]$, proceeding layer by layer from the inputs to the output. The result is a single Łukasiewicz formula, and we show next that its truth function equals $\langle \mathcal{N} \rangle$ on $[0,1]^{d_0}$.

The substitutions act on the formulae, and correspond to composition at the level of truth functions as follows. We claim that, for every Łukasiewicz formula τ in the variables $x_1, \ldots, x_d$ and all Łukasiewicz formulae $\chi_1, \ldots, \chi_d$, the formula $\widetilde{\tau}$ obtained from τ by replacing every occurrence of $x_{i'}$ by $\chi_{i'}$, $i' = 1, \ldots, d$, satisfies

$$\widetilde{\tau}^{\mathbb{I}} = \tau^{\mathbb{I}}(\chi_1^{\mathbb{I}}, \ldots, \chi_d^{\mathbb{I}}). \tag{9}$$

Here a formula in the variables $x_1, \ldots, x_d$ need not contain all of them; its truth function is in every case regarded as a map $[0,1]^d \to [0,1]$, so that for a constant τ it is the constant map of d variables. The claim holds trivially when τ is a single variable. It holds for a constant τ as well, since the replacement then leaves τ unchanged and both sides of (9) equal that constant. Suppose now that the claim holds for τ', and consider $\tau = \neg\tau'$. The replacement affects only the variable occurrences in τ' and leaves the $\neg$ in place, so that $\widetilde{\tau} = \neg(\widetilde{\tau'})$, where $\widetilde{\tau'}$ denotes the result of the replacement applied to τ', and evaluation applies $\neg$ in $\mathbb{I}$ to the truth function of $\widetilde{\tau'}$ (Section 1.3), whence

$$\widetilde{\tau}^{\mathbb{I}} = \neg\big(\widetilde{\tau'}^{\,\mathbb{I}}\big) = \neg\big(\tau'^{\mathbb{I}}(\chi_1^{\mathbb{I}}, \ldots, \chi_d^{\mathbb{I}})\big) = \tau^{\mathbb{I}}(\chi_1^{\mathbb{I}}, \ldots, \chi_d^{\mathbb{I}}).$$

The cases $\tau = \gamma \oplus \eta$ and $\tau = \gamma \odot \eta$ follow in the same manner, the binary operation being applied to the truth functions of both parts and the induction hypothesis being invoked for γ and for η. As every Łukasiewicz formula is built from variables and constants by these three operations, (9) holds in general.

We now apply (9) level by level, proving by induction on j that the level-j formulae, once the replacements performed in passing from the inputs to level j have been carried out, have truth functions $\langle\!\langle v_i^{(j)} \rangle\!\rangle$, $i = 1, \ldots, d_j$. At $j = 1$ the $\chi_{i'}$ are the input-node formulae $[v_{i'}^{(0)}] = x_{i'}$, so that the replacement leaves the level-1 formulae unchanged; their truth functions are $\langle v_i^{(1)} \rangle = \langle\!\langle v_i^{(1)} \rangle\!\rangle$ (Section 2.3). For the induction step, let $j \in \{2, \ldots, L\}$ and denote by $\widetilde{\tau}_{i'}$, $i' = 1, \ldots, d_{j-1}$, the level-$(j-1)$ formulae once the replacements up to level $j-1$ have been carried out, so that $\widetilde{\tau}_{i'}^{\mathbb{I}} = \langle\!\langle v_{i'}^{(j-1)} \rangle\!\rangle$ by the induction hypothesis. Taking $\tau = [v_i^{(j)}]$ and $\chi_{i'} = \widetilde{\tau}_{i'}$, (9) shows that the formula obtained at $v_i^{(j)}$ has as truth function

the local map $\langle v_i^{(j)} \rangle$ evaluated at the global maps $\langle\langle v_1^{(j-1)} \rangle\rangle, \ldots, \langle\langle v_{d_{j-1}}^{(j-1)} \rangle\rangle$ of the level-$(j-1)$ nodes, the parents of $v_i^{(j)}$, which is the global map $\langle\langle v_i^{(j)} \rangle\rangle$ (Definition 8). At $j = L$ the claim reads $\langle\langle v_{\text{out}} \rangle\rangle = \langle \mathcal{M} \rangle = \langle \mathcal{N} \rangle$ (Section 2.2), the truth function of the formula returned by Extraction, Step III.

3 Graphical representation of formulae

The extraction algorithm of [22], which we recast in Section 2 to act on the ReLU network's computational graph, produces a Łukasiewicz formula as a finite string. Extraction, Step I converts the ReLU network to a σ-network and Step II labels each node of that σ-network with a formula whose truth function is the node's local map; Step III then composes these node formulae by substitution into a single string. It is this last step that discards the network's graph structure, thereby losing the information needed to reconstruct the network from the formula. Stopping at Step II instead—retaining the σ-network's layered graph together with its node formulae—preserves that structural information. Working on this architecture-preserving graph, rather than on the flattened formula, is what makes identification of networks possible. This section systematically develops the layered graph, with its node formulae, into a graphical representation of the extracted formula. The result is a *compositional normal form*[2]. By Proposition 3, every McNaughton function admits such a form. The entire identification program in this paper is organized around the compositional normal form.

The following example demonstrates the loss of structure in Extraction, Step III concretely. The same McNaughton function is realized by two networks of different architecture, and Step III maps both to strings that differ only in bracketing—a difference dissolved by the associativity axiom Ax. 2′—so the extracted string cannot tell the two architectures apart.

Example 4. The two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ in Figure 10 have different architectures, $(1, 2, 1, 1)$ and $(1, 1, 1, 1)$, but realize the same function on $[0, 1]$, namely

$$\langle \mathcal{N}_1 \rangle(x_1) = \langle \mathcal{N}_2 \rangle(x_1) = \begin{cases} 0, & 0 \leq x_1 \leq 5/6, \\ 6x_1 - 5, & 5/6 < x_1 \leq 1. \end{cases}$$

Reading the weights and biases off Figure 10, $\mathcal{N}_1$ computes $\rho(\rho(4x_1 - 3) + \rho(2x_1 - 1) - 1)$ and $\mathcal{N}_2$ computes $\rho(3\rho(2x_1 - 1) - 2)$, both equal to $\max\{6x_1 - 5, 0\}$ on $[0, 1]$. Applying Extraction, Steps I and II yields distinct σ-networks $\mathcal{M}_1$, $\mathcal{M}_2$ (Figure 11) with distinct node formulae (Figure 12); the passage from the σ-networks to the node formulae is done by applying Lemma 1 at each node. Because every node's output stays within the unit interval $[0, 1]$, σ and ρ agree at every node, and Extraction, Step I introduces no new nodes; it merely replaces each ρ by σ and applies σ at the output node, so $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide with $\mathcal{N}_1$ and $\mathcal{N}_2$ apart from the activation function. Extraction, Step III—composing the node formulae by substitution, layer by layer from the inputs to the output—flattens $\mathcal{M}_1$ to $(x_1 \odot (x_1 \odot (x_1 \odot x_1))) \odot (x_1 \odot x_1)$ and $\mathcal{M}_2$ to $(x_1 \odot x_1) \odot ((x_1 \odot x_1) \odot (x_1 \odot x_1))$, two

2. Classical normal forms for Łukasiewicz logic are *flat*. The form of [10], for instance, writes every McNaughton function as a conjunction of disjunctions of *simple McNaughton functions*, the conjunction and disjunction being the lattice connectives $\wedge = \min$ and $\vee = \max$ (not the strong $\odot, \oplus$). The form developed here is compositional instead; Remark 1 draws the comparison in detail.

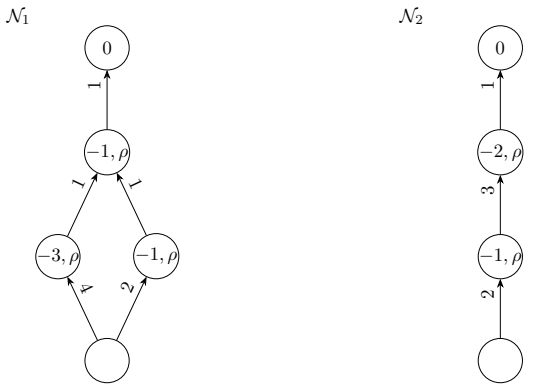

Figure 10: Two functionally equivalent ReLU networks $\mathcal{N}_1$, $\mathcal{N}_2$ with different architectures. A hidden node is labeled by its bias and activation function, and an edge by its weight; the output node applies no activation function and is labeled by its bias alone.

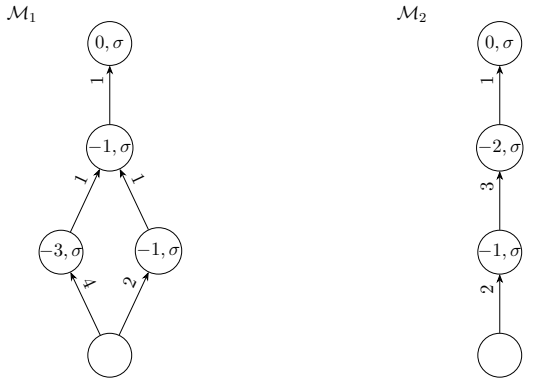

Figure 11: σ-networks $\mathcal{M}_1$ and $\mathcal{M}_2$ obtained from $\mathcal{N}_1$ and $\mathcal{N}_2$ by Extraction, Step I.

$\odot$-products of six copies of x_1 differing only in bracketing, hence interderivable by the associativity axiom Ax. $2'$ alone. That the two extracted formulae differ only in bracketing is special to this example. Functionally equivalent networks need not extract to strings so closely related; two networks realizing $\max\{x_1, x_2\}$, for instance, can extract to the distinct formulae $x_2 \oplus (x_1 \odot \neg x_2)$ and $x_1 \oplus \neg(\neg x_2 \oplus x_1)$ (Figure 13). Theorem 2 guarantees that any two strings with the same underlying truth function are related through application of the MV axioms; Section 4 lifts this derivability to substitution graphs.

3.1 Substitution graphs

In this subsection we define the *substitution graph*, the central object of the paper's development. A substitution graph consists of a layered graph together with Łukasiewicz formulae attached to its nodes, the edges prescribing how these formulae compose; the represented formula is read off the graph by substituting, along each edge, the formula at

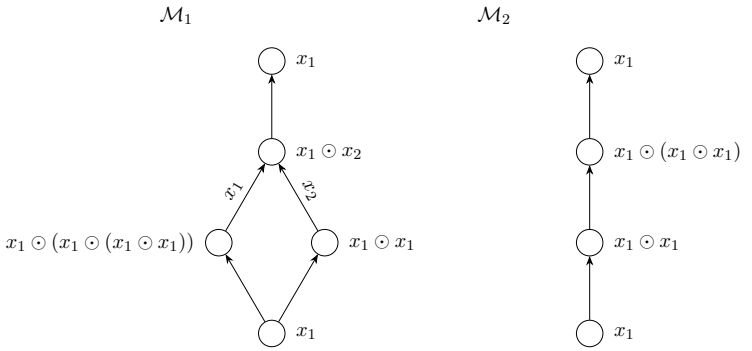

Figure 12: Formulae associated with individual nodes after Extraction, Step II. Each node is labeled by its extracted formula, in the variables supplied by its parents; substituting these formulae bottom-up yields the formula realized by the network. For nodes with several parents, the incoming edges are labeled by the variable each supplies—so the node of $\mathcal{M}_1$ where the two branches meet forms their conjunction $x_1 \odot x_2$.

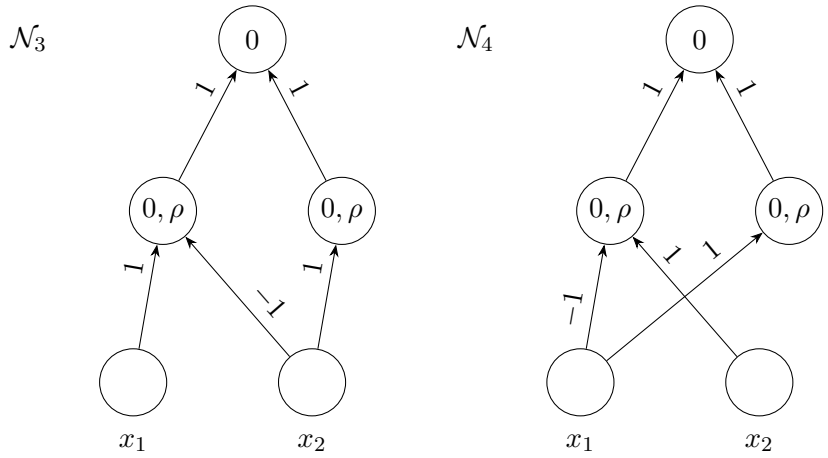

Figure 13: Two networks $\mathcal{N}_3$ and $\mathcal{N}_4$ that both realize $\max\{x_1, x_2\}$ on $[0,1]^2$ yet extract to the distinct formulae $x_2 \oplus (x_1 \odot \neg x_2)$ and $x_1 \oplus \neg(\neg x_2 \oplus x_1)$; zero-weight edges are omitted.

a node for the corresponding variable in its children. While the definition stands on its own, the substitution graphs from which our derivations start arise from extraction, as the layered graph of a σ-network with each node labeled by the formula extracted from it. The underlying operation—replacing the variables of a formula by other formulae—is *substitution*, which we now define formally.

Definition 10 (Substitution)**.** Let $x_1, \ldots, x_n$ be propositional variables, and let $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq \{x_1, \ldots, x_n\}$. A *substitution* ζ is a finite set $\zeta = \{x_{i_1} \mapsto \chi_1, \ldots, x_{i_k} \mapsto \chi_k\}$ whose *substitutors*

$\chi_1, \ldots, \chi_k$ are Łukasiewicz formulae; the set $\{x_{i_1}, \ldots, x_{i_k}\}$ is the *domain* of ζ. Applying ζ to a Łukasiewicz formula τ simultaneously replaces every occurrence of each x_{i_j} by χ_j; the resulting formula is denoted $\tau(\zeta)$, written out as $\tau(\{x_{i_1} \mapsto \chi_1, \ldots, x_{i_k} \mapsto \chi_k\})$. If a particular x_{i_j} does not occur in τ, replacing it has no effect, e.g., $(x_2 \odot x_3)(\{x_1 \mapsto x_1 \oplus x_1\}) = x_2 \odot x_3$. When τ is a constant ($\tau \in \{0, 1\}$), no replacement applies and $\tau(\zeta) = \tau$.

For example, $(x_1 \odot \neg x_1)(\{x_1 \mapsto x_2 \oplus x_3\}) = (x_2 \oplus x_3) \odot \neg(x_2 \oplus x_3)$.
We now attach formulae to the nodes of a layered graph and use the edges to compose them.

Definition 11 (Substitution graph)**.** Let (V, E) be a layered graph of architecture $(d_0, \ldots, d_L)$, and let $\mathcal{C} = \{[v] : v \in V\}$ assign a Łukasiewicz formula to every node, as follows. Each input node carries a distinct variable, $[v_i^{(0)}] = x_i$, $i = 1, \ldots, d_0$. A node $v_i^{(j)}$ at level $j \in \{1, \ldots, L\}$, $i = 1, \ldots, d_j$, has the d_{j-1} nodes of level $j-1$ as its parents, and carries a Łukasiewicz formula $[v_i^{(j)}]$ in the variables $x_1, \ldots, x_{d_{j-1}}$ (for $j = 1$, the d_0 input variables $x_1, \ldots, x_{d_0}$), the variable $x_{i'}$ being associated with the parent $v_{i'}^{(j-1)}$, $i' = 1, \ldots, d_{j-1}$. We call $\mathcal{G} = (V, E, \mathcal{C})$ a *substitution graph* and (V, E) its *layered graph*.

The node formulae can be composed through layer substitutions. For each $j = 1, \ldots, L$, the *layer substitution*

$$\zeta^{(j-1,j)} = \{x_{i'} \mapsto [v_{i'}^{(j-1)}] : i' = 1, \ldots, d_{j-1}\}$$

replaces, in the formula of every level-j node, each variable by the formula of the associated parent at level $j-1$. The formula *represented* by $\mathcal{G}$ is obtained by applying the layer substitutions starting from $[v_{\text{out}}]$ and working layer by layer toward the inputs,

$$[\mathcal{G}] = [v_{\text{out}}]\,(\zeta^{(L-1,L)})(\zeta^{(L-2,L-1)}) \cdots (\zeta^{(0,1)}), \tag{10}$$

that is, $\zeta^{(L-1,L)}$ is applied to $[v_{\text{out}}]$, then $\zeta^{(L-2,L-1)}$ to the resulting formula, and so on, finishing with $\zeta^{(0,1)}$—iterated application reads from the left, each substitution applying to the formula produced thus far, i.e., $\tau(\zeta)(\zeta') = \big(\tau(\zeta)\big)(\zeta')$.

Reading out $[\mathcal{G}]$ via (10) composes the node formulae from the output inward, whereas Extraction, Step III composes the same formulae from the inputs outward. Establishing that the two yield the same formula requires the composition of substitutions, which we define next.

Definition 12 (Substitution composition)**.** [12, pp. 74][2] Let $\zeta = \{x_{i_1} \mapsto \chi_1, \ldots, x_{i_k} \mapsto \chi_k\}$ be a substitution with Łukasiewicz substitutors $\chi_1, \ldots, \chi_k$, and let ζ' be a substitution. The *composition* $\zeta \bullet \zeta'$ applies ζ' to each substitutor of ζ,

$$\zeta \bullet \zeta' = \{x_{i_1} \mapsto \chi_1(\zeta'), \ldots, x_{i_k} \mapsto \chi_k(\zeta')\}.$$

Lemma 2. *Let $\zeta = \{x_{i_1} \mapsto \chi_1, \ldots, x_{i_k} \mapsto \chi_k\}$ and ζ' be substitutions, and let τ be a formula with variables in $\{x_{i_1}, \ldots, x_{i_k}\}$. Applying ζ to τ and then ζ' to the result yields the same formula as applying the single substitution $\zeta \bullet \zeta'$ to τ,*

$$\big(\tau(\zeta)\big)(\zeta') = \tau(\zeta \bullet \zeta').$$

Proof See Appendix C.1. ■

We now formalize the substitution graph that extraction associates with a ReLU network.

