CLN2INV: LEARNING LOOP INVARIANTS WITH CONTINUOUS LOGIC NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Program verification offers a framework for ensuring program correctness and therefore systematically eliminating different classes of bugs. Inferring loop invariants is one of the main challenges behind automated verification of real-world programs which often contain many loops. In this paper, we present Continuous Logic Network (CLN), a novel neural architecture for automatically learning loop invariants directly from program execution traces. Unlike existing neural networks, CLNs can learn precise and explicit representations of formulas in Satisfiability Modulo Theories (SMT) for loop invariants from program execution traces. We develop a new sound and complete semantic mapping for assigning SMT formulas to continuous truth values that allows CLNs to be trained efficiently. We use CLNs to implement a new inference system for loop invariants, CLN2INV, that significantly outperforms existing approaches on the popular Code2Inv dataset. CLN2INV is the first tool to solve all 124 theoretically solvable problems in the Code2Inv dataset. Moreover, CLN2INV takes only 1.1 second on average for each problem, which is 40× faster than existing approaches. We further demonstrate that CLN2INV can even learn 12 significantly more complex loop invariants than the ones required for the Code2Inv dataset.

1 INTRODUCTION

Program verification offers a principled approach for systematically eliminating different classes of bugs and proving the correctness of programs. However, as programs have become increasingly complex, real-world program verification often requires prohibitively expensive manual effort (Wilcox et al., 2015; Gu et al., 2016; Chajed et al., 2019). Recent efforts have focused on automating the program verification process, but automated verification of general programs with unbounded loops remains an open problem (Nelson et al., 2017; 2019).

Verifying programs with loops requires determining *loop invariants*, which captures the effect of the loop on the program state irrespective of the actual number of loop iterations. Automatically inferring correct loop invariants is a challenging problem that is undecidable in general and difficult to solve in practice (Blass & Gurevich, 2001; Furia et al., 2014). Existing approaches use stochastic search (Sharma & Aiken, 2016), heurstics-based search (Galeotti et al., 2015), PAC learning based on counter examples (Padhi & Millstein, 2017), or reinforcement learning (Si et al., 2018). However, these approaches often struggle to learn complex, real-world loop invariants.

In this paper, we introduce a new approach to learning loop invariants by modeling the loop behavior from program execution traces using a new type of neural architecture. We note that inferring loop invariants can be posed as learning formulas in Satisfiability Modulo Theories (SMT) (Biere et al., 2009) over program variables collected from program execution traces (Nguyen et al., 2017). In principle, Neural networks seem well suited to this task because they can act as universal function approximators and have been successfully applied in various domains that require modeling of arbitrary functions (Hornik et al., 1989; Goodfellow et al., 2016). However, loop invariants must be represented as explicit SMT formulas to be usable for program verification. Unfortunately, existing methods for extracting logical rules from general neural architectures lack sufficient precision (Augasta & Kathirvalavakumar, 2012), while inductive logic learning lacks sufficient expressiveness for use in verification (Evans & Grefenstette, 2018).

We address this issue by developing a novel neural architecture, Continuous Logic Network (CLN), which is able to efficiently learn explicit and precise representations of SMT formulas by using continuous truth values. Unlike existing neural architectures, CLNs can represent a learned SMT formula explicitly in its structure and thus allow us to precisely extract the exact formula from a trained model.

In order to train CLNs, we introduce a new semantic mapping for SMT formulas to continuous truth values. Our semantic mapping builds on BL, or basic fuzzy logic (Hájek, 2013), to support general SMT formulas in a continuous logic setting. We further prove that our semantic model is sound (i.e., truth assignments for the formulas are consistent with their discrete counterparts) and complete (i.e., all formulas can be represented) with regard to the discrete SMT formula space. These properties allow CLNs to represent any quantifier-free SMT formula operating on mixed integer-real arithmetic as an end-to-end differentiable series of operations.

We use CLNs to implement a new inference system for loop invariants, CLN2INV, that significantly outperforms state-of-the-art tools on the Code2Inv dataset by solving all 124 theoretically solvable problems in the dataset. This is 20 problems more than LoopInvGen, the winner of the SyGus 2018 competition loop invariant track (Si et al., 2018). Moreover, CLN2INV finds invariants for each program in 1.1 second on average, more than 40 times faster than LoopInvGen. We also demonstrate CLN2INV is able to learn complex, real-world loop invariants with combinations of conjunctions and disjunctions of multivariable constraints.