Definition 13 (Extracted substitution graph)**.** For a ReLU network $\mathcal{N}$, let (V, E) be the layered graph of the σ-network obtained from $\mathcal{N}$ by Extraction, Step I, and let $\mathcal{C}$ be the node formulae produced by Extraction, Step II. We call $\texttt{ext}(\mathcal{N}) = (V, E, \mathcal{C})$ the substitution graph *extracted* from $\mathcal{N}$.

It remains to relate $[\mathcal{G}]$, the formula represented by $\mathcal{G}$ in the sense of Definition 11, to the string produced by Extraction, Step III. Step III composes the node formulae from the inputs outward, whereas (10) composes them from the output inward; the next proposition shows that the two orders yield the same formula, so $[\mathcal{G}]$ is exactly the string Extraction, Step III returns.

Proposition 2. *For every substitution graph $\mathcal{G}$, Extraction, Step III, applied to the node formulae of $\mathcal{G}$, returns the formula $[\mathcal{G}]$ defined in* (10)*. Moreover, the iterated application in* (10) *can be carried out in a single substitution,*

$$[\mathcal{G}] = [v_{out}](\eta_L), \tag{11}$$

where $\eta_1 = \zeta^{(0,1)}$ and $\eta_{j+1} = \zeta^{(j,j+1)} \bullet \eta_j$, for $j = 1, \ldots, L-1$.

Proof Equation (10) reads out $[\mathcal{G}]$ by applying the layer substitutions $\zeta^{(L-1,L)}, \ldots, \zeta^{(0,1)}$ to $[v_{\text{out}}]$ from the output inward, whereas Extraction, Step III rewrites the node formulae from the inputs outward. We show that both procedures amount to applying to $[v_{\text{out}}]$ the same substitution η_L, given by the recursion $\eta_1 = \zeta^{(0,1)}$, $\eta_{j+1} = \zeta^{(j,j+1)} \bullet \eta_j$, for $j = 1, \ldots, L-1$.

For (10), we claim that, for $j = 1, \ldots, L$, every formula θ with variables in $\{x_1, \ldots, x_{d_{j-1}}\}$ satisfies

$$\theta\,(\zeta^{(j-1,j)})(\zeta^{(j-2,j-1)}) \cdots (\zeta^{(0,1)}) = \theta(\eta_j);$$

the case $j = L$, $\theta = [v_{\text{out}}]$ gives $[\mathcal{G}] = [v_{\text{out}}](\eta_L)$. The claim is proved by induction on j. For $j = 1$ it follows from the definition of η_1. For the step from j to $j+1$, assume that every formula θ with variables in $\{x_1, \ldots, x_{d_{j-1}}\}$ satisfies the identity above; we must show the corresponding statement for $j+1$, that is, that every formula θ' with variables in $\{x_1, \ldots, x_{d_{(j+1)-1}}\} = \{x_1, \ldots, x_{d_j}\}$, the domain of $\zeta^{(j,j+1)}$, satisfies $\theta'\,(\zeta^{(j,j+1)})(\zeta^{(j-1,j)}) \cdots (\zeta^{(0,1)}) = \theta'(\eta_{j+1})$. Fix such a θ'. Every variable of θ' is replaced, under $\zeta^{(j,j+1)}$, by its substitutor, a level-j node formula with variables in $\{x_1, \ldots, x_{d_{j-1}}\}$ (Definition 11); the formula $\theta'(\zeta^{(j,j+1)})$ hence has its variables in $\{x_1, \ldots, x_{d_{j-1}}\}$. The induction assumption, applied with $\theta = \theta'(\zeta^{(j,j+1)})$, thus yields the first equality in

$$\theta'\,(\zeta^{(j,j+1)})(\zeta^{(j-1,j)}) \cdots (\zeta^{(0,1)}) = \big(\theta'(\zeta^{(j,j+1)})\big)(\eta_j) = \theta'(\zeta^{(j,j+1)} \bullet \eta_j) = \theta'(\eta_{j+1});$$

the second equality is Lemma 2, applicable since the variables of θ' lie in $\{x_1, \ldots, x_{d_j}\}$, the domain of $\zeta^{(j,j+1)}$.

For Extraction, Step III, we claim that, for $j = 1, \ldots, L$, the rewritten formula of every level-j node $v_i^{(j)}$ is $[v_i^{(j)}](\eta_j)$; the case $j = L$ shows that Step III returns $[v_{\text{out}}](\eta_L)$.

The claim is again proved by induction on j. For $j = 1$, Step III substitutes, for each edge $(v_{i'}^{(0)}, v_i^{(1)})$, the formula $[v_{i'}^{(0)}] = x_{i'}$ (Definition 11) into all occurrences of $x_{i'}$ in $[v_i^{(1)}]$; the substitution applied, $\zeta^{(0,1)} = \{x_{i'} \mapsto x_{i'}\}$, is thus the identity, and the rewritten formula of $v_i^{(1)}$ is $[v_i^{(1)}] = [v_i^{(1)}](\zeta^{(0,1)}) = [v_i^{(1)}](\eta_1)$. For the step from j to $j+1$, assume that the rewritten formula of every level-j node $v_i^{(j)}$ is $[v_i^{(j)}](\eta_j)$; we must show that the rewritten formula of every level-$(j+1)$ node $v_i^{(j+1)}$ is $[v_i^{(j+1)}](\eta_{j+1})$. Step III replaces, for each edge $(v_{i'}^{(j)}, v_i^{(j+1)})$, all occurrences of $x_{i'}$ in $[v_i^{(j+1)}]$ by the rewritten parent formula, which, by the induction assumption, is $[v_{i'}^{(j)}](\eta_j)$; it thus applies to $[v_i^{(j+1)}]$ the substitution $\{x_{i'} \mapsto [v_{i'}^{(j)}](\eta_j) : i' = 1, \ldots, d_j\}$, which, by Definition 12, equals $\zeta^{(j,j+1)} \bullet \eta_j$, that is, η_{j+1}. The rewritten formula of $v_i^{(j+1)}$ is therefore $[v_i^{(j+1)}](\eta_{j+1})$, as desired. ■

Extraction identifies the truth function of $[\mathtt{ext}(\mathcal{N})]$ with $\langle\mathcal{N}\rangle$, and construction the function realized by the network built from the graph $\mathcal{G}$ with $[\mathcal{G}]^{\mathbb{I}}$. Both compose along the edges of (V, E), by syntactic substitution on the node formulae and by function composition on their truth functions. The following lemma states that the two agree in the sense that the truth function of the composed formula $[\mathcal{G}]$ is the function composition of the truth functions of the node formulae.

Lemma 3. *Let $\mathcal{G} = (V, E, \mathcal{C})$ be a substitution graph of depth L and architecture $(d_0, \ldots, d_L)$, and let $\mathcal{K}$ be the σ-network (Definition 9) with layered graph (V, E) whose nodes have local maps $\langle v\rangle = [v]^{\mathbb{I}}$, $v \in V$. Then $\langle\mathcal{K}\rangle = [\mathcal{G}]^{\mathbb{I}}$ on $[0,1]^{d_0}$.*
Proof See Appendix C.2. ■

Applied to the extracted graph $\mathtt{ext}(\mathcal{N})$, Proposition 2 shows that $[\mathtt{ext}(\mathcal{N})]$ is the formula Extraction, Step III produces from $\mathcal{N}$. The node formulae of $\mathtt{ext}(\mathcal{N})$, delivered by Extraction, Step II, satisfy $[v]^{\mathbb{I}} = \langle v\rangle$ (Section 2.3), so Lemma 3, with $\mathcal{K}$ the σ-network obtained from $\mathcal{N}$ by Extraction, Step I, yields $[\mathtt{ext}(\mathcal{N})]^{\mathbb{I}} = \langle\mathcal{K}\rangle$, which equals $\langle\mathcal{N}\rangle$ (Section 2.2), on $[0,1]^{d_0}$.

3.2 A normal form based on substitution

A McNaughton function is the truth function of infinitely many Łukasiewicz formulae, since syntactically different formulae can evaluate to the same function, e.g., $\neg\neg x_1$ and x_1 both compute the identity map $x_1 \mapsto x_1$, and $(x_1 \odot \neg x_2) \oplus x_2$ and $(x_2 \odot \neg x_1) \oplus x_1$ both realize $\max\{x_1, x_2\}$. Theorem 2 characterizes this redundancy, showing that two Łukasiewicz formulae have the same truth function if and only if they are interderivable by the MV axioms.

The substitution-graph encoding of a Łukasiewicz formula contains more information than the formula itself. Each node in the graph carries a component Łukasiewicz formula, and the edges record how these node formulae compose into the overall formula. Such a graph structure is what the extraction algorithm of Section 2 delivers from a ReLU network, and construction (Section 5) reads a network back from the graph. Theorem 2 reduces the redundancy among a McNaughton function's Łukasiewicz formulae to interderivability by the MV axioms. Representing any of these functionally equivalent formulae by a substitution graph adds redundancy of two further kinds, since a formula can in general be composed

in many different ways. The first source of redundancy is *per-node*. Replacing the formula at a given node by any other formula corresponding to the same truth function leaves the function represented by the graph unchanged, so each node's formula is fixed only up to MV-equivalence—the logical redundancy, localized to a node. The second source is *architectural*. Distinct substitution graphs of different architecture can represent the same function.

For the substitution graph $\mathcal{G}$, the represented formula $[\mathcal{G}]$ is obtained by composing the node formulae along the edges, *flattening* $\mathcal{G}$ to a single string that no longer encodes the layered graph (V, E). Architecturally distinct substitution graphs realizing the same function may therefore flatten to strings differing only in bracketing (such as $\mathcal{M}_1$ and $\mathcal{M}_2$ in Example 4) or to altogether different strings with the same underlying truth function. The architecture is information the graph carries but its flattening does not, which is why construction builds a network from the substitution graph and not from the formula alone, and why the compositional normal form is needed in place of the flat normal forms of the existing literature.

Architectural redundancy comprises two parts. One part is substantive, consisting of genuinely different architectures realizing the same function, and is encompassed by the redundancy characterized by Theorem 3. The other is trivial, the degenerate structure excluded by the non-degeneracy conditions in Definition 14 below. For a non-degenerate network $\mathcal{N}$, $\texttt{constr}(\texttt{ext}(\mathcal{N})) = \mathcal{N}$; thus `constr` is a left inverse of `ext` (Proposition 5), so that non-degeneracy is sufficient for exact recovery. Four phenomena produce this degenerate structure. Call a hidden node *active* at an input x if $\langle\langle v \rangle\rangle(x) > 0$ and *inactive* if $\langle\langle v \rangle\rangle(x) = 0$. A node inactive on all of $[0,1]^{d_0}$ is called *dead*; its global map is then identically zero, so it contributes nothing to its children and can be removed without changing the network's input-output map. A node active on all of $[0,1]^{d_0}$ has activation reducing to the identity map, so it can be absorbed into an adjacent layer. Two hidden nodes at the same level with the same local map produce identical outputs and can be merged into one node, with their outgoing weights added. Finally, a hidden node all of whose outgoing weights are zero can be removed without changing the network's input-output map. In each of the four cases the network reduces to a smaller one realizing the same function. The following definition rules out all four phenomena.

Definition 14. A ReLU network with input nodes $V^{(0)}$ and output node v_{out} is *non-degenerate on* $[0,1]^{d_0}$ if:

- every hidden node $v \in V \setminus (V^{(0)} \cup \{v_{\text{out}}\})$ is active for some $x \in [0,1]^{d_0}$;
- every hidden node is inactive for some $x \in [0,1]^{d_0}$;
- no two nodes at the same level have the same local map;
- every hidden node has at least one nonzero outgoing weight.

With the trivial redundancies excluded by non-degeneracy, what remains is to fix the form of the Łukasiewicz formula at each node of the substitution graph. We do this by requiring every node formula to be the result of applying Extraction, Step II to a single σ-node; such formulae are the *minterms* defined below. Substitution graphs extracted

from ReLU networks are automatically of this form—their node formulae are produced by Extraction, Step II—so the requirement adds no hypothesis to Proposition 5; rather, it delimits the class of substitution graphs on which the construction algorithm (Section 5) operates. A substitution graph whose node formulae are all minterms is said to be in *normal form*.

Definition 15 (Minterms and normal form)**.** A Łukasiewicz formula τ is a *minterm* if $\tau = \texttt{EXTR}(\sigma(m_1x_1 + \cdots + m_nx_n + b))$ for some $m_1, \ldots, m_n, b \in \mathbb{Z}$; the minterms form the set $\mathcal{C}_{\text{norm}}$. A substitution graph $\mathcal{G} = (V, E, \mathcal{C})$ is *normal* if every formula $[v] \in \mathcal{C}$ is a minterm. The truncations of integer-affine maps, that is, the local maps of σ-nodes, form the set

$$C = \{\sigma(m_1x_1 + \cdots + m_nx_n + b) : n \in \mathbb{N},\ m_1, \ldots, m_n, b \in \mathbb{Z}\}.$$

Our terminology differs from that of Boolean algebra, where a *minterm* is a conjunction containing every variable exactly once.

MV derivations carried out on substitution graphs can pass through graphs that are not normal, that is, graphs carrying at one or more nodes a formula that is not the extract of a single σ-node; `constr` does not apply to such graphs, which therefore carry no reading as ReLU networks in the sense of Definition 7. This is the price paid for representing formulae extracted from ReLU networks in a way that retains the network's layered structure; we detail this in Section 4. Networks do, however, reappear at the endpoints of such a derivation, which is why $\mathcal{N}_1 \overset{\mathcal{MV}}{\sim} \mathcal{N}_2$ is defined through a pullback (Definition 22) rather than as a chain of networks.

As established in Section 2.4, a minterm $\tau = \texttt{EXTR}(\sigma(f))$ satisfies $\tau^{\mathbb{I}} = \sigma(f) \in C$; the truth function of a node formula hence equals the local map of the σ-node the formula was extracted from. Each node of a normal substitution graph thereby carries a local map, read off from its formula, and on a substitution graph extracted from a ReLU network these are the local maps of the nodes of the corresponding σ-network (Lemma 4 below). We denote the mapping that takes a minterm to its truth function by κ.

Definition 16 (κ-mapping)**.** Define $\kappa : \mathcal{C}_{\text{norm}} \to C$ by

$$\kappa(\tau) = \begin{cases} 0, & \tau = 0, \\ 1, & \tau = 1, \\ \tau^{\mathbb{I}}, & \text{otherwise.} \end{cases} \tag{12}$$

Lemma 4. *Let $\mathcal{N}$ be a ReLU network and $\texttt{ext}(\mathcal{N}) = (V, E, \mathcal{C})$ the substitution graph extracted from $\mathcal{N}$ (Definition 13). Then $\kappa([v]) = \langle v \rangle$ for every $v \in V$, where $[v] \in \mathcal{C}$ is the formula carried by v and $\langle v \rangle$ its local map.*

Proof Since $\kappa(0) = 0 = 0^{\mathbb{I}}$ and $\kappa(1) = 1 = 1^{\mathbb{I}}$, we have $\kappa(\tau) = \tau^{\mathbb{I}}$ for every $\tau \in \mathcal{C}_{\text{norm}}$. For an input node, $[v_i^{(0)}] = x_i$, so $\kappa([v_i^{(0)}]) = x_i^{\mathbb{I}} = \langle v_i^{(0)} \rangle$. For a non-input node v with local map $\langle v \rangle = \sigma(f)$, extraction sets $[v] = \texttt{EXTR}(\sigma(f))$ and $[v]^{\mathbb{I}} = \sigma(f)$ (Section 2.4), so $\kappa([v]) = [v]^{\mathbb{I}} = \langle v \rangle$. ∎

While normality imposes a restriction on the substitution graphs admitted, the following result shows that every McNaughton function is representable by a normal substitution graph, which, in turn, corresponds to a ReLU network.

Proposition 3. *For $f : [0,1]^n \to [0,1]$, the following statements are equivalent:*

(i) f is a McNaughton function;
(ii) f is realized by a ReLU network with integer weights and biases;
(iii) $f = [\mathcal{G}]^{\mathbb{I}}$ for some normal substitution graph $\mathcal{G}$.

Proof For (i) $\Rightarrow$ (ii), by Theorem 1, f is the truth function of a Łukasiewicz formula τ, and by [22, Thm. 3.2] there is a ReLU network with integer weights and biases realizing $\tau^{\mathbb{I}} = f$ on $[0,1]^n$. For (ii) $\Rightarrow$ (iii), let $\mathcal{N}$ realize f. The substitution graph $\mathtt{ext}(\mathcal{N})$ extracted from $\mathcal{N}$ (Definition 13) is normal, its node formulae being the outputs of Extraction, Step II and hence minterms, and $[\mathtt{ext}(\mathcal{N})]^{\mathbb{I}} = \langle \mathcal{N} \rangle = f$, as observed after Proposition 2. For (iii) $\Rightarrow$ (i), $[\mathcal{G}]$ arises from the node formulae of $\mathcal{G}$ by substitution (10) and is therefore itself a Łukasiewicz formula, so $f = [\mathcal{G}]^{\mathbb{I}}$ is the truth function of a Łukasiewicz formula and hence a McNaughton function by Theorem 1. ■

Remark 1 (Relation to classical Łukasiewicz normal forms)**.** The compositional normal form developed here and the classical normal forms of Łukasiewicz logic differ in how they combine their building blocks, the formulae from which the normal form is built. Following Aguzzoli [1], Di Nola and Lettieri [10] write every McNaughton function as a conjunction of disjunctions of *simple McNaughton functions* $\sigma(m_1 x_1 + \cdots + m_n x_n + b)$, i.e., the truth functions of the minterms of Definition 15. The conjunction and disjunction are the connectives $x \wedge y = \neg(\neg x \odot y) \odot y$ and $x \vee y = \neg(\neg x \oplus y) \oplus y$, respectively. In shape this is the Łukasiewicz counterpart of the Boolean conjunctive normal form, the simple McNaughton functions occupying the place of the disjunctive clauses. Mundici [17] instead assigns to each McNaughton function f a unimodular triangulation of $[0,1]^n$ on each simplex of which f coincides with one of the linear polynomials p_j of Theorem 1, and writes f as a finite $\oplus$-sum of *Schauder hats*. Each vertex of the triangulation carries one such hat. The hat at v is a McNaughton function, affine on all simplices, positive at v, and zero at every other vertex, hence vanishing on the simplices that do not contain v. What all these forms have in common is that their building blocks are combined by a fixed pattern of connectives and never substituted into one another; we call such forms *flat*.