Our main contributions are:

- We introduce a new semantic mapping for assigning continuous truth values to SMT formulas that is theoretically grounded and enables learning formulas through backpropagation. We further prove that our semantic model is sound and complete.
- We develop a novel neural architecture, Continuous Logic Networks (CLNs), that to the best of our knowledge is the first to efficiently learn precise and explicit SMT formulas by construction.
- We use CLNs to implement a new loop invariant inference system, CLN2INV, that is the first to solve all 124 theoretically solvable problems in the Code2Inv dataset, 20 more than the existing methods. CLN2INV is able to find invariants for each problem in 1.1 second on average, $40\times$ faster than existing systems.
- We further show CLN2INV is able to learn 12 more complex loop invariants than the ones present in the Code2Inv dataset with combinations of multivariable constraints.

Related Work. Traditionally, loop invariant learning has relied on stochastic or heuristics-guided search (Sharma & Aiken, 2016; Galeotti et al., 2015). Other approaches like NumInv analyze traces and discover simple conjunctions of equalities by solving a system of linear equations (Nguyen et al., 2017). LoopInvGen uses PAC learning of CNF using counter-examples (Padhi & Millstein, 2017). By contrast, Code2Inv learns to guess loop invariants using reinforcement learning with recurrent and graph neural networks (Si et al., 2018). However, these approaches struggle to learn complex invariants. Unlike these works, CLN2INV can efficiently learn complex invariants directly from execution traces.

There is a long line of work on PAC learning of boolean formulas, but learning precise formulas require a prohibitively large number of samples (Kearns et al., 1994). Several recent works use different forms of differentiable logic to learn boolean logic formulas from noisy data (Kimmig et al., 2012; Evans & Grefenstette, 2018; Payani & Fekri, 2019) or improving adversarial robustness by applying logical rules to training (Fischer et al., 2019). By contrast, our work learns precise SMT formulas directly by construction, allowing us to learn richer predicates with compact representation in a noiseless setting.

2 Background

In this section, we introduce the problem of inferring loop invariants and provide a brief overview of Satisfiability Modulo Theories (SMT), which are used to represent loop invariants. We then provide an introduction to basic fuzzy logic, which we later extend to formulate our new continuous semantic mapping for SMT.

2.1 LOOP INVARIANTS

Loop invariants capture loop behavior irrespective of number of iterations, which is crucial for verifying programs with loops. Given a loop, `while(`LC`)`$\{C\}$, a precondition P, and a post-condition Q, the verification task involves finding a loop invariant I that can be concluded from the pre-condition and implies the post-condition (Hoare, 1969). Formally, it must satisfy the following three conditions, in which the second is a Hoare triple describing the loop:

$$P \implies I \qquad \{I \wedge LC\}\, C\, \{I\} \qquad \neg LC \wedge I \implies Q$$

Example of Loop Invariant. Consider the example loop in Fig.1. For a loop invariant to be usable, it must be valid for the precondition $(t = 10 \wedge u = 0)$, the recursion step when $t \neq 0$, and the post condition $(u = 20)$ when the loop condition is no longer satisfied, i.e., $t = 0$. The correct and precise invariant I for the program is $(2t + u = 20)$.

```
//pre: t=10 /\ u=0
while (t != 0){
  t = t - 1;
  u = u + 2;
}
//post: u=20
```

(a) Example loop

The desired loop invariant I for the left program is a boolean function over program variables t, u such that:

$$\forall t\, u, \begin{cases} t = 10 \wedge u = 0 & \implies I(t,u) & (pre) \\ I(t,u) \wedge (t \neq 0) & \implies I(t-1, u+2) & (inv) \\ I(t,u) \wedge (t = 0) & \implies u = 20 & (post) \end{cases}$$

(b) The desired and precise loop invariant I is $(2t + u = 20)$.

Figure 1: Example Loop Invariant inference problem.