Our compositional normal form instead combines minterms by substitution along a substitution graph, so that a minterm may be substituted into the variables of another and the composition extends to the depth L of the graph. For a substitution graph extracted from a ReLU network $\mathcal{N}$, this depth is that of $\mathcal{N}$, since extraction turns the network's layered composition of maps into a composition of formulae rather than into a single string. To the best of our knowledge no normal form of this kind has appeared in the Łukasiewicz logic literature.

The flat normal forms prescribe the shape in which the function is written, which in the Boolean case makes the canonical disjunctive and conjunctive normal forms unique up to the order of their terms. Our compositional normal form prescribes no shape and is highly non-unique; Theorem 3 characterizes this non-uniqueness.

4 Syntactic derivation on substitution graphs

The identification program behind Theorem 3 proceeds in three steps—extraction, derivation, and construction. Extraction produces from the networks $\mathcal{N}_1$ and $\mathcal{N}_2$ the substitution graphs $\mathtt{ext}(\mathcal{N}_1)$ and $\mathtt{ext}(\mathcal{N}_2)$, whose represented formulae have the same truth function, $[\mathtt{ext}(\mathcal{N}_1)]^{\mathbb{I}} = [\mathtt{ext}(\mathcal{N}_2)]^{\mathbb{I}}$; by Theorem 3, the two graphs are therefore related by $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$. This section shows how the graph derivation $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$ is carried out, by a finite sequence of local operations—node rewrites by MV axioms, layer collapses, and layer expansions. The graphs passed through in the course of this rewriting need not all be normal, as already mentioned in Section 3. Construction—the algorithm returning a ReLU network from a normal substitution graph—is therefore applied only at the endpoints of the derivation $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$, where the graphs are guaranteed to be normal; in between, the operations act on graphs alone.

An MV derivation on graphs realizes an MV derivation of the formulae the graphs represent, the relation $\tau_1 \overset{\mathcal{MV}}{\sim} \tau_2$ appearing in Theorem 2. An MV derivation applies MV axioms to subformulae. For instance, in $\tau = ((x_1 \oplus x_2) \odot \neg(x_1 \oplus x_2)) \oplus x_3$, applying Ax. $3'$ $(x \odot \neg x = 0)$ to the subformula $(x_1 \oplus x_2) \odot \neg(x_1 \oplus x_2)$ replaces it by 0, Ax. 1 $(x \oplus y = y \oplus x)$ then yields $x_3 \oplus 0$, and Ax. 5 $(x \oplus 0 = x)$ reduces this to x_3. Four notions make the process precise—subformula, logic equation, instantiation of an axiom, and derivation.

Definition 17. A formula γ is a *subformula* of τ if it is a substring of τ that is itself a formula. Every formula is a subformula of itself.

Definition 18. A *logic equation* is an expression $\tau = \tau'$ where τ, τ' are formulae.

Definition 19. An *axiom* e is a logic equation $\epsilon = \epsilon'$, as in Ax. 1–$9'$ of Definition 4. The logic equation $\tau = \tau'$ is an *instantiation* of e if there exists a substitution ζ (Definition 10) with $\tau = \epsilon(\zeta)$ and $\tau' = \epsilon'(\zeta)$, or with $\tau = \epsilon'(\zeta)$ and $\tau' = \epsilon(\zeta)$. For example, $x_3 \oplus 0 = 0 \oplus x_3$ and $0 \oplus x_3 = x_3 \oplus 0$ both instantiate Ax. 1 under $\zeta = \{x \mapsto x_3,\, y \mapsto 0\}$.

Definition 20 (Derivation)**.** Let e be an axiom. We write $\tau \overset{e}{\sim} \tau'$, and say that τ' is *derived from* τ *by* e, if τ' arises from τ by replacing one occurrence of a subformula γ of τ by a formula γ' such that the logic equation $\gamma = \gamma'$ is an instantiation of e. For a set $\mathcal{E}$ of axioms, we write $\tau \overset{\mathcal{E}}{\sim} \tau'$ if there is a finite sequence $\tau = \tau_1, \ldots, \tau_T = \tau'$ with $\tau_t \overset{e_t}{\sim} \tau_{t+1}$ and $e_t \in \mathcal{E}$, $t = 1, \ldots, T-1$.

Equivalence of $\overset{\mathcal{E}}{\sim}$ follows by taking $T = 1$ for reflexivity, reversing the sequence for symmetry, and concatenating sequences for transitivity.

Taking $\mathcal{E} = \mathcal{MV}$ gives the relation $\overset{\mathcal{MV}}{\sim}$ underlying Theorem 3; the axiom set $\mathcal{E}$ in Definition 20 is left general to account for the DMV and Riesz extensions of Section 6. The rewriting of τ in the example above is a derivation $\tau \overset{\mathcal{E}}{\sim} x_3$ with $\mathcal{E} = \mathcal{MV}$.

4.1 Graph manipulation

We now develop graph derivation in three stages—its definition, the local operations it carries out, and the resulting completeness statement.

Definition 21 (Graph derivation)**.** We lift the relations $\overset{e}{\sim}$ and $\overset{\mathcal{E}}{\sim}$ of Definition 20 to substitution graphs $\mathcal{G}, \mathcal{G}'$ through their represented formulae, writing $\mathcal{G} \overset{e}{\sim} \mathcal{G}'$ if $[\mathcal{G}] \overset{e}{\sim} [\mathcal{G}']$ and $\mathcal{G} \overset{\mathcal{E}}{\sim} \mathcal{G}'$ if $[\mathcal{G}] \overset{\mathcal{E}}{\sim} [\mathcal{G}']$.

A second pullback carries the relation from graphs to networks.

Definition 22 (Network derivation)**.** We lift the relation $\overset{\mathcal{E}}{\sim}$ of Definition 21 to ReLU networks $\mathcal{N}_1, \mathcal{N}_2$ through their extracted graphs, writing $\mathcal{N}_1 \overset{\mathcal{E}}{\sim} \mathcal{N}_2$ if $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{E}}{\sim} \mathtt{ext}(\mathcal{N}_2)$.

Chang's theorem (Theorem 2) carries over to graphs as follows.

Lemma 5. *Let $\mathcal{G}$ and $\mathcal{G}'$ be substitution graphs with the same number of input nodes. If $[\mathcal{G}]^{\mathbb{I}} = [\mathcal{G}']^{\mathbb{I}}$, then $\mathcal{G} \overset{\mathcal{MV}}{\sim} \mathcal{G}'$.*

Proof Chang's theorem (Theorem 2) turns the hypothesis $[\mathcal{G}]^{\mathbb{I}} = [\mathcal{G}']^{\mathbb{I}}$ into $[\mathcal{G}] \overset{\mathcal{MV}}{\sim} [\mathcal{G}']$, which is $\mathcal{G} \overset{\mathcal{MV}}{\sim} \mathcal{G}'$ by Definition 21. ■

Definition 21 relates the represented formulae only, leaving the derivation on the graph invisible. The following example identifies the local operations such a derivation requires.

Example 5 (A derivation carried out on graphs)**.** The networks $\mathcal{N}_3$ and $\mathcal{N}_4$ in Figure 13, both realizing $\max\{x_1, x_2\}$, extract to the respective formulae $\tau_3 = x_2 \oplus (x_1 \odot \neg x_2)$ and $\tau_4 = x_1 \oplus \neg(\neg x_2 \oplus x_1)$, connected by the derivation

$$\begin{aligned} x_2 \oplus (x_1 \odot \neg x_2) &\overset{\text{Ax. 1}}{=} (x_1 \odot \neg x_2) \oplus x_2 \overset{\text{Ax. 9}}{=} (x_2 \odot \neg x_1) \oplus x_1 \\ &\overset{\text{Ax. 7}}{=} (\neg\neg x_2 \odot \neg x_1) \oplus x_1 \overset{\text{Ax. 6}}{=} \neg(\neg x_2 \oplus x_1) \oplus x_1 \overset{\text{Ax. 1}}{=} x_1 \oplus \neg(\neg x_2 \oplus x_1). \end{aligned} \tag{13}$$

We now show how this derivation can be carried out on the substitution graph $\mathtt{ext}(\mathcal{N}_3)$ (Figure 14). To this end we first derive the node formulae of $\mathtt{ext}(\mathcal{N}_3)$. Its left hidden node computes $\rho(x_1 - x_2)$, which on $[0,1]^2$ equals $\sigma(x_1 - x_2)$, so Extraction, Step II (Section 2.3) assigns it the formula $x_1 \odot \neg x_2$. The right hidden node computes $\rho(x_2)$, which equals $\sigma(x_2)$ on $[0,1]^2$, and carries the formula x_2. The output node applies no activation. Extraction, Step I, which converts the ρ-network into a σ-network, applies σ at the output node as well, without effect since $\langle \mathcal{N}_3 \rangle = \max\{x_1, x_2\}$ never exceeds 1. The output node therefore computes $\sigma(x_1 + x_2)$ in the variables supplied by the hidden layer and carries the formula $x_2 \oplus x_1$.

The starting graph $\mathtt{ext}(\mathcal{N}_3)$ (Figure 14) is normal, each of the three node formulae just derived being a minterm. The first step of (13), by Ax. 1, simply exchanges the two arguments of the outermost $\oplus$, replacing the output node's formula $x_2 \oplus x_1$ by $x_1 \oplus x_2$, which is again the extract of $\sigma(x_1 + x_2)$ and hence a minterm, so that the resulting substitution graph is normal.

After this first step the graph represents $(x_1 \odot \neg x_2) \oplus x_2$. The second step of (13), by Ax. 9, rewrites this formula in its entirety and, unlike the first, is no node rewrite, since no single node carries the string being rewritten. Concretely, the formula $(x_1 \odot \neg x_2) \oplus x_2$ is

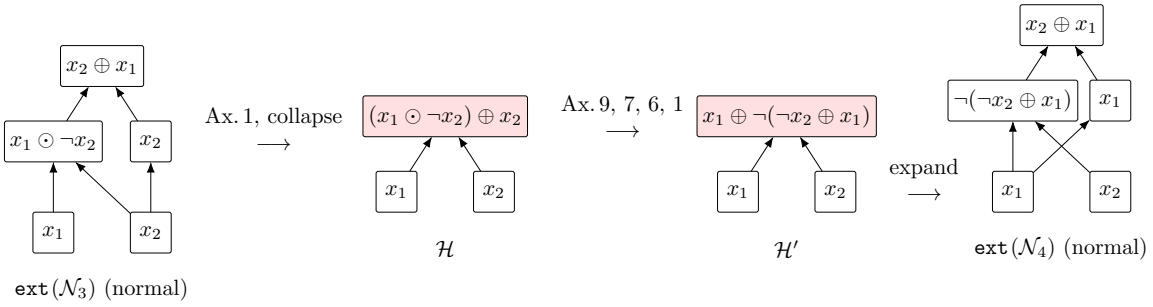

Figure 14: The derivation of Example 5 carried out on graphs. The endpoints $\texttt{ext}(\mathcal{N}_3)$ and $\texttt{ext}(\mathcal{N}_4)$ are normal and hence correspond to ReLU networks. The first arrow combines the Ax. 1 rewrite at the output node with the layer collapse needed to prepare the second step.

spread over two levels in the graph, its subformula $x_1 \odot \neg x_2$ sitting at a hidden node and its outermost $\oplus$ at the output node. Collapsing the hidden layer, by inserting the hidden node formulae into the output node's formula, yields the depth-one substitution graph $\mathcal{H}$, whose single node carries $(x_1 \odot \neg x_2) \oplus x_2$ and at which the application of Ax. 9 becomes a node rewrite. The resulting node formula $(x_1 \odot \neg x_2) \oplus x_2$, however, is no minterm, so $\mathcal{H}$ is not normal. This is seen as follows. The truth function $\max\{x_1, x_2\}$ of this formula has a gradient taking on two distinct nonzero values, namely $(1, 0)$ on $\{x_1 > x_2\}$ and $(0, 1)$ on $\{x_2 > x_1\}$, while the gradient of $\sigma(m_1 x_1 + \cdots + m_n x_n + b)$ (Definition 15) can only have the single nonzero value $(m_1, \ldots, m_n)$. Passage through non-normal graphs is the price of applying axioms that relate formulae extending over several layers, since such formulae can be confined to a single node only by collapsing those layers, and the composite formula so obtained need not be a minterm.

With the entire formula now confined to a single node, the second to fifth steps of (13) are node rewrites throughout and carry $\mathcal{H}$ to $\mathcal{H}'$ on a fixed layered graph, through $(x_2 \odot \neg x_1) \oplus x_1$, $(\neg\neg x_2 \odot \neg x_1) \oplus x_1$, and $\neg(\neg x_2 \oplus x_1) \oplus x_1$. Each of these five depth-one substitution graphs is not normal, as its single node has truth function $\max\{x_1, x_2\}$ and therefore does not carry a minterm. The endpoint of the derivation must again be a normal substitution graph, so that `constr` can be applied to deliver a ReLU network. To this end, we expand $\mathcal{H}'$ into two levels, which returns $\texttt{ext}(\mathcal{N}_4)$, again normal, its node formulae $\neg(\neg x_2 \oplus x_1)$, x_1, and $x_2 \oplus x_1$ being the extracts of $\sigma(x_2 - x_1)$, $\sigma(x_1)$, and $\sigma(x_1 + x_2)$.

We now formalize the three operations—node rewrites, layer collapses, and layer expansions—and show in Proposition 4 that they suffice to carry out every graph derivation.

Definition 23 (Node rewrite)**.** Let $\mathcal{G}$ be a substitution graph, $\mathcal{E}$ a set of axioms, $e \in \mathcal{E}$, and v a non-input node of $\mathcal{G}$ at level j. A *node rewrite of v by e* replaces the formula $[v]$ by a formula τ' in the variables $x_1, \ldots, x_{d_{j-1}}$ with $[v] \overset{e}{\sim} \tau'$, leaving (V, E) and all other node formulae unchanged. A *node rewrite by $\mathcal{E}$* is a node rewrite by some $e \in \mathcal{E}$.

While a node rewrite leaves the local map of v unchanged, it need not preserve normality of the substitution graph, as the rewritten formula need not be a minterm. To see this,

consider a node carrying the minterm x_1 and rewrite it into $\neg\neg x_1$ by Ax. 7, which leaves the truth function $\sigma(x_1)$ unchanged. A minterm with this truth function must be of the form $\texttt{EXTR}(\sigma(g))$ with $g = m_1x_1 + \cdots + m_nx_n + b$ an integer affine map and $\sigma(g) = \sigma(x_1)$. Now the points $x \in [0,1]^n$ with $0 < x_1 < 1$ form a non-empty open set, on which $\sigma(x_1) = x_1 \in (0,1)$ and hence $\sigma(g)$ takes values in $(0,1)$ as well. Fix an x in this set. As $\sigma(t)$ lies in $(0,1)$ only for $t \in (0,1)$, it follows that $g(x) \in (0,1)$, and σ acting as the identity on $(0,1)$ gives $g(x) = \sigma(g(x)) = x_1$. The affine maps $g(x)$ and x_1 thus agree on an open set and must therefore be equal everywhere. Applying Lemma 1 with $m_1 = 1$, so that $f_\circ = 0$, yields $\texttt{EXTR}(\sigma(x_1)) = (\sigma(0) \oplus x_1) \odot \sigma(1) = x_1$. The only minterm with truth function $\sigma(x_1)$ is hence x_1.

Definition 24 (Layer collapse and expansion)**.** Let $\mathcal{G}$ be a substitution graph of depth L. For $L \geq 2$ and $k \in \{1, \ldots, L-1\}$, *collapsing* layer k removes the level-k nodes, joins levels $k-1$ and $k+1$, and replaces the formula $[v]$ of every level-$(k+1)$ node by $[v](\zeta^{(k,k+1)})$, which is a formula in the level-$(k-1)$ variables because the substitutors of $\zeta^{(k,k+1)}$, the level-k node formulae of $\mathcal{G}$, are. For $k \in \{1, \ldots, L\}$, *expanding* at level k inserts a new level of $m \geq 1$ nodes between levels $k-1$ and k, shifting every level from k on up by one, so that the depth becomes $L+1$. The inserted nodes, which now constitute level k, carry formulae $\chi_1, \ldots, \chi_m$ in the level-$(k-1)$ variables and supply the new variables $x_1, \ldots, x_m$ to the level above, and the formula $[v]$ of every level-$(k+1)$ node of the expanded graph is replaced by a formula $[v]'$ in these variables according to $[v]'(\{x_1 \mapsto \chi_1, \ldots, x_m \mapsto \chi_m\}) = [v]$. While collapse is determined by $\mathcal{G}$ and k, expansion involves the choice of m, the χ_i, and the $[v]'$; collapsing layer k is undone by taking, all with reference to $\mathcal{G}$ before the collapse, $m = d_k$, $\chi_{i'} = [v_{i'}^{(k)}]$, $i' = 1, \ldots, d_k$, and $[v]'$ the formula of the level-$(k+1)$ node in question, but other choices build layers unrelated to any collapse. In both cases the result is again a substitution graph.

Layer collapse and expansion have no formula-level counterpart; they modify the graph $\mathcal{G}$ without affecting the represented formula $[\mathcal{G}]$ (Lemmata 9 and 10, both in Appendix D). A node rewrite by an axiom $e \in \mathcal{E}$, in contrast, alters the represented formula, but only within its $\mathcal{E}$-derivation class, $[\mathcal{G}] \overset{\mathcal{E}}{\sim} [\mathcal{G}']$ (Proposition 6, in Appendix D). A single node rewrite may take more than one derivation step at the level of the represented formula, which is why the relation here is $\overset{\mathcal{E}}{\sim}$ and not $\overset{e}{\sim}$. This can be seen as follows. Flattening $\mathcal{G}$ substitutes the formula of a given node into every child referring to it, so $[v]$ may occur repeatedly in $[\mathcal{G}]$, while Definition 20 replaces one occurrence of $[v]$ at a time. The following example illustrates this. Take a graph of depth two whose single hidden node v carries the formula χ and whose output node carries $x_1 \oplus x_1$, so that $[\mathcal{G}] = \chi \oplus \chi$. A rewrite of v turning χ into χ' produces $[\mathcal{G}'] = \chi' \oplus \chi'$, two derivation steps away from $[\mathcal{G}]$. Whichever of the three operations—node rewrite, layer collapse, layer expansion—is applied, the resulting graph $\mathcal{G}'$ satisfies $\mathcal{G} \overset{\mathcal{E}}{\sim} \mathcal{G}'$ (Definition 21).

We are now ready to state the main result of this section, namely that every graph derivation can be carried out by node rewrites, layer collapses, and layer expansions.

Proposition 4 (Graph derivation by local operations)**.** *Let $\mathcal{E}$ be a set of axioms and let $\mathcal{G}$, $\mathcal{G}'$ be substitution graphs with the same number of input nodes. If $\mathcal{G} \overset{\mathcal{E}}{\sim} \mathcal{G}'$, then $\mathcal{G}$ and $\mathcal{G}'$*

are connected by a finite sequence of layer collapses, layer expansions (Definition 24), and node rewrites by $\mathcal{E}$ (Definition 23).

Proof Since $\mathcal{G} \overset{\mathcal{E}}{\sim} \mathcal{G}'$, Definition 21 gives $[\mathcal{G}] \overset{\mathcal{E}}{\sim} [\mathcal{G}']$, and Definition 20 asserts the existence of a corresponding derivation $[\mathcal{G}] = \tau_1 \overset{e_1}{\sim} \tau_2 \overset{e_2}{\sim} \cdots \overset{e_{T-1}}{\sim} \tau_T = [\mathcal{G}']$ with $e_t \in \mathcal{E}$, $t = 1, \ldots, T-1$. Let d_0 denote the common number of input nodes of $\mathcal{G}$ and $\mathcal{G}'$. The formulae τ_t may all be assumed to have their variables among $x_1, \ldots, x_{d_0}$. Indeed, an axiom instantiation can introduce variables absent from both $[\mathcal{G}]$ and $[\mathcal{G}']$, as when Ax. 4, $x \oplus 1 = 1$, is applied from right to left and replaces 1 by $y \oplus 1$ for such a variable y. Replacing every new variable by, for concreteness, x_1 leaves the endpoints unchanged and preserves each derivation step (Lemma 12, in Appendix D), yielding an end-to-end derivation with all formulae in $x_1, \ldots, x_{d_0}$.

Now flatten $\mathcal{G}$ by collapsing its hidden layers (Lemma 9), obtaining a graph of depth one whose single non-input node carries the formula $[\mathcal{G}] = \tau_1$ over the d_0 input nodes. Next, construct the depth-one graphs H_t, $t = 1, \ldots, T$, whose only non-input node carries τ_t over the d_0 input nodes; flattening $\mathcal{G}$ produces H_1. The step $\tau_t \overset{e_t}{\sim} \tau_{t+1}$ can now be read as a graph operation, since in a depth-one graph the represented formula is the formula of the single non-input node; it is a rewrite of that node by e_t, carrying H_t to H_{t+1}. Flattening $\mathcal{G}'$ likewise produces H_T. Finally, reversing the collapses that flatten $\mathcal{G}'$ turns them into expansions leading from H_T back to $\mathcal{G}'$, each collapse being undone by the corresponding expansion (Definition 24), so that $\mathcal{G}'$ itself is recovered. We thus pass from $\mathcal{G}$ to H_1 by collapses, from H_1 to H_T by single-node rewrites, and from H_T to $\mathcal{G}'$ by expansions. ■

We now state the graph counterpart of Theorem 2.

Theorem 4 (Graph-level completeness)**.** Let $\mathcal{G}$ and $\mathcal{G}'$ be substitution graphs with the same number of input nodes. Then $[\mathcal{G}]^{\mathbb{I}} = [\mathcal{G}']^{\mathbb{I}}$ if and only if $\mathcal{G}$ and $\mathcal{G}'$ are connected by a finite sequence of layer collapses, layer expansions (Definition 24), and node rewrites by $\mathcal{MV}$ (Definition 23).

Proof If $[\mathcal{G}]^{\mathbb{I}} = [\mathcal{G}']^{\mathbb{I}}$, then Lemma 5 gives $\mathcal{G} \overset{\mathcal{MV}}{\sim} \mathcal{G}'$, and Proposition 4 exhibits the associated sequence of layer collapses, layer expansions, and node rewrites. Conversely, layer collapse and layer expansion leave the formula represented by the graph unchanged (Lemmata 9 and 10), while a node rewrite by an axiom of $\mathcal{MV}$ gives $[\mathcal{G}] \overset{\mathcal{MV}}{\sim} [\mathcal{G}']$ (Proposition 6). Each operation therefore satisfies $[\mathcal{G}] \overset{\mathcal{MV}}{\sim} [\mathcal{G}']$, and hence we get $[\mathcal{G}]^{\mathbb{I}} = [\mathcal{G}']^{\mathbb{I}}$ overall. ■

5 Constructing ReLU networks from normal substitution graphs

This section establishes the construction step of the identification program behind Theorem 3. The construction algorithm developed here builds, from any normal substitution graph $\mathcal{G}$, a ReLU network $\texttt{constr}(\mathcal{G})$ with $\langle \texttt{constr}(\mathcal{G}) \rangle = [\mathcal{G}]^{\mathbb{I}}$ and $\texttt{constr}(\texttt{ext}(\mathcal{N})) = \mathcal{N}$ for every $\mathcal{N} \in \mathfrak{N}$, that is, `constr` is a left inverse of `ext` on $\mathfrak{N}$. Left-invertibility is what renders the normal substitution graph $\texttt{ext}(\mathcal{N})$ a faithful representation of $\mathcal{N}$. It also makes `ext` injective on $\mathfrak{N}$, as $\texttt{ext}(\mathcal{N}_1) = \texttt{ext}(\mathcal{N}_2)$ implies $\mathcal{N}_1 = \texttt{constr}(\texttt{ext}(\mathcal{N}_1)) =$

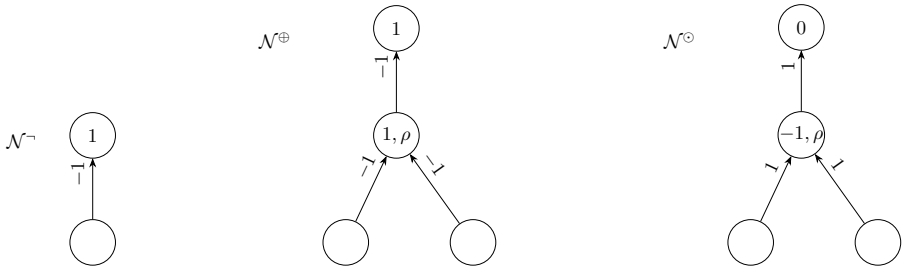

Figure 15: The elementary building blocks of the construction algorithm in [22]. As in Figure 10, a hidden node is labeled by its bias and activation function, an edge by its weight, and the output node, which applies no activation function, by its bias alone. The three networks thus compute $1-x_1 = \neg x_1$, $1-\rho(1-x_1-x_2) = x_1 \oplus x_2$, and $\rho(x_1+x_2-1) = x_1 \odot x_2$, respectively.

$\texttt{constr}(\texttt{ext}(\mathcal{N}_2)) = \mathcal{N}_2$. Distinct networks in $\mathfrak{N}$ therefore have distinct substitution graphs, and a classification carried out on normal substitution graphs is a classification of networks.

We build on the construction algorithm of [22], which takes a Łukasiewicz formula as input and builds a ReLU network by concatenating three elementary networks $\mathcal{N}^{\neg}$, $\mathcal{N}^{\oplus}$, $\mathcal{N}^{\odot}$ (Figure 15), realizing the operations $\neg$, $\oplus$, and $\odot$, according to how they appear in the formula. The architecture of the network so obtained is dictated by the syntax of the formula and bears no relation to the architecture of a network the formula may have been extracted from.

5.1 Construction, Step I: From normal substitution graphs to σ-networks

Let $\mathcal{G} = (V, E, \mathcal{C})$ be a normal substitution graph of depth L and architecture $(d_0, \dots, d_L)$. The network we construct inherits its nodes and edges from the layered graph (V, E) of $\mathcal{G}$, so that its architecture is $(d_0, \dots, d_L)$. It remains to specify weights, biases, and activation functions. For each node $v_i^{(j)}$ of $\mathcal{G}$ with $j \geq 1$, apply the κ-mapping (Definition 16) to the Łukasiewicz formula $[v_i^{(j)}] \in \mathcal{C}_{\text{norm}}$. The result takes one of the two forms

$$\kappa([v_i^{(j)}]) = \sigma\Big(b_{v_i^{(j)}} + \sum_{i'} m_{i'} x_{i'}\Big), \quad m_{i'}, b_{v_i^{(j)}} \in \mathbb{Z}, \tag{14}$$

$$\kappa([v_i^{(j)}]) = c, \quad c \in \{0, 1\}. \tag{15}$$

The affine combination inside the right-hand side of (14) is uniquely determined by $\kappa([v_i^{(j)}])$ whenever some $m_{i'}$ is nonzero, equivalently, whenever $\kappa([v_i^{(j)}])$ is non-constant. Indeed, a non-constant affine combination takes values in $(0, 1)$ on a non-empty open set, on which σ acts as the identity; two combinations with equal σ-image therefore agree there, and two affine maps agreeing on an open set are equal. In case (14) we can hence identify the weight on the edge $(v_{i'}^{(j-1)}, v_i^{(j)})$ with $m_{i'}$, for $i' = 1, \dots, d_{j-1}$, and the bias at $v_i^{(j)}$ with $b_{v_i^{(j)}}$. If all $m_{i'}$ vanish, the bias is not uniquely determined, being an integer and $\sigma(b)$ taking the value 1 for every $b \geq 1$ and the value 0 for every $b \leq 0$, and (15) fixes it to the corresponding value

c; every edge entering $v_i^{(j)}$ then receives the weight 0 and the node the bias c. All non-input nodes, the output node included, are endowed with the activation function σ (Definition 9). Every node v of the resulting σ-network $\mathcal{M} = (V, E, \mathcal{W}, \mathcal{B}, \Psi)$ (Definition 9) has local map $\langle v \rangle = \kappa([v])$. For $j \geq 1$ this holds by construction, the weights, the bias, and the activation function of v having been chosen so that $\langle v \rangle$ is the right-hand side of (14), respectively the constant $\sigma(c) = c$ of (15); at an input node it holds since $[v_i^{(0)}] = x_i$. Moreover, by Definition 16, $\kappa([v]) = [v]^{\mathbb{I}}$, so $\mathcal{M}$ realizes $[\mathcal{G}]^{\mathbb{I}}$ on $[0,1]^{d_0}$ by Lemma 3.

5.2 Construction, Step II: From σ-networks to ρ-networks

This step converts the σ-network $\mathcal{M}$ delivered by Construction, Step I into a functionally equivalent ρ-network. The conversion rests on the identities

$$\sigma(x) = \rho(x), \quad x \leq 1, \tag{16}$$

$$\sigma(x) = \rho(x) - \rho(x-1), \quad x \in \mathbb{R}. \tag{17}$$

We proceed through the hidden layers from the output side toward the input side, that is, in the order $j = L-1, \ldots, 1$; the output node is treated at the end of this step. The parents of a node are then still σ-nodes when its input interval (6) is computed, so that the parent outputs lie in $[0,1]$, as (6) requires. Extraction, Step I faces the same requirement and meets it by the reverse order, the parents of a ρ-node being converted to σ-nodes before the node itself is converted. Each node $v_i^{(j)}$ with input interval $[\ell_{v_i^{(j)}}, \mathcal{L}_{v_i^{(j)}}]$, computed as in (6), is treated as follows:

- if $\mathcal{L}_{v_i^{(j)}} \leq 1$: replace the activation function by ρ, which leaves the local map unchanged, according to (16);
- if $\mathcal{L}_{v_i^{(j)}} > 1$: replace the activation function by ρ and add a companion node $v_{i1}^{(j)}$ to layer j, joined to the parents and children of $v_i^{(j)}$, with the same incoming weights, outgoing weights negated, and bias $b_{v_i^{(j)}} - 1$; the contributions of $v_i^{(j)}$ and $v_{i1}^{(j)}$ to their children then sum to the output of the original σ-node, according to (17).

Once a layer has been converted, we *aggregate* its ρ-nodes, merging those of equal local map into one, with outgoing weights added, and removing every node of the layer whose outgoing weights are then all zero. Aggregation does not change the realized function. It removes the degeneracies excluded by the third and fourth conditions of Definition 14, and thereby ensures that, for $\mathcal{G} = \mathtt{ext}(\mathcal{N})$, Construction, Step II returns $\mathcal{N}$ itself rather than a functionally equivalent expansion of it.

It remains to treat the output node v_{out}, which, in accordance with Definition 9, was endowed with the activation function σ in Construction, Step I. For $\mathcal{G} = \mathtt{ext}(\mathcal{N})$ the pre-activation of v_{out} equals $\langle \mathcal{N} \rangle$ and hence lies in $[0,1]$, so that σ acts as the identity and can be removed, the architecture remaining that of $\mathcal{G}$. For a general normal graph we have to determine whether the pre-activation g of v_{out} lies in $[0,1]$. If it does, σ is again removed; if not, a layer has to be inserted, σ being realized through (17) by turning v_{out} and a companion node with bias $b_{v_{\text{out}}} - 1$ into hidden ρ-nodes that feed a new output node

with weights 1 and -1 and bias 0. As the interval bounds (6) may overestimate the true range of g and could hence insert the additional layer needlessly, the condition $0 \leq g \leq 1$ on $[0,1]^{d_0}$ has to be decided exactly. At this point the network feeding v_{out} carries ρ-nodes in all hidden layers, so that the condition is an input-output property of a network realizing a piecewise-linear function. We can hence employ the Branch-and-Bound framework of [5], which decides such properties by branching over the input domain or over the phase of a ReLU node, active or inactive, and bounding the output on each resulting subproblem through a convex relaxation.

We now show that Construction, Step II undoes Extraction, Step I on non-degenerate networks.

Lemma 6. *Let $\mathcal{N} \in \mathfrak{N}$, let $\mathcal{M}$ be the σ-network obtained from $\mathcal{N}$ by Extraction, Step I, and let $\mathcal{N}'$ be the ρ-network Construction, Step II produces from $\mathcal{M}$. Then $\mathcal{N}' = \mathcal{N}$.*

Proof We track a single hidden level through the two passages, from $\mathcal{N}$ to $\mathcal{M}$ by Extraction, Step I and from $\mathcal{M}$ to $\mathcal{N}'$ by Construction, Step II. The input nodes are left untouched by both passages, and the output node is dealt with at the end. Arbitrarily fix a hidden level j, $j \in \{1, \ldots, L-1\}$, and a node u of $\mathcal{N}$ at that level, with local map $\rho(f_u)$, where f_u denotes the affine combination feeding u, input interval $[\ell_u, \mathcal{L}_u]$ (according to (6)), and outgoing weights $w_u = \{w_{\tilde{v},u} : (u,\tilde{v}) \in E\}$, not all zero by non-degeneracy. Extraction, Step I replaces u by the cluster of σ-nodes $\sigma(f_u - i)$, $0 \leq i \leq k_u - 1$, where $k_u = \lceil \mathcal{L}_u \rceil$ if $\mathcal{L}_u > 1$ and $k_u = 1$ otherwise, each carrying the incoming weights of u and the outgoing weights w_u. Extraction, Step I and Construction, Step II both evaluate (6) at u with the parents of u outputting values in $[0,1]$, so that the input interval of the cluster member $\sigma(f_u - i)$ has upper end $\mathcal{L}_u - i$. A companion is added to $\sigma(f_u - i)$, through (17), precisely when $\mathcal{L}_u - i > 1$, that is, as $k_u - 1 < \mathcal{L}_u \leq k_u$, for $i \leq k_u - 2$. Construction, Step II therefore converts the cluster $\{\sigma(f_u - i)\}_{i=0}^{k_u - 1}$ into ρ-nodes of $\mathcal{N}'$, all carrying the incoming weights of u, namely

$$\rho(f_u),\ \rho(f_u - 1),\ \ldots,\ \rho(f_u - k_u + 1), \qquad \text{all with outgoing weights } w_u,$$

and the companions

$$\rho(f_u - 1),\ \ldots,\ \rho(f_u - k_u + 1), \qquad \text{all with outgoing weights } -w_u.$$

The two lists differ only in the entry $\rho(f_u)$, so that the outgoing weights of the nodes originating from u sum to w_u at the local map $\rho(f_u)$ and to zero at $\rho(f_u - i)$, $1 \leq i \leq k_u - 1$. By the third condition of Definition 14 the maps f_u are pairwise distinct, so that aggregation leaves a unique node with local map $\rho(f_u)$ and outgoing weights w_u for each level-j node u of $\mathcal{N}$, and removes all others, their weights summing to zero. Altogether, level j of $\mathcal{N}'$ equals level j of $\mathcal{N}$ and hence, by virtue of $j \in \{1, \ldots, L-1\}$ being arbitrary, $\mathcal{N}$ and $\mathcal{N}'$ agree at every hidden level. At the output node of $\mathcal{M}$ the pre-activation equals $\langle \mathcal{N} \rangle \in [0,1]$, so that σ can be removed without changing the output map. Hence $\mathcal{N}' = \mathcal{N}$. ■

Example 6. We illustrate Lemma 6 by applying Construction, Step II to the σ-network $\mathcal{M}$ of Figure 9, which Extraction, Step I produced from the ρ-network $\mathcal{N}$ of Figure 7. The hidden layers are treated in the order $j = 3, 2, 1$, followed by the output node, and $\mathcal{L}$ denotes

throughout the upper end of the input interval (6). At level 3 the node $v_1^{(3)}$ has $\mathcal{L} = 1$, so that its activation function is merely replaced by ρ. At level 2 the nodes $v_1^{(2)}, v_{11}^{(2)}, v_2^{(2)}$ have $\mathcal{L} = 2, 1, 1$, so that only $v_1^{(2)}$ receives a companion—a new node, not displayed in Figure 9—of bias 0 and outgoing weight $+1$, the outgoing weight of $v_1^{(2)}$ being -1. This companion and $v_{11}^{(2)}$ carry the same local map and outgoing weights of opposite sign, so that aggregation merges them into a node of weight zero and removes it, leaving level 2 with $v_1^{(2)}$ and $v_2^{(2)}$, both now ρ-nodes. We proceed in the same way at level 1. The removal at level 2 has deleted the edges into $v_{11}^{(2)}$, so that $v_1^{(1)}$ and $v_{11}^{(1)}$ are each left with the two outgoing weights -1 and $+1$, into $v_1^{(2)}$ and $v_2^{(2)}$. Of $v_1^{(1)}, v_{11}^{(1)}, v_2^{(1)}$, having $\mathcal{L} = 2, 1, 1$, only $v_1^{(1)}$ receives a companion, of bias -2 and outgoing weights $+1$ and -1, the negatives of those of $v_{11}^{(1)}$ in each component, so that this companion and $v_{11}^{(1)}$ cancel. At the output node the pre-activation equals $\langle \mathcal{N} \rangle$, the conversion of the hidden layers having left the input-output map unchanged, and takes values in $[0, 1]$, so that σ can be dropped. The result is the network $\mathcal{N}$ of Figure 7.