2.2 SATISFIABILITY MODULO THEORIES

Satisfiability Modulo Theories (SMT) are an extension of Boolean Satisfiability that allow solvers to reason about complex problems efficiently. Loop invariants and other formulas in program verification are usually encoded with quantifier-free SMT. A formula F in quantifier-free SMT can be inductively defined as below:

$$F := E_1 \bowtie E_2 \mid \neg F \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \qquad \bowtie \in \{=, \neq, <, >, \leq, \geq\}$$

where E_1 and E_2 are expressions of terms. The loop invariant $(2t + u = 20)$ in Fig. 1 is an SMT formula. Nonlinear arithmetic theories admit higher-order terms such as t^2 and $t * u$, allowing them to express more complex constraints. For example, $(\neg(2 \geq t^2))$ is an SMT formula that is true when the value of the high-order term t^2 is larger than 2.

2.3 BASIC FUZZY LOGIC (BL)

Basic fuzzy logic (BL) is a class of logic that uses continuous truth values in the range $[0, 1]$ and is differentiable almost everywhere[1] (Hájek, 2013). BL defines logical conjunction with functions called *t-norms*, which must satisfy specific conditions to ensure that the behavior of the logic is consistent with boolean First Order Logic. Formally, a t-norm (denoted $\otimes$) in BL is a binary operator over truth values in the interval $[0, 1]$ satisfying the following conditions:

1) *commutativity* and *associativity*: the order in which a set of t-norms on continuous truth values are evaluated should not change the result: $x \otimes y = y \otimes x \quad x \otimes (y \otimes z) = (x \otimes y) \otimes z$.

2) *monotonicity*: increasing any input value to a t-norm operation should not cause the result to decrease: $x_1 \leq x_2 \implies x_1 \otimes y \leq x_2 \otimes y$

3) *consistency*: the result of any t-norm applied to a truth value and 1 should be 1, and the result of any truth value and 0 should be 0: $1 \otimes x = x$. $0 \otimes x = 0$

Besides these conditions, BL also requires that t-norms be continuous.

[1] Almost everywhere indicates the function is differentiable everywhere except for a set of measure 0. For example, a Rectified Linear Unit is differentiable almost everywhere except at zero.

3 CONTINUOUS SATISFIABILITY MODULO THEORIES

We introduce a *continuous semantic mapping*, $\mathcal{S}$, for SMT on BL that is end-to-end differentiable. The mapping $\mathcal{S}$ associates SMT formulas with continuous truth values while preserving each formula's semantics. In this paper, we only consider quantifier-free formulas. This process is analogous to constructing t-norms for BL, where a t-norm operates on continuous logical inputs.

We define three desirable properties for continuous semantic mapping $\mathcal{S}$ that will preserve formula semantics while facilitating parameter training with gradient descent:

1. $\mathcal{S}(F)$ should be consistent with BL. For any two formulas F and F', where $F(x)$ is satisfied and $F'(x)$ is unsatisfied with an assignment x of formula terms, we should have $\mathcal{S}(F')(x) < \mathcal{S}(F)(x)$. This will ensure the semantics of SMT formulas are preserved.
2. $\mathcal{S}(F)$ should be differentiable almost everywhere. This will facilitate training with gradient descent through backpropogation.
3. $\mathcal{S}(F)$ should be increasing everywhere as the terms in the formula approach constraint satisfaction, and decreasing everywhere as the terms in the formula approach constraint violation. This ensures there is always a nonzero gradient for training.

Continuous semantic mapping. We first define the mapping for "$>$" (greater-than) and "$\geq$" (greater-than-or-equal-to) as well as adopting definitions for "$\neg$", "$\wedge$", and "$\vee$" from BL. All other operators can be derived from these. For example, "$\leq$" (less-than-or-equal-to) is derived using "$\geq$" and "$\neg$", while "$=$" (equality) is then defined as the conjunction of formulas using "$\leq$" and "$\geq$." Given constants $B > 0$ and $\epsilon > 0$, we first define the the mapping $\mathcal{S}$ on "$>$" and "$\geq$" using shifted and scaled sigmoid functions:

$$\mathcal{S}(t > u) \triangleq \frac{1}{1 + e^{-B(t-u-\epsilon)}} \qquad \mathcal{S}(t \geq u) \triangleq \frac{1}{1 + e^{-B(t-u+\epsilon)}}$$

Illustrations of these functions are given in Appendix A. The validity of our semantic mapping lie in the following facts, which can be proven with basic algebra.