5.3 Construction is a left inverse of extraction

Proposition 5. *For every $\mathcal{N} \in \mathfrak{N}$, $\mathtt{constr}(\mathtt{ext}(\mathcal{N})) = \mathcal{N}$.*

Proof Construction is the composition of its two steps, so that the assertion concerns the round trip

$$\mathcal{N} \xrightarrow{\text{Extr. I}} \mathcal{M} \xrightarrow{\text{Extr. II}} \mathtt{ext}(\mathcal{N}) \xrightarrow{\text{Constr. I}} \mathcal{M}' \xrightarrow{\text{Constr. II}} \mathcal{N}'. \tag{18}$$

Lemma 6 supplies the last arrow, but only for the σ-network Extraction, Step I produced from $\mathcal{N}$. It hence remains to show that $\mathcal{M}' = \mathcal{M}$.

Extraction, Step II retains of $\mathcal{M}$ nothing but its layered graph and one Łukasiewicz formula per node, and Construction, Step I has to recover the weights and biases of $\mathcal{M}$ from these formulae. As the node formulae of $\mathtt{ext}(\mathcal{N})$ are outputs of Extraction, Step II, they are minterms, so that $\mathtt{ext}(\mathcal{N})$ is normal (Definition 15), its layered graph is that of $\mathcal{M}$ (Definition 13), and $\kappa([v]) = \langle v \rangle$ for every node v (Lemma 4).

We now argue that Construction, Step I recovers, at every non-input node v, the node's weights and bias from $\kappa([v])$. If $\langle v \rangle$ is non-constant, the weights and the bias can be read from $\kappa([v])$, as shown in Section 5.1, and v is assigned in $\mathcal{M}'$ the incoming weights and the bias it carries in $\mathcal{M}$. If $\langle v \rangle$ is constant, all $m_{i'}$ vanish, hence so do the incoming weights of v in $\mathcal{M}$, and Construction, Step I reproduces them correctly, but sets the bias to $c \in \{0, 1\}$, which differs from the bias of v in $\mathcal{M}$ whenever the latter is > 1 or < 0. Only a node with constant local map whose bias in $\mathcal{M}$ differs from c can hence obstruct $\mathcal{M}' = \mathcal{M}$. We next show that non-degeneracy excludes the existence of such nodes. A hidden node u of $\mathcal{N}$ is active at some input and inactive at another (first and second conditions of Definition 14), so that the affine combination f_u feeding u is non-constant, and with it the local maps $\sigma(f_u - i)$ of the cluster members emerging from u. At the output node, if $\mathcal{N}$ is deep, the local map is non-constant as well, since every node at the last hidden level carries a nonzero weight into v_{out}, its only child (fourth condition). A constant local map is thus confined to shallow $\mathcal{N}$ with vanishing input weights—a network that lies in $\mathfrak{N}$, the first, second, and fourth conditions holding vacuously because there are no hidden nodes and the third since

the input nodes carry the distinct local maps $x_1, \ldots, x_n$. In this case $\langle \mathcal{N} \rangle$ is the bias at v_{out}, an integer equal to 0 or 1 as $\mathcal{N}$ has integer weights and biases and maps into $[0, 1]$; (15) assigns exactly this value to the bias of v_{out}, so that $\mathcal{M}' = \mathcal{M}$. ■

Proposition 5 completes the construction step. On $\mathfrak{N}$, construction inverts extraction, so a non-degenerate network is recovered from, and uniquely encoded by, its normal form.

6 Extensions

We now show how Theorem 3 extends to three further settings, namely, a finite input domain, rational network weights and biases, and real network weights and biases. Trained networks rarely have integer weights and biases, unless integrality is imposed during training, which is what motivates the latter two settings. The first is relevant when a network can be probed only at finitely many inputs, as in fixed-point implementations, where identification is based on finitely many function values.

Rational and real weights and biases enlarge the class of networks and with it the class of realized functions. The correspondence underlying the entire development in this paper—the truth functions of the logic being exactly the functions realized by the networks—then forces a corresponding enlargement of Łukasiewicz logic, by division operators for rational weights and biases and by scalar multiplication for real ones. The finite-input-domain case, in contrast, leaves the class of networks under consideration, that of Theorem 3 with its integer weights and biases, untouched and changes the semantics instead. The truth values are reduced from $[0, 1]$ to a finite set, the standard MV algebra is replaced by a finite-valued one, and functional equivalence becomes coarser, so that more networks count as equivalent and, correspondingly, more axioms are available to connect them.

In each of the three settings considered, only those of the steps (i)–(iii) of Section 1 that require adjustment are treated in detail, with the remaining ones carrying over unchanged. In each setting the graph-level relation is obtained from the formula-level one as in Definition 21, and the network-level relation as in Definition 22, with `ext` the extraction and $\mathcal{E}$ the axiom set of the setting under consideration. The non-degeneracy condition (Definition 14) is insensitive to the ring the weights and biases are drawn from and is imposed throughout without further mention. Lemma 2 and (9) are proved by induction on the structure of the formula. The operations added to the language below are all unary, each adding to that induction a case carried out as for $\neg$, so that both results hold in the extended languages.

6.1 Finite input domain

For $k \in \mathbb{N}$, restricting the input domain from $[0, 1]^n$ to the finite grid I_k^n of resolution k, with $I_k = \{0, 1/k, \ldots, (k-1)/k, 1\}$, weakens functional equivalence, in the sense that two networks may agree on I_k^n but differ elsewhere on $[0, 1]^n$. The uniform spacing of I_k is not a matter of convenience, a finite subset of $[0, 1]$ that contains 0 and 1 and is closed under $\oplus$, $\odot$, and $\neg$ being necessarily of this form; the I_k are thus precisely the finite subalgebras of $\mathbb{I}$. The network identification problem is correspondingly richer, and the axiom set reflects this, the axioms of $\mathcal{MV}$ being augmented by those of Definition 25 below, which render formulae

interderivable that are not interderivable in $\mathcal{MV}$. For instance, $x \oplus x$ and x agree on $\{0,1\}$ but not on $[0,1]$, and are hence interderivable at $k=1$ but not in $\mathcal{MV}$.

The finite case requires $(k+1)$-valued Łukasiewicz logic $\mathrm{Ł}_k$, defined on the truth values I_k, whose algebraic counterpart is the $(k+1)$-valued MV algebra defined next.

Definition 25. [14]; see also [8, Cor. 8.2.4] for $k \in \{1,2\}$ and [8, Thm. 8.5.1] for $k \geq 3$. We define $\oplus^j x$ and $\odot^j x$ as the j-fold disjunction $x \oplus \cdots \oplus x$ and the j-fold conjunction $x \odot \cdots \odot x$, respectively. For $k=1$, a 2-valued MV algebra is an MV algebra (Definition 4) satisfying the additional axiom $x \oplus x = x$. For $k \geq 2$, a $(k+1)$-valued MV algebra is an MV algebra satisfying the additional axioms

$$\begin{aligned} \oplus^{k+1} x &= \oplus^k x, \\ \odot^{k+1}\big(\oplus^j(\odot^{j-1}x)\big) &= \oplus^{k+1}(\odot^j x), \quad \text{for every } j \in \{2,\dots,k-1\} \text{ that does not divide } k. \end{aligned}$$

We write $\mathcal{MV}_k$ for the axioms of $\mathcal{MV}$ together with the additional axioms stated in this definition, which lead to the grid symmetries Ax. $\mathcal{MV}_k$ and Ax. $\mathcal{MV}_{k,j}$ of Appendix A.

Definition 26. For $k \in \mathbb{N}$, the standard $(k+1)$-valued MV algebra is $\mathcal{I}_k = (I_k, \oplus, \odot, \neg, 0, 1)$ with operations $x \oplus y = \min\{1, x+y\}$, $x \odot y = \max\{0, x+y-1\}$, $\neg x = 1-x$ restricted to I_k.

Since I_k contains 0 and 1 and is closed under $\oplus$, $\odot$, and $\neg$, every Łukasiewicz formula takes values in I_k when its variables do.

For $k=1$ we recover the Boolean case. The operations of Definition 26 restricted to $I_1 = \{0,1\}$ are Boolean disjunction, conjunction, and negation, so that $\mathcal{I}_1$ is the two-element Boolean algebra and $\mathrm{Ł}_1$ is Boolean logic. The additional axiom $x \oplus x = x$ of Definition 25 is what turns an MV algebra into a Boolean one [8, Cor. 8.2.4].

Syntactically, $\mathrm{Ł}_k$ and Łukasiewicz logic coincide, being built from the same connectives and hence having the same formulae. Semantically they differ, the truth values being I_k in place of $[0,1]$. Steps (i) and (iii) of Section 1 carry over unchanged. Extraction and construction are algorithms on networks and graphs, hence unaffected by the restriction of the domain, and the identities of Lemma 1 and the equality of Lemma 3 hold pointwise on $[0,1]^n$ and *a fortiori* on I_k^n.

The new ingredient here is step (ii), which follows from the finite-valued completeness theorem of Grigolia [14].

Theorem 5 ([14])**.** Let τ, τ' be formulae in $\mathrm{Ł}_k$ and let $k, n \in \mathbb{N}$. If $\tau^{\mathcal{I}_k} = \tau'^{\mathcal{I}_k}$ on I_k^n, then $\tau \overset{\mathcal{MV}_k}{\sim} \tau'$.

With Theorem 5 supplying step (ii) in place of Chang's completeness theorem (Theorem 2), we obtain the identification result over I_k^n.

Theorem 6. For $n \in \mathbb{N}$, let $\mathfrak{N}$ be the class of non-degenerate ReLU networks with integer weights and biases realizing functions $f : [0,1]^n \to [0,1]$. For $k \in \mathbb{N}$, the set of axioms $\mathcal{MV}_k$ completely identifies $\mathfrak{N}$ over I_k^n, that is, $\mathcal{N}_1 \sim_{I_k^n} \mathcal{N}_2$ implies $\mathcal{N}_1 \overset{\mathcal{MV}_k}{\sim} \mathcal{N}_2$ for all $\mathcal{N}_1, \mathcal{N}_2 \in \mathfrak{N}$.

Proof Step (i) (extraction): set $\tau_i = [\mathtt{ext}(\mathcal{N}_i)]$, $i = 1, 2$. Extraction is unaffected by the domain restriction, and $\tau_i^{\mathcal{I}_k} = \tau_i^{\mathbb{I}} = \langle \mathcal{N}_i \rangle$ on I_k^n because I_k is closed under $\oplus$, $\odot$, and $\neg$. Since $\mathcal{N}_1 \sim_{I_k^n} \mathcal{N}_2$ means $\langle \mathcal{N}_1 \rangle = \langle \mathcal{N}_2 \rangle$ on I_k^n (Section 1), we have $\tau_1^{\mathcal{I}_k} = \tau_2^{\mathcal{I}_k}$. Step (ii) (derivation): Theorem 5 yields $\tau_1 \overset{\mathcal{MV}_k}{\sim} \tau_2$, which is $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{MV}_k}{\sim} \mathtt{ext}(\mathcal{N}_2)$ by Definition 21, and hence $\mathcal{N}_1 \overset{\mathcal{MV}_k}{\sim} \mathcal{N}_2$ by Definition 22. ■

At $k = 1$, Theorem 6 states that the formulae extracted from two networks in $\mathfrak{N}$ agreeing on the Boolean cube $\{0, 1\}^n$ are interderivable through the axioms of Boolean algebra.

Both relations in Theorem 6 become finer as the grid is refined. If k divides k', then $I_k^n \subseteq I_{k'}^n$, so that $I_{k'}^n$ is the finer grid, and $\mathcal{I}_k$ is a subalgebra of $\mathcal{I}_{k'}$. Agreement on $I_{k'}^n$ therefore implies agreement on I_k^n. Likewise $\tau \overset{\mathcal{MV}_{k'}}{\sim} \tau'$ implies $\tau \overset{\mathcal{MV}_k}{\sim} \tau'$. Indeed, derivations preserve truth functions, so that $\tau^{\mathcal{I}_{k'}} = \tau'^{\mathcal{I}_{k'}}$ and hence $\tau^{\mathcal{I}_k} = \tau'^{\mathcal{I}_k}$, which is $\tau \overset{\mathcal{MV}_k}{\sim} \tau'$ by Theorem 5. At $k = 1$ the grid is the Boolean cube $\{0, 1\}^n$, and networks realizing different functions on $[0, 1]^n$ can satisfy $\mathcal{N}_1 \sim_{\{0,1\}^n} \mathcal{N}_2$, Theorem 6 relating them by $\overset{\mathcal{MV}_1}{\sim}$. This leads to the notion of networks agreeing, in the sense of interderivability, up to a given resolution. As $I_k \subseteq I_{k'}$ only for k dividing k', resolutions have to be compared along a chain, and we take the dyadic one, $k = 2^b$, the grid of b-bit fixed-point arithmetic. Two networks agree up to resolution 2^b if $\mathcal{N}_1 \overset{\mathcal{MV}_{2^b}}{\sim} \mathcal{N}_2$ but $\mathcal{N}_1 \overset{\mathcal{MV}_{2^{b+1}}}{\not\sim} \mathcal{N}_2$. Such networks are interderivable in a b-bit fixed-point implementation, although they realize different functions on $[0, 1]^n$.

6.2 Rational weights

As digital computers represent numbers by finite bit strings, the weights and biases of neural networks stored on computers are all rational, and in the floating-point and fixed-point formats even dyadic. The rational case hence covers ReLU networks stored on digital computers.

Neural networks with rational weights and biases realize continuous piecewise linear functions with rational coefficients, the rational McNaughton functions. The logic whose truth functions these are is Rational Łukasiewicz logic, introduced by Gerla [13], which extends Łukasiewicz logic by the unary connectives δ_i, $i \in \mathbb{N}$, with $\delta_i x = x/i$, $x \in [0, 1]$.

The formulae of this logic and their algebraic counterpart, the standard DMV algebra, are defined next.

Definition 27. A *Rational Łukasiewicz formula* is built from propositional variables, the constants 0 and 1, and the operations $\oplus$, $\odot$, $\neg$, and δ_i ($i \in \mathbb{N}$).

Definition 28. [13, Def. 3.1] A *divisible MV (DMV) algebra* is an MV algebra extended with unary operations $\{\delta_i\}_{i\in\mathbb{N}}$ satisfying $\oplus^n \delta_n x = x$ and $\delta_n x \odot (\oplus^{n-1} \delta_n x) = 0$ for all $n \in \mathbb{N}$. We write $\mathcal{DMV}$ for the axioms of $\mathcal{MV}$ listed in Definition 4 together with these two. The operations δ_i induce a further tier of symmetries, the scaling symmetries Ax. DMV-n of Appendix A. The standard DMV algebra is $\mathbb{I}_D = ([0, 1], \oplus, \odot, \neg, 0, 1, \{\delta_i\}_{i\in\mathbb{N}})$ with $\delta_i x = x/i$.

Here step (iii) of Section 1 does not need modifications, while steps (i) and (ii) do.

Step (i) requires a modification of Extraction, Step II. Specifically, for a σ-node with rational weights and bias, let s be the least common multiple of their denominators, so that

sf_u has integer coefficients. The identity

$$\sigma(x) = \tfrac{1}{s}\big(\sigma(sx) + \sigma(sx-1) + \cdots + \sigma(sx-(s-1))\big), \quad x \in \mathbb{R}, \tag{19}$$

converts the node to s integer-coefficient σ-nodes, to which Lemma 1 applies, returning Łukasiewicz formulae $\tau_0, \ldots, \tau_{s-1}$ for $\sigma(sf_u - i)$, $i = 0, \ldots, s-1$. The overall node formula is then $\bigoplus_{i=0}^{s-1} \delta_s \tau_i$, whose truth function is the right-hand side of (19), the truncation in $\oplus$ never taking effect as the sum equals $\sigma(f_u) \leq 1$. Extraction, Steps I and III are unchanged. The minterms of Definition 15 have to be replaced, as they are built from integer coefficients. We let C^D be the set of truncations $\sigma(m_1 x_1 + \cdots + m_n x_n + b)$ with $m_1, \ldots, m_n, b \in \mathbb{Q}$, $\mathcal{C}^D_{\mathrm{norm}}$ the set of corresponding formulae produced by the modified Extraction, Step II, and $\kappa^D : \mathcal{C}^D_{\mathrm{norm}} \to C^D$ the mapping taking such a formula to its truth function, as in Definition 16. Lemmata 3 and 4 hold with $\mathbb{I}_D$ in place of $\mathbb{I}$ and κ^D in place of κ.

Construction, Step I reads the rational weights and bias off $\kappa^D([v])$, with $m_{i'}$ and $b_{v_i^{(j)}}$ in (14) now rational, and Construction, Step II uses (16) and (17), which hold for every $x \in \mathbb{R}$. The uniqueness argument of Section 5.1, which shows that the weights and the bias can be read off the local map unambiguously, carries over and covers, in addition, the constant local maps of value $c \in (0,1)$, which cannot arise for integer coefficients. Indeed, suppose that $\sigma\big(b_{v_i^{(j)}} + \sum_{i'} m_{i'} x_{i'}\big) = c$ for all $x \in [0,1]^{d_{j-1}}$, with $c \in (0,1)$. As $\sigma(t) = c$ has $t = c$ as its only solution, $b_{v_i^{(j)}} + \sum_{i'} m_{i'} x_{i'} = c$ for all $x \in [0,1]^{d_{j-1}}$. An affine map constant on a set with non-empty interior has vanishing linear part, whence $m_{i'} = 0$ for all i' and $b_{v_i^{(j)}} = c$. Such local maps are therefore covered by (14).

We turn to step (ii), which follows from the completeness theorem for DMV algebras [13].

Theorem 7 ([13, Thms. 3.14–3.15])**.** Let τ_1, τ_2 be Rational Łukasiewicz formulae. If $\tau_1^{\mathbb{I}_D} = \tau_2^{\mathbb{I}_D}$, then $\tau_1 \overset{\mathcal{DMV}}{\sim} \tau_2$.

With Theorem 7 supplying step (ii), we obtain the identification result for rational weights and biases.

Theorem 8. For $n \in \mathbb{N}$, let $\mathfrak{N}$ be the class of non-degenerate ReLU networks with rational weights and biases realizing functions $f : [0,1]^n \to [0,1]$. The set of axioms $\mathcal{DMV}$ completely identifies $\mathfrak{N}$ over $[0,1]^n$, that is, $\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$ implies $\mathcal{N}_1 \overset{\mathcal{DMV}}{\sim} \mathcal{N}_2$ for all $\mathcal{N}_1, \mathcal{N}_2 \in \mathfrak{N}$.

Proof Step (i) (extraction): set $\tau_i = [\mathtt{ext}(\mathcal{N}_i)]$, $i = 1, 2$, with Extraction, Step II modified as above, so that $\tau_i^{\mathbb{I}_D} = \langle \mathcal{N}_i \rangle$ on $[0,1]^n$. Since $\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$, we have $\tau_1^{\mathbb{I}_D} = \tau_2^{\mathbb{I}_D}$. Step (ii) (derivation): Theorem 7 yields $\tau_1 \overset{\mathcal{DMV}}{\sim} \tau_2$, which is $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{DMV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$ by Definition 21, and hence $\mathcal{N}_1 \overset{\mathcal{DMV}}{\sim} \mathcal{N}_2$ by Definition 22. ∎

6.3 Real weights

Real weights and biases are the idealization that goes with exact arithmetic, and the setting in which neural networks are usually studied. ReLU networks with real weights and biases realize continuous piecewise linear functions with real coefficients, the real McNaughton

functions. These are the truth functions of Riesz Łukasiewicz logic $\mathcal{RL}$, introduced by Di Nola and Leuştean [11], which extends Łukasiewicz logic by the unary connectives Δ_r, $r \in [0,1]$, with $\Delta_r x = rx$, $x \in [0,1]$.

The formulae of this logic and their algebraic counterpart, the standard RMV algebra, are defined next.

Definition 29. An $\mathcal{RL}$ *formula* is built from propositional variables, the constants 0 and 1, and the operations $\oplus$, $\odot$, $\neg$, and Δ_r ($r \in [0,1]$).

Definition 30. [11, Def. 3.1] A *Riesz MV (RMV) algebra* is an MV algebra with underlying set A, extended with $\{\Delta_r\}_{r\in[0,1]}$ satisfying $\Delta_r(x \odot \neg y) = (\Delta_r x) \odot \neg(\Delta_r y)$, $\Delta_{r\odot\neg q}x = (\Delta_r x) \odot \neg(\Delta_q x)$, $\Delta_r(\Delta_q x) = \Delta_{rq}x$, $\Delta_1 x = x$, for all scalars $r, q \in [0,1]$ and all $x, y \in A$. Here $r \odot \neg q = \max\{0, r-q\}$. We write $\mathcal{RMV}$ for the axioms of $\mathcal{MV}$ listed in Definition 4 together with these four. The operations Δ_r enlarge the scaling symmetries of Appendix A from the countable family Ax. DMV-n, $n \in \mathbb{N}$, to the uncountable family Ax. Riesz-r, $r \in (0,1]$. The standard RMV algebra is $\mathbb{I}_R = ([0,1], \oplus, \odot, \neg, 0, 1, \{\Delta_r\}_{r\in[0,1]})$ with $\Delta_r x = rx$.

Here step (iii) of Section 1 does not need modifications, while steps (i) and (ii) do.

Step (i) changes only at Extraction, Step II. The identities of Lemma 1 continue to hold for real coefficients, the proof in Appendix B nowhere using that the coefficients are integers, but each application of these identities lowers a weight by one and hence leaves a fractional part in $[0,1)$, which, when non-zero, is removed by the following specialization of [11, Lemma 7.5(a)].

Lemma 7 ([11, Lemma 7.5(a)])**.** *Let $f(x) = m_1x_1 + \cdots + m_nx_n + b$ with $m_1, \ldots, m_n, b \in \mathbb{R}$, let i be an index with $m_i \in (0,1]$, and set $f_\circ = f - m_ix_i$. Then, on $[0,1]^n$,*

$$\sigma(f) = (\sigma(f_\circ) \oplus \Delta_{m_i} x_i) \odot \sigma(f_\circ + 1). \tag{7'}$$

Formula extraction proceeds as in the integer case, eliminating at each step the first variable present, by Lemma 1 and Lemma 7 in this order. Extraction, Steps I and III are unchanged. The minterms of Definition 15 have to be replaced here as well. We let C^R be the set of truncations $\sigma(m_1x_1 + \cdots + m_nx_n + b)$ with $m_1, \ldots, m_n, b \in \mathbb{R}$, $\mathcal{C}^R_{\text{norm}}$ the set of corresponding formulae produced by the modified Extraction, Step II, and $\kappa^R : \mathcal{C}^R_{\text{norm}} \to C^R$ the mapping taking such a formula to its truth function, as in Definition 16. Lemmata 3 and 4 hold with $\mathbb{I}_R$ in place of $\mathbb{I}$ and κ^R in place of κ.

Construction, Step I reads the real weights and bias off $\kappa^R([v])$, with $m_{i'}$ and $b_{v_i^{(j)}}$ in (14) now real, and Construction, Step II uses (16) and (17). The uniqueness argument of Section 5.1 carries over as in Section 6.2, constant local maps of value $c \in (0,1)$ again being covered by (14).

We turn to step (ii), which follows from the completeness theorem for RMV algebras [11].

Theorem 9 ([11, Thm. 4.4])**.** Let τ_1, τ_2 be $\mathcal{RL}$ formulae. If $\tau_1^{\mathbb{I}_R} = \tau_2^{\mathbb{I}_R}$, then $\tau_1 \overset{\mathcal{RMV}}{\sim} \tau_2$.

With Theorem 9 supplying step (ii), we obtain the identification result for real weights and biases.

Theorem 10. For $n \in \mathbb{N}$, let $\mathfrak{N}$ be the class of non-degenerate ReLU networks with real weights and biases realizing functions $f : [0,1]^n \to [0,1]$. The set of axioms $\mathcal{RMV}$ completely identifies $\mathfrak{N}$ over $[0,1]^n$, that is, $\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$ implies $\mathcal{N}_1 \overset{\mathcal{RMV}}{\sim} \mathcal{N}_2$ for all $\mathcal{N}_1, \mathcal{N}_2 \in \mathfrak{N}$.

Proof Step (i) (extraction): set $\tau_i = [\mathtt{ext}(\mathcal{N}_i)]$, $i = 1, 2$, with Extraction, Step II modified as above, so that $\tau_i^{\mathbb{I}_R} = \langle \mathcal{N}_i \rangle$ on $[0,1]^n$. Since $\mathcal{N}_1 \sim_{[0,1]^n} \mathcal{N}_2$, we have $\tau_1^{\mathbb{I}_R} = \tau_2^{\mathbb{I}_R}$. Step (ii) (derivation): Theorem 9 yields $\tau_1 \overset{\mathcal{RMV}}{\sim} \tau_2$, which is $\mathtt{ext}(\mathcal{N}_1) \overset{\mathcal{RMV}}{\sim} \mathtt{ext}(\mathcal{N}_2)$ by Definition 21, and hence $\mathcal{N}_1 \overset{\mathcal{RMV}}{\sim} \mathcal{N}_2$ by Definition 22. ■

All results so far pertain to networks on $[0,1]^n$ realizing $[0,1]$-valued functions, which is not what one encounters in practice. We next show that Theorem 10 continues to hold with $[0,1]^n$ replaced by an arbitrary box, in the sense that two networks agreeing on the box are interderivable through $\mathcal{RMV}$ after an affine normalization of input and output. Specifically, let $\mathcal{N}_1, \mathcal{N}_2$ be non-degenerate ReLU networks with real weights and biases, and let $B = \prod_{i=1}^n [a_i, c_i]$, with $a_i, c_i \in \mathbb{R}$, $a_i < c_i$, be such that $\mathcal{N}_1 \sim_B \mathcal{N}_2$, that is, $\langle \mathcal{N}_1 \rangle = \langle \mathcal{N}_2 \rangle$ on B. Let $\alpha : [0,1]^n \to B$ be the affine bijection $\alpha(x)_i = a_i + (c_i - a_i)x_i$. The function $\langle \mathcal{N}_1 \rangle \circ \alpha = \langle \mathcal{N}_2 \rangle \circ \alpha$ is continuous, $\langle \mathcal{N}_i \rangle$ being continuous (Definition 1) and α affine, and hence attains a minimum $t_{\min}$ and a maximum $t_{\max}$ on the compact set $[0,1]^n$. Let $\beta(t) = (t - t_{\min})/(t_{\max} - t_{\min})$ if $t_{\max} > t_{\min}$, and $\beta(t) = t - t_{\min}$ if $t_{\max} = t_{\min}$, in the latter case the function being constant and β sending its value to 0; in both cases β is affine and maps the range into $[0,1]$. Precomposing the first layer of $\mathcal{N}_i$ with α and postcomposing its output node with β modifies weights and biases only, α entering the affine combination of the first layer and β the affine map the network ends in (Definition 1), and yields networks $\mathcal{N}_i^\sharp$ realizing $\beta \circ \langle \mathcal{N}_i \rangle \circ \alpha$ on $[0,1]^n$, taking values in $[0,1]$. Since $\langle \mathcal{N}_1 \rangle = \langle \mathcal{N}_2 \rangle$ on B, we have $\mathcal{N}_1^\sharp \sim_{[0,1]^n} \mathcal{N}_2^\sharp$, and $\mathcal{N}_1^\sharp, \mathcal{N}_2^\sharp$ are non-degenerate because α and β are invertible. Theorem 10 therefore yields $\mathcal{N}_1^\sharp \overset{\mathcal{RMV}}{\sim} \mathcal{N}_2^\sharp$, as desired.

7 Concluding remarks

An interesting question left open in the developments in this paper is whether the finite sequence of MV axiom applications whose existence Theorem 3 guarantees can be made explicit, in the sense of an algorithm that takes the two networks and returns such a sequence. In the classical setting of switching circuits and Boolean algebra, the counterparts of ReLU networks and MV algebra as described in Section 1.3, this is possible. Two circuits computing the same Boolean function are transformed into each other by an explicit sequence of Boolean axiom rewrites as follows. Each circuit corresponds to a Boolean formula, and any such formula can be brought to full disjunctive normal form by a set of rewriting rules, every one of which is a Boolean identity. This set is terminating and confluent [9, Sec. 5.1], so that the rules may be applied in any order and always halt at the same result. Each of the identities the rules apply is either a Boolean axiom or derivable from the Boolean axioms in finitely many steps, so that every rule application unfolds into finitely many Boolean axiom applications and the passage from a formula to its normal form, its *normalization*, is a derivation in the sense of Definition 20. The normal form reached is determined by the truth table of the formula, a table of 2^n values for a function in n variables, and is unique

up to the order of its terms. As the two circuits compute the same function, their formulae have the same truth table, and the rewriting therefore takes both to the same normal form, up to that order. Explicitly, let τ_1 and τ_2 be the formulae corresponding to the two circuits, and record the successive formulae produced by their normalizations,

$$\tau_1 = \varphi_1, \ldots, \varphi_p = \delta, \qquad \tau_2 = \theta_1, \ldots, \theta_q = \delta',$$

with δ and δ' agreeing up to the order of their terms. Since associativity permits re-bracketing and commutativity the interchange of neighboring terms, every reordering of the terms of a full disjunctive normal form is achieved by finitely many Boolean axiom rewrites, so that the passage from δ to δ' is again a derivation in the sense of Definition 20, which is what allows the two normalizations to be glued. Every rewrite, from φ_i to φ_{i+1} or from θ_i to θ_{i+1}, replaces a subformula by one an identity equates it to and is therefore reversible, the De Morgan law, for instance, turning $\neg(a \wedge b)$ into $\neg a \vee \neg b$ in one direction and back in the other. Reversing the sequence for τ_2 and gluing the three parts hence yields a derivation from τ_1 to τ_2 in the sense of Definition 20. The field of logic synthesis is built on the idea of transforming Boolean circuits by algebraic rewriting [20].

For two ReLU networks realizing the same function, extraction delivers two Łukasiewicz formulae with the same truth function. What is missing is a terminating and confluent set of rewriting rules bringing a Łukasiewicz formula to a canonical form, be it flat or compositional. Flattening does not help, removing the freedom in the normal form architecture (Proposition 4) but leaving, at the single node that remains, two Łukasiewicz formulae with the same truth function and no rules driving them to a common form. While canonical forms for Łukasiewicz formulae are available, every formula having the same truth function as an infimum of finite suprema of formulae realizing single truncated affine functions [10], and the passage from the truth function to such a form is constructive [17], these forms are read off the McNaughton representation of the truth function rather than reached by rewriting, and hence deliver no sequence of axiom applications. One can always enumerate the derivations starting from the first formula until the second appears, and completeness (Theorems 2, 5, 7, 9) guarantees that it eventually does. The same holds in the Boolean case, by Theorem 5 at $k = 1$. However, in the Boolean case the enumeration need not be carried out, termination and confluence taking the formula to its canonical form directly. This can be used to find an explicit derivation sequence for $k = 1$ in Theorem 6. All other cases remain open.

Declaration on the use of AI-assisted technologies

This paper started from a complete manuscript of ours, containing all definitions, statements, and proofs, written without the use of AI. We subsequently revised it line by line in dialogue with Claude (Anthropic), and in the course of this revision expanded concepts, proved new results, and developed the mathematical language of the paper further. The tool proposed formulations, drafted prose, supplied routine proof steps, audited notation, hypotheses, terminology, and cross-references for consistency, and verified algebraic identities symbolically and numerically. We read every proposal adversarially and accepted, modified, or rejected it on that reading, much of the final wording being ours verbatim. Nothing entered the manuscript unverified, and we take full responsibility for its content.

References

[1] Stefano Aguzzoli. *Geometric and Proof-Theoretic Issues in Łukasiewicz Propositional Logics*. PhD thesis, University of Siena, 1998.

[2] Franz Baader and Wayne Snyder. Unification theory. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 8, pages 445–533. Elsevier, 2001.

[3] H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In *PARLE Parallel Architectures and Languages Europe*, volume 259 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 1987. doi: 10.1007/3-540-17945-3_8.

[4] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Interacting Hopf algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, 2017. doi: 10.1016/j.jpaa.2016.06.002.

[5] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020. ISSN 1533-7928.

[6] C. C. Chang. Algebraic analysis of many valued logics. *Transactions of the American Mathematical Society*, 88(2):467–490, 1958. ISSN 0002-9947. doi: 10.2307/1993227.

[7] Chen Chung Chang. A new proof of the completeness of the Łukasiewicz axioms. *Transactions of the American Mathematical Society*, 93(1):74–80, 1959.

[8] Roberto L. O. Cignoli, Itala M. L. D'Ottaviano, and Daniele Mundici. *Algebraic Foundations of Many-Valued Reasoning*, volume 7 of *Trends in Logic*. Kluwer Academic Publishers, Dordrecht, 2000. doi: 10.1007/978-94-015-9480-6.

[9] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier, 1990.

[10] A. Di Nola and A. Lettieri. On normal forms in Łukasiewicz logic. *Archive for Mathematical Logic*, 43(6):795–823, August 2004.

[11] Antonio Di Nola and Ioana Leuştean. Łukasiewicz logic and Riesz spaces. *Soft Computing*, 18(12):2349–2363, 2014.

[12] David A. Duffy. *Principles of Automated Theorem Proving*. Wiley Professional Computing. John Wiley & Sons, Chichester, 1991. ISBN 0-471-92784-8.

[13] Brunella Gerla. Rational Łukasiewicz logic and DMV-algebras. *Neural Network World*, 11(6):579–594, 2001.

[14] Revaz Grigolia. Algebraic analysis of Łukasiewicz–Tarski's n-valued logical systems. In Ryszard Wójcicki and Grzegorz Malinowski, editors, *Selected Papers on Łukasiewicz Sentential Calculi*, pages 81–92. Ossolineum, Wrocław, 1977.

[15] Elisenda Grigsby, Kathryn Lindsey, and David Rolnick. Hidden symmetries of ReLU networks. In *Proceedings of the 40th International Conference on Machine Learning*, pages 11734–11760. PMLR, July 2023.

[16] Robert McNaughton. A theorem about infinite-valued sentential logic. *The Journal of Symbolic Logic*, 16(1):1–13, 1951. ISSN 0022-4812. doi: 10.2307/2268660.

[17] D. Mundici. A constructive proof of McNaughton's theorem in infinite-valued logic. *The Journal of Symbolic Logic*, 59(2):596–602, 1994.

[18] Mary Phuong and Christoph H Lampert. Functional vs. parametric equivalence of relu networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[19] Alan Rose and J. Barkley Rosser. Fragments of many-valued statement calculi. *Transactions of the American Mathematical Society*, 87(1):1–53, 1958.

[20] Claude E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57(12):713–723, 1938.

[21] Verner Vlačić and Helmut Bölcskei. Affine symmetries and neural network identifiability. *Advances in Mathematics*, 376:1–72, 2021. Article 107485.

[22] Yani Zhang and Helmut Bölcskei. Extracting formulae in many-valued logic from deep neural networks. *IEEE Transactions on Signal Processing*, 74:1–12, 2026. doi: 10.1109/TSP.2025.3615511.

[23] Bo Zhao, Robin Walters, and Rose Yu. Symmetry in neural network parameter spaces. *Transactions on Machine Learning Research*, 2025. arXiv:2506.13018.

Appendix A. Symmetries induced by the axioms

The MV operations are given by

$$\begin{aligned} x \odot y &= \max\{0, x + y - 1\} = \rho(x + y - 1) \\ x \oplus y &= \min\{1, x + y\} = 1 - \rho(-x - y + 1) \\ \neg x &= 1 - x. \end{aligned}$$

The DMV extension adds unary operations $\delta_n : x \mapsto x/n$ ($n \in \mathbb{N}$); the Riesz extension adds $\Delta_r : x \mapsto rx$ ($r \in [0, 1]$). Using these expressions, the MV axioms in Definition 4 can be rewritten as compositions of affine maps and ρ. In the MV setting (integer coefficients) the axioms fall into three tiers; the DMV and Riesz extensions introduce a fourth. The grid symmetries of the finite-domain setting (Section 6.1) form no tier of their own and are listed at the end of this appendix.

Tier 1: Degenerate symmetries. These identities hold by direct arithmetic, the argument of every ρ either cancelling to a constant or lying permanently in the linear or the clipped-negative regime, so that no clipping transition occurs. They correspond to trivial structural network simplifications, replacing a subnetwork of constant output by a single node realizing that constant value or eliminating an identity operation. Ax. 6 and 6′ are the sole exceptions, their ρ-arguments crossing zero on $[0, 1]^2$, each axiom's two sides expanding to the same ρ-expression, so that its two syntactically different formulae realize the same ReLU network.

- Ax. 3. $x \oplus \neg x = 1$

$$1 - \rho(-x - (1 - x) + 1) = 1, \quad x \in [0, 1]$$

- Ax. 3′. $x \odot \neg x = 0$

$$\rho(x + 1 - x - 1) = 0, \quad x \in [0, 1]$$

- Ax. 4. $x \oplus 1 = 1$

$$1 - \rho(-x - 1 + 1) = 1, \quad x \in [0, 1]$$

- Ax. 4′. $x \odot 0 = 0$

$$\rho(x + 0 - 1) = 0, \quad x \in [0, 1]$$

- Ax. 5. $x \oplus 0 = x$

$$1 - \rho(-x - 0 + 1) = x, \quad x \in [0, 1]$$

- Ax. 5′. $x \odot 1 = x$

$$\rho(x + 1 - 1) = x, \quad x \in [0, 1]$$

- Ax. 6. $\neg(x \oplus y) = \neg x \odot \neg y$

$$\rho(-x - y + 1) = \rho(1 - x + 1 - y - 1), \quad x, y \in [0, 1]$$

- Ax. 6′. $\neg(x \odot y) = \neg x \oplus \neg y$

$$1 - \rho(x + y - 1) = 1 - \rho(-(1 - x) - (1 - y) + 1), \quad x, y \in [0, 1]$$

- Ax. 7. $x = \neg(\neg x)$

$$x = 1 - (1 - x), \quad x \in [0, 1]$$

- Ax. 8. $\neg 0 = 1$

$$1 - 0 = 1$$

Tier 2: Permutation symmetries. Each of the two axioms reorders neurons within a hidden layer and applies the same permutation to the corresponding incoming and outgoing weights.

- Ax. 1. $x \oplus y = y \oplus x$

$$1 - \rho(-x - y + 1) = 1 - \rho(-y - x + 1), \quad x, y \in [0, 1].$$

- Ax. 1′. $x \odot y = y \odot x$

$$\rho(x + y - 1) = \rho(y + x - 1), \quad x, y \in [0, 1].$$

Tier 3: Deep symmetries. The primitive axioms in this tier span at least three network layers, with at least one inner ρ clipping on $[0, 1]^n$. In Ax. 2 this is $\rho(-y - z + 1)$, which vanishes at $y = z = 1$ and is positive at $y = z = 0$, in Ax. 2′ it is $\rho(y + z - 1)$, which vanishes at $y = z = 0$ and is positive at $y = z = 1$, and in Ax. 9 and 9′ it is $\rho(x - y)$ and $\rho(y - x)$, vanishing on $\{x \leq y\}$ and $\{y \leq x\}$, respectively. Further genuine deep symmetries, spanning more than three layers, can be composed from these and Tier 1 identities.

- Ax. 2. $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

$$1 - \rho(-x + \rho(-y - z + 1)) = 1 - \rho(\rho(-x - y + 1) - z), \quad x, y, z \in [0, 1].$$

- Ax. 2′. $x \odot (y \odot z) = (x \odot y) \odot z$

$$\rho(x + \rho(y + z - 1) - 1) = \rho(\rho(x + y - 1) + z - 1), \quad x, y, z \in [0, 1].$$

- Ax. 9. $(x \odot \neg y) \oplus y = (y \odot \neg x) \oplus x$

$$1 - \rho(-\rho(x + 1 - y - 1) - y + 1) = 1 - \rho(-\rho(y + 1 - x - 1) - x + 1), \quad x, y \in [0, 1]$$

- Ax. 9′. $(x \oplus \neg y) \odot y = (y \oplus \neg x) \odot x$

$$\rho(1 - \rho(-x - 1 + y + 1) + y - 1) = \rho(1 - \rho(-y - 1 + x + 1) + x - 1), \quad x, y \in [0, 1]$$

Tier 4 (DMV/Riesz): Scaling symmetries. The operations $\delta_n(x) = x/n$ ($n \in \mathbb{N}$) and $\Delta_r(x) = rx$ ($r \in [0,1]$) are affine and hence trivially single-layer networks (see Definition 1). The two axioms of Definition 28 translate into

- Definition 28, first axiom ($n \in \mathbb{N}$). $\oplus^n \delta_n x = x$

$$1 - \rho(1 - x) = x, \quad x \in [0,1],$$

- Definition 28, second axiom ($n \in \mathbb{N}$). $\delta_n x \odot (\oplus^{n-1} \delta_n x) = 0$

$$\rho(x - 1) = 0, \quad x \in [0,1],$$

both degenerate, the second having the same ρ-form as Ax. $4'$, so that they pertain to Tier 1 and lead to no new category of symmetry. New symmetries arise from the interaction of δ_n with $\oplus$, $\odot$, and $\neg$. To see this, we apply δ_n to the truncated difference $x \odot \neg y$, whose ρ-form is $\rho(x - y)$. Concretely, positive homogeneity of ρ yields the following symmetry.

- Ax. DMV-n ($n \in \mathbb{N}$). $\delta_n(x \odot \neg y) = (\delta_n x) \odot \neg(\delta_n y)$

$$\tfrac{1}{n}\,\rho(x - y) = \rho\big(\tfrac{x-y}{n}\big), \quad x, y \in [0,1].$$

The same computation with r in place of $1/n$ gives the Riesz counterpart, which for Δ_r is the first axiom of Definition 30.

- Ax. Riesz-r ($r \in (0,1]$; for $r = 0$ both sides vanish). $\Delta_r(x \odot \neg y) = (\Delta_r x) \odot \neg(\Delta_r y)$

$$r\,\rho(x - y) = \rho\big(r(x - y)\big), \quad x, y \in [0,1].$$

The remaining three axioms of Definition 30 add nothing further. The second is the same identity with the roles of scalar and argument exchanged, its ρ-form being $\rho(r - q)\,x = \rho\big((r - q)x\big)$, while the third and fourth contain no ρ, stating that rescalings compose, $r(qx) = (rq)x$, and that Δ_1 is the identity.

Finite domain: grid symmetries. The axioms that Definition 25 adds differ from those above in holding on I_k only. With $\oplus^m x = 1 - \rho(1 - mx)$ and $\odot^m x = \rho(mx - m + 1)$, $x \in [0,1]$, the axiom $\oplus^{k+1} x = \oplus^k x$, which for $k = 1$ reads $x \oplus x = x$, translates into

- Ax. $\mathcal{MV}_k$. $\oplus^{k+1} x = \oplus^k x$

$$1 - \rho(-(k+1)x + 1) = 1 - \rho(-kx + 1), \quad x \in I_k,$$

an identity that fails on $[0,1]$. To see this, note that

$$\oplus^k x = \min\{1, kx\}, \qquad \oplus^{k+1} x = \min\{1, (k+1)x\}.$$

For $x = m/k \in I_k$, $m \in \{0, 1, \ldots, k\}$, we have $kx = m$ and $(k+1)x = (k+1)m/k \geq m$, so that

$$\oplus^k x = \oplus^{k+1} x = \begin{cases} 0, & m = 0, \\ 1, & m \geq 1, \end{cases}$$

whereas at $x = 1/(2k)$,

$$\oplus^k x = \tfrac{1}{2}, \qquad \oplus^{k+1} x = \tfrac{k+1}{2k}.$$

The second set of axioms of Definition 25 translates into

- Ax. $\mathcal{MV}_{k,j}$ ($j \in \{2,\dots,k-1\}$ not dividing k). $\odot^{k+1}\big(\oplus^{j}(\odot^{j-1}x)\big) = \oplus^{k+1}(\odot^{j}x)$

$$\rho\Big((k+1)\big(1-\rho\big(1-j\,\rho((j-1)x-j+2)\big)\big)-k\Big)$$
$$= 1-\rho\big(1-(k+1)\,\rho(jx-j+1)\big), \quad x \in I_k,$$

the left-hand side nesting ρ to depth three and the right-hand side to depth two. Structurally Ax. $\mathcal{MV}_{k,j}$ meets the Tier 3 criterion, nesting ρ to depth three with an inner ρ that clips. Both identities hold on I_k only, however, and therefore belong to no tier.

Appendix B. Proof of Lemma 1

Proof For all $t \in \mathbb{R}$, $\sigma(t) = 1-\sigma(1-t)$, which establishes (8).

To show (7), recall that $f_\circ = f - x_1$, so $f = f_\circ + x_1$. The proof below uses only $f = f_\circ + x_1$ and $x_1 \in [0,1]$, so that (7) holds for arbitrary real $m_1,\dots,m_n,b$; the hypothesis $m_1 \geq 1$ serves only to guarantee termination of EXTR, which applies (7) with m_1 lowered by one at each step (Section 2.3). As (7) is an identity between functions on $[0,1]^n$, it suffices to verify it at an arbitrary $x \in [0,1]^n$; fix such an x and distinguish four cases by the value $f_\circ(x)$, following [17].

Case 1: $f_\circ(x) \geq 1$. Then $f \geq 1$, so the left-hand side of (7) is $\sigma(f) = 1$, and the right-hand side is

$$(\sigma(f_\circ) \oplus x_1) \odot \sigma(f_\circ + 1) = (1 \oplus x_1) \odot 1 = 1.$$

Case 2: $f_\circ(x) \leq -1$. Then $f \leq 0$, so $\sigma(f) = 0$, and the right-hand side is

$$(\sigma(f_\circ) \oplus x_1) \odot \sigma(f_\circ + 1) = (0 \oplus x_1) \odot 0 = 0.$$

Case 3: $-1 < f_\circ(x) \leq 0$. Then $f \in (-1,1]$, and the right-hand side of (7) is

$$\begin{aligned}
&(\sigma(f_\circ) \oplus x_1) \odot \sigma(f_\circ + 1)\\
&= (0 \oplus x_1) \odot (f_\circ + 1)\\
&= x_1 \odot (f_\circ + 1)\\
&= \max\{0, x_1 + f_\circ + 1 - 1\}\\
&= \max\{0, f\}\\
&= \sigma(f).
\end{aligned}$$

Case 4: $0 < f_\circ(x) < 1$. Then $f \in (0,2)$, and the right-hand side of (7) is

$$\begin{aligned}
&(\sigma(f_\circ) \oplus x_1) \odot \sigma(f_\circ + 1)\\
&= (f_\circ \oplus x_1) \odot 1\\
&= f_\circ \oplus x_1\\
&= \min\{1, f_\circ + x_1\}\\
&= \min\{1, f\}\\
&= \sigma(f).
\end{aligned}$$

In each of the four cases the two sides agree at x, establishing (7). ∎

Appendix C. Deferred proofs in Section 3

C.1 Proof of Lemma 2

Proof A Łukasiewicz formula τ is generated from propositional variables and the constants $0, 1$ by the operations $\neg, \oplus, \odot$. We prove the identity $\big(\tau(\zeta)\big)(\zeta') = \tau(\zeta \bullet \zeta')$ by induction on the structure of τ—we verify it for the constants and for single variables (base cases) and show that it is inherited under each of the three formation rules (induction step); the principle of structural induction then yields it for every τ with variables in $\{x_{i_1}, \dots, x_{i_k}\}$.

Base cases. If $\tau \in \{0, 1\}$, then by Definition 10, $\tau(\zeta) = \tau$ and, likewise, $\tau(\zeta \bullet \zeta') = \tau$; hence $\big(\tau(\zeta)\big)(\zeta') = \tau = \tau(\zeta \bullet \zeta')$. If τ is a single variable, then $\tau = x_{i_j}$ for some $j \in \{1, \dots, k\}$, so $\tau(\zeta) = \chi_j$ and $\big(\tau(\zeta)\big)(\zeta') = \chi_j(\zeta') = \tau(\zeta \bullet \zeta')$ by Definition 12.

Induction step. Otherwise τ is of one of the forms $\neg\tau'$, $\gamma \oplus \eta$, or $\gamma \odot \eta$ for Łukasiewicz formulae τ', γ, η, whose variables automatically lie in $\{x_{i_1}, \dots, x_{i_k}\}$. The induction hypothesis is that the identity $\big(\theta(\zeta)\big)(\zeta') = \theta(\zeta \bullet \zeta')$ holds for each of $\theta = \tau', \gamma, \eta$. Since a substitution replaces variable occurrences and leaves the connectives unchanged, $(\neg\tau')(\zeta) = \neg(\tau'(\zeta))$ and $(\gamma \oplus \eta)(\zeta) = \gamma(\zeta) \oplus \eta(\zeta)$, and likewise for $\odot$.

- If $\tau = \neg\tau'$, then $\big(\tau(\zeta)\big)(\zeta') = \big(\neg(\tau'(\zeta))\big)(\zeta') = \neg\big(\big(\tau'(\zeta)\big)(\zeta')\big) = \neg\big(\tau'(\zeta \bullet \zeta')\big) = \tau(\zeta \bullet \zeta')$, where the third equality holds by the induction hypothesis for τ'.
- If $\tau = \gamma \oplus \eta$, then $\big(\tau(\zeta)\big)(\zeta') = \big(\gamma(\zeta) \oplus \eta(\zeta)\big)(\zeta') = \big(\gamma(\zeta)\big)(\zeta') \oplus \big(\eta(\zeta)\big)(\zeta') = \gamma(\zeta \bullet \zeta') \oplus \eta(\zeta \bullet \zeta') = \tau(\zeta \bullet \zeta')$, where the third equality holds by the induction hypothesis for γ and η.
- The case $\tau = \gamma \odot \eta$ is treated exactly as $\tau = \gamma \oplus \eta$, with $\odot$ in place of $\oplus$.

This completes the induction. ■

C.2 Proof of Lemma 3

Proof The lemma asserts that $\langle \mathcal{K} \rangle = [\mathcal{G}]^{\mathbb{I}}$. By Definition 8 and (11) this reads $\langle\!\langle v_{\text{out}} \rangle\!\rangle = \big([v_{\text{out}}](\eta_L)\big)^{\mathbb{I}}$, which is what we prove. Let $\eta_0 = \{x_1 \mapsto x_1, \dots, x_{d_0} \mapsto x_{d_0}\}$ be the *identity substitution*, so that $\tau(\eta_0) = \tau$ for every formula τ in $x_1, \dots, x_{d_0}$, and let η_j, $j = 1, \dots, L$, be as in Proposition 2. We show by induction on $j \in \{0, 1, \dots, L\}$ that $\langle\!\langle v_i^{(j)} \rangle\!\rangle = \big([v_i^{(j)}](\eta_j)\big)^{\mathbb{I}}$ for every level-j node, the case $j = L$ being the lemma. For $j = 0$,

$$\big([v_i^{(0)}](\eta_0)\big)^{\mathbb{I}} = \big[v_i^{(0)}\big]^{\mathbb{I}} = x_i^{\mathbb{I}} = \langle v_i^{(0)} \rangle = \langle\!\langle v_i^{(0)} \rangle\!\rangle,$$

the first equality because η_0 is the identity substitution, the second and third by Definition 11 and Section 2.3, and the fourth by Definition 8. For the step from $j-1$ to j, Definition 8 gives

$$\begin{aligned} \langle\!\langle v_i^{(j)} \rangle\!\rangle(x) &= \sigma\Big(\sum_{i'=1}^{d_{j-1}} w_{v_i^{(j)}, v_{i'}^{(j-1)}} \, \langle\!\langle v_{i'}^{(j-1)} \rangle\!\rangle(x) + b_{v_i^{(j)}}\Big) \\ &= \langle v_i^{(j)} \rangle\big(\langle\!\langle v_1^{(j-1)} \rangle\!\rangle(x), \dots, \langle\!\langle v_{d_{j-1}}^{(j-1)} \rangle\!\rangle(x)\big), \end{aligned}$$

where the second equality follows from (3) for $1 \le j \le L-1$ and from (4) for $j = L$. With $\langle v \rangle = [v]^{\mathbb{I}}$, the hypothesis of the lemma, this reads

$$\langle\!\langle v_i^{(j)} \rangle\!\rangle = [v_i^{(j)}]^{\mathbb{I}}\big(\langle\!\langle v_1^{(j-1)} \rangle\!\rangle, \ldots, \langle\!\langle v_{d_{j-1}}^{(j-1)} \rangle\!\rangle\big), \tag{20}$$

the right-hand side of which equals $\big([v_i^{(j)}](\eta_j)\big)^{\mathbb{I}}$. Indeed, with $\chi_{i'} = [v_{i'}^{(j-1)}](\eta_{j-1})$, $i' = 1, \ldots, d_{j-1}$,

$$\begin{aligned}[v_i^{(j)}]^{\mathbb{I}}\big(\langle\!\langle v_1^{(j-1)} \rangle\!\rangle, \ldots, \langle\!\langle v_{d_{j-1}}^{(j-1)} \rangle\!\rangle\big) &= [v_i^{(j)}]^{\mathbb{I}}\big(\chi_1^{\mathbb{I}}, \ldots, \chi_{d_{j-1}}^{\mathbb{I}}\big) \\ &= \big([v_i^{(j)}](\zeta^{(j-1,j)} \bullet \eta_{j-1})\big)^{\mathbb{I}} = \big([v_i^{(j)}](\eta_j)\big)^{\mathbb{I}},\end{aligned}$$

the first equality by the induction assumption $\langle\!\langle v_{i'}^{(j-1)} \rangle\!\rangle = \chi_{i'}^{\mathbb{I}}$, the second by (9) with $\tau = [v_i^{(j)}]$, the substitution taking $x_{i'}$ to $\chi_{i'}$ being $\zeta^{(j-1,j)} \bullet \eta_{j-1}$ by Definition 12 with $\zeta^{(j-1,j)} = \big\{x_{i'} \mapsto [v_{i'}^{(j-1)}] : i' = 1, \ldots, d_{j-1}\big\}$ (Definition 11), and the third by the recursion of Proposition 2, the case $j = 1$ using $\zeta^{(0,1)} \bullet \eta_0 = \eta_1$. At $j = L$ this reads

$$\langle \mathcal{K} \rangle = \langle\!\langle v_{\text{out}} \rangle\!\rangle = \big([v_{\text{out}}](\eta_L)\big)^{\mathbb{I}} = [\mathcal{G}]^{\mathbb{I}},$$

with the first equality by Definition 8 and the third by (11). ■

Appendix D. Auxiliary results for Section 4

Composition of substitutions is not associative in general. For $\zeta = \{x_1 \mapsto x_2\}$, $\zeta' = \{x_3 \mapsto x_4\}$, and $\zeta'' = \{x_2 \mapsto x_5\}$, Definition 12 gives $(\zeta \bullet \zeta') \bullet \zeta'' = \{x_1 \mapsto x_5\}$ and $\zeta \bullet (\zeta' \bullet \zeta'') = \{x_1 \mapsto x_2\}$. The following lemma establishes associativity when every variable occurring in the substitutors of ζ lies in the domain of ζ'.

Lemma 8. *Let $\zeta = \{x_{i_1} \mapsto \chi_1, \ldots, x_{i_k} \mapsto \chi_k\}$, ζ', and ζ'' be substitutions such that the variables of $\chi_1, \ldots, \chi_k$ lie in the domain of ζ'. Then*

$$(\zeta \bullet \zeta') \bullet \zeta'' = \zeta \bullet (\zeta' \bullet \zeta'').$$

Proof The substitutor associated with x_{i_j} is $\big(\chi_j(\zeta')\big)(\zeta'')$ on the left-hand side and $\chi_j(\zeta' \bullet \zeta'')$ on the right-hand side; the two coincide by Lemma 2 applied to $\tau = \chi_j$. Both sides have domain $\{x_{i_1}, \ldots, x_{i_k}\}$ (Definition 12), so the two substitutions are equal. ■

Let $\mathcal{G}$ be a substitution graph of depth L with layer substitutions $\zeta^{(j-1,j)}$, $j = 1, \ldots, L$ (Definition 11). Equation (10) defines $[\mathcal{G}]$ by iterated substitution. We show that this iteration can be carried out in a single substitution, specifically that

$$[\mathcal{G}] = [v_{\text{out}}](\zeta^{(L-1,L)})(\zeta^{(L-2,L-1)}) \cdots (\zeta^{(0,1)}) = [v_{\text{out}}](\zeta^{(L-1,L)} \bullet \zeta^{(L-2,L-1)} \bullet \cdots \bullet \zeta^{(0,1)}). \tag{21}$$

By Definition 11 a level-j node formula has its variables among $x_1, \ldots, x_{d_{j-1}}$, the domain of $\zeta^{(j-1,j)}$, and the substitutors of $\zeta^{(j,j+1)}$ are exactly the level-j node formulae. The substitutors of $\zeta^{(j,j+1)} \bullet \zeta^{(j-1,j)}$ therefore have their variables among $x_1, \ldots, x_{d_{j-2}}$, the

domain of $\zeta^{(j-2,j-1)}$, so that the substitutor variables lie in the domain of the next layer substitution, and this recurs down the chain. Lemmata 2 and 8 thus apply at every stage. Lemma 2 turns each pair of successive applications $\big(\tau(\zeta)\big)(\zeta')$ into the single application $\tau(\zeta \bullet \zeta')$, so that repeating it from the left replaces the L applications in (10) by one application of a composition of all L layer substitutions. Lemma 8 identifies the left-nested composition so obtained, $\big(\cdots(\zeta^{(L-1,L)} \bullet \zeta^{(L-2,L-1)}) \bullet \cdots\big) \bullet \zeta^{(0,1)}$, with every other bracketing, so that the unbracketed chain above is unambiguous notation for their common value.

Lemma 9. *Let $\mathcal{G}$ be a substitution graph of depth $L \geq 2$ and let $k \in \{1, \ldots, L-1\}$. If $\mathcal{G}'$ is obtained from $\mathcal{G}$ by collapsing layer k (Definition 24), then $[\mathcal{G}'] = [\mathcal{G}]$.*

Proof Collapsing layer k replaces the formula of every level-$(k+1)$ node v by $[v](\zeta^{(k,k+1)})$ and joins levels $k-1$ and $k+1$. If $k+1 < L$, the layer substitution applied in $\mathcal{G}'$ to the level-$(k+2)$ formulae is no longer $\zeta^{(k+1,k+2)}$, the corresponding substitution of $\mathcal{G}$, but $\{x_i \mapsto [v_i^{(k+1)}](\zeta^{(k,k+1)})\} = \zeta^{(k+1,k+2)} \bullet \zeta^{(k,k+1)}$, while all other layer substitutions are unchanged. The chain of $\mathcal{G}'$ has therefore one factor fewer than that of $\mathcal{G}$, the pair $\zeta^{(k+1,k+2)}, \zeta^{(k,k+1)}$ having been replaced by its composite, and

$$[\mathcal{G}'] = [v_{\text{out}}]\big(\zeta^{(L-1,L)} \bullet \cdots \bullet \zeta^{(k+2,k+3)} \bullet (\zeta^{(k+1,k+2)} \bullet \zeta^{(k,k+1)}) \bullet \zeta^{(k-1,k)} \bullet \cdots \bullet \zeta^{(0,1)}\big) = [\mathcal{G}].$$

Here, the last equality holds because the composition substituted into $[v_{\text{out}}]$ carries the same L factors, in the same order, as the chain of $\mathcal{G}$, differing from it only by the bracket around $\zeta^{(k+1,k+2)} \bullet \zeta^{(k,k+1)}$, and all bracketings of the chain denote the same substitution by Lemma 8. If $k+1 = L$, the only level-$(k+1)$ node is v_{out} and there is no level $k+2$, so the collapse absorbs $\zeta^{(L-1,L)}$ into the output formula, replacing $[v_{\text{out}}]$ by $[v_{\text{out}}](\zeta^{(L-1,L)})$, and leaves the remaining layer substitutions unchanged. Hence

$$\begin{aligned}[\mathcal{G}'] &= \big([v_{\text{out}}](\zeta^{(L-1,L)})\big)\big(\zeta^{(L-2,L-1)} \bullet \cdots \bullet \zeta^{(0,1)}\big) \\ &= [v_{\text{out}}]\big(\zeta^{(L-1,L)} \bullet (\zeta^{(L-2,L-1)} \bullet \cdots \bullet \zeta^{(0,1)})\big) = [\mathcal{G}],\end{aligned}$$

where the second equality holds by Lemma 2 and the third because $\zeta^{(L-1,L)} \bullet (\zeta^{(L-2,L-1)} \bullet \cdots \bullet \zeta^{(0,1)})$ carries the same L factors in the same order as the chain of $\mathcal{G}$, now bracketed after the first, and all bracketings of the chain denote the same substitution by Lemma 8. ■

Lemma 10. *Let $\mathcal{G}$ be a substitution graph of depth L, let $k \in \{1, \ldots, L\}$, and let $\mathcal{G}'$ be obtained from $\mathcal{G}$ by expanding at level k (Definition 24), the $m \geq 1$ inserted nodes carrying the formulae $\chi_1, \ldots, \chi_m$ and every level-k node v of $\mathcal{G}$, which is of level $k+1$ in $\mathcal{G}'$, carrying in $\mathcal{G}'$ a formula $[v]'$ with $[v]'(\{x_1 \mapsto \chi_1, \ldots, x_m \mapsto \chi_m\}) = [v]$. Then $[\mathcal{G}'] = [\mathcal{G}]$.*

Proof The graph $\mathcal{G}'$ has depth $L+1$ and the inserted nodes constitute its level k, so we may collapse layer k of $\mathcal{G}'$ and apply Lemma 9. The collapse replaces the formula $[v]'$ of every level-$(k+1)$ node of $\mathcal{G}'$ by $[v]'(\zeta^{(k,k+1)})$, with $\zeta^{(k,k+1)}$ the layer substitution of $\mathcal{G}'$, and joins levels $k-1$ and $k+1$ (Definition 24). The substitutors of $\zeta^{(k,k+1)}$ are the formulae of the level-k nodes of $\mathcal{G}'$, so that $\zeta^{(k,k+1)} = \{x_1 \mapsto \chi_1, \ldots, x_m \mapsto \chi_m\}$ and $[v]'(\zeta^{(k,k+1)}) = [v]$. The collapse thus returns $\mathcal{G}$, and hence $[\mathcal{G}'] = [\mathcal{G}]$. ■

Proposition 6 (Soundness of node rewrites)**.** *Let $\mathcal{G}$ be a substitution graph, let $\mathcal{E}$ be a set of axioms, let $e \in \mathcal{E}$, and let $\mathcal{G}'$ be obtained from $\mathcal{G}$ by a node rewrite by e (Definition 23). Then $[\mathcal{G}] \overset{\mathcal{E}}{\sim} [\mathcal{G}']$.*

The proof of Proposition 6 is based on the following two lemmata.

Lemma 11. *Let e be an axiom and let $\zeta = \{x_{i_1} \mapsto \chi_1, \ldots, x_{i_k} \mapsto \chi_k\}$ be a substitution. Let τ be a formula with variables in $\{x_{i_1}, \ldots, x_{i_k}\}$, let $p \in \{1, \ldots, k\}$, and let χ'_p be a formula with $\chi_p \overset{e}{\sim} \chi'_p$. Set $\zeta' = \{x_{i_1} \mapsto \chi_1, \ldots, x_{i_p} \mapsto \chi'_p, \ldots, x_{i_k} \mapsto \chi_k\}$. Then $\tau(\zeta)$ is carried to $\tau(\zeta')$ by finitely many derivation steps, each by an application of e.*

Proof Let ℓ be the number of occurrences of x_{i_p} in τ, finite as τ is a finite string (Definition 2). The formulae $\tau(\zeta)$ and $\tau(\zeta')$ agree except at the ℓ positions of x_{i_p} in τ, occupied by χ_p in $\tau(\zeta)$ and by χ'_p in $\tau(\zeta')$. Replacing χ_p by χ'_p at these positions one at a time yields $\tau(\zeta) = \gamma^0, \gamma^1, \ldots, \gamma^\ell = \tau(\zeta')$. Each of these replacements, $\gamma^{q-1} \overset{e}{\sim} \gamma^q$, $q = 1, \ldots, \ell$, is a single application of e (Definition 20), hence ℓ such applications carry $\tau(\zeta)$ to $\tau(\zeta')$. ■

Lemma 12. *Let $\mathcal{E}$ be a set of axioms and let $e \in \mathcal{E}$. For formulae τ, τ' and a substitution ζ, if $\tau \overset{e}{\sim} \tau'$, then $\tau(\zeta) \overset{e}{\sim} \tau'(\zeta)$. Consequently, if $\tau \overset{\mathcal{E}}{\sim} \tau'$, then $\tau(\zeta) \overset{\mathcal{E}}{\sim} \tau'(\zeta)$.*

Proof Write the axiom e as the logic equation $\epsilon = \epsilon'$ between formulae ϵ, ϵ' (Definition 19). Since $\tau \overset{e}{\sim} \tau'$, there is an occurrence of a subformula γ of τ whose replacement by a formula γ' yields τ', with $\gamma = \gamma'$ an instantiation of e (Definition 20), that is, $\gamma = \epsilon(\zeta_*)$ and $\gamma' = \epsilon'(\zeta_*)$ for some substitution ζ_* (Definition 19). Extending ζ_* by the identity mapping on the variables of ϵ and ϵ' outside its domain leaves $\epsilon(\zeta_*)$ and $\epsilon'(\zeta_*)$ unchanged, so that the variables of ϵ and ϵ' may be assumed to lie in the domain of ζ_*. Applying ζ to τ turns this occurrence into an occurrence of $\big(\epsilon(\zeta_*)\big)(\zeta) = \epsilon(\zeta_* \bullet \zeta)$ in $\tau(\zeta)$ (Lemma 2). Since τ' carries $\epsilon'(\zeta_*)$ at this position and agrees with τ elsewhere, $\tau'(\zeta)$ carries $\big(\epsilon'(\zeta_*)\big)(\zeta) = \epsilon'(\zeta_* \bullet \zeta)$ (Lemma 2 again) at the same position and agrees with $\tau(\zeta)$ elsewhere. As $\epsilon(\zeta_* \bullet \zeta) = \epsilon'(\zeta_* \bullet \zeta)$ instantiates e, the replacement of $\epsilon(\zeta_* \bullet \zeta)$ by $\epsilon'(\zeta_* \bullet \zeta)$ is a single derivation step on $\tau(\zeta)$, so that $\tau(\zeta) \overset{e}{\sim} \tau'(\zeta)$. Definition 19 admits both orientations of an instantiation, and the case $\gamma = \epsilon'(\zeta_*)$, $\gamma' = \epsilon(\zeta_*)$ is treated identically. The second claim follows by applying the first to each step of any derivation $\tau = \tau_1 \overset{e_1}{\sim} \cdots \overset{e_{T-1}}{\sim} \tau_T = \tau'$ with $e_t \in \mathcal{E}$ (Definition 20). ■

Proof [Proof of Proposition 6] Let L be the depth of $\mathcal{G}$ and let the node rewrite replace the formula $[v]$ of the node v by τ' with $[v] \overset{e}{\sim} \tau'$. If $v = v_{\text{out}}$, then $[\mathcal{G}] = [v_{\text{out}}](Z)$ by (21) and $[\mathcal{G}'] = \tau'(Z)$, with the substitution $Z = \zeta^{(L-1,L)} \bullet \cdots \bullet \zeta^{(0,1)}$, and $[\mathcal{G}] \overset{e}{\sim} [\mathcal{G}']$ by Lemma 12. Otherwise, let v be the i-th node at level k, $1 \le k \le L-1$, and set $\beta = [v_{\text{out}}](\zeta^{(L-1,L)} \bullet \cdots \bullet \zeta^{(k+1,k+2)})$ (for $k = L-1$, $\beta = [v_{\text{out}}]$), a formula in the level-k variables, so that, by Lemmata 2 and 8,

$$[\mathcal{G}] = \beta(\zeta^{(k,k+1)})(\zeta^{(k-1,k)}) \cdots (\zeta^{(0,1)}). \tag{22}$$

Let $\widetilde{\zeta}$ be obtained from

$$\zeta^{(k,k+1)} = \{x_1 \mapsto [v_1^{(k)}], \ldots, x_{d_k} \mapsto [v_{d_k}^{(k)}]\}$$

by replacing the substitutor of x_i, which is $[v_i^{(k)}] = [v]$, by τ', so that

$$[\mathcal{G}'] = \beta(\widetilde{\zeta})(\zeta^{(k-1,k)}) \cdots (\zeta^{(0,1)}). \tag{23}$$

By Lemma 11, $\beta(\zeta^{(k,k+1)})$ is carried to $\beta(\widetilde{\zeta})$ by finitely many derivation steps, each by an application of e, and hence $\beta(\zeta^{(k,k+1)}) \overset{\mathcal{E}}{\sim} \beta(\widetilde{\zeta})$ as $e \in \mathcal{E}$. Applying Lemma 12 with $\tau = \beta(\zeta^{(k,k+1)})$, $\tau' = \beta(\widetilde{\zeta})$, and $\zeta = \zeta^{(k-1,k)}$, then with $\zeta = \zeta^{(k-2,k-1)}$, and so on down to $\zeta^{(0,1)}$, gives $\beta(\zeta^{(k,k+1)})(\zeta^{(k-1,k)}) \cdots (\zeta^{(0,1)}) \overset{\mathcal{E}}{\sim} \beta(\widetilde{\zeta})(\zeta^{(k-1,k)}) \cdots (\zeta^{(0,1)})$. The left-hand side is $[\mathcal{G}]$ by (22) and the right-hand side is $[\mathcal{G}']$ by (23), so that $[\mathcal{G}] \overset{\mathcal{E}}{\sim} [\mathcal{G}']$. ■