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} \frac{1}{1 + e^{-B(t-u-\epsilon)}} = \begin{cases} 1 & t > u \\ 0 & t \leq u \end{cases} \qquad \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} \frac{1}{1 + e^{-B(t-u+\epsilon)}} = \begin{cases} 1 & t \geq u \\ 0 & t < u \end{cases}$$

When ϵ goes to zero and $B * \epsilon$ goes to infinity, our continuous mapping of "$>$" and "$\geq$" will preserve their original semantics. Under these conditions, our mapping satisfies all three desirable properties. In practice, for small ϵ and large B, the properties are also satisfied if $|t - u| > \epsilon$.

Next we define the mapping $\mathcal{S}$ for boolean operators "$\wedge$", "$\vee$" and "$\neg$" using BL. Recall that in BL, a t-norm is a continuous function that behaves like logical conjunction. In §2.3, we outlined requirements for a valid t-norm. Three widely used t-norms that satisfy the requirements are the Lukaseiwicz t-norm (Lukasiewicz, 1930), the Godel t-norm (Baaz et al., 1996), and the product t-norm (Hájek et al., 1996). Each t-norm has a *t-conorm* associated with it (denoted $\oplus$), which can be considered as logical disjunction. Given a t-norm $\otimes$, the t-conorm can be derived with DeMorgan's law: $t \oplus u \triangleq \neg(\neg t \otimes \neg u)$.

	Lukasiewicz :	Godel:	Product:
t-norm($\otimes$)	$max(0, t + u - 1)$	$min(t, u)$	$t * u$
t-conorm($\oplus$)	$min(t + u, 1)$	$max(t, u)$	$t + u - t * u$

Given a specific t-norm $\otimes$ and its corresponding t-conorm $\oplus$, it is straightforward to define mappings of "$\wedge$", "$\vee$" and "$\neg$":

$$\mathcal{S}(F_1 \wedge F_2) \triangleq \mathcal{S}(F_1) \otimes \mathcal{S}(F_2) \qquad \mathcal{S}(F_1 \vee F_2) \triangleq \mathcal{S}(F_1) \oplus \mathcal{S}(F_2) \qquad \mathcal{S}(\neg F) \triangleq 1 - \mathcal{S}(F)$$

Based on the above definitions, the mapping for other operators can be derived as follows:

$$\mathcal{S}(t < u) = \mathcal{S}(\neg(t \geq u)) = \frac{1}{1 + e^{B(t-u+\epsilon)}} \qquad \mathcal{S}(t \leq u) = \mathcal{S}(\neg(t > u)) = \frac{1}{1 + e^{B(t-u-\epsilon)}}$$

$$\mathcal{S}(t = u) = \mathcal{S}((t \geq u) \wedge (t \leq u)) = \frac{1}{1 + e^{-B(t-u+\epsilon)}} \otimes \frac{1}{1 + e^{B(t-u-\epsilon)}}$$

The mapping $\mathcal{S}$ on "=" is valid since the following limit holds (see Appendix B for the proof).

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} \mathcal{S}(t = u) \quad = \quad \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} \frac{1}{1 + e^{-B(t-u+\epsilon)}} \otimes \frac{1}{1 + e^{B(t-u-\epsilon)}} = \begin{cases} 1 & t = u \\ 0 & t \neq u \end{cases}$$

The mapping for other operators shares similar behavior in the limit, and also fulfill our desired properties under the same conditions.

Using our semantic mapping $\mathcal{S}$, most of the standard operations of integer and real arithmetic, including addition, subtraction, multiplication, division, and exponentiation, can be used normally and mapped to continuous truth values while keeping the entire formula differentiable. Moreover, any expression in SMT that has an integer or real-valued result can be mapped to continuous logical values via these formulas, although end-to-end differentiability may not be maintained in cases where specific operations are nondifferentiable.

4 Continuous Logic Networks

In this section, we describe the construction of Continuous Logic Networks (CLNs) based on our continuous semantic mapping for SMT on BL.

CLN Construction. CLNs use our continuous semantic mapping to represent SMT formulas as an end-to-end differentiable model that can learn selected constants in the formula. When constructing a CLN, we work from an *SMT Formula Template*, in which every value is marked as either an input term, a constant, or a learnable parameter. Given an SMT Formula Template, we dynamically construct a CLN as a computational graph, where input terms are marked as model inputs. The operations in each SMT clause are recursively added to the graph, followed by logical operations on clauses. Figure 2 shows an example formula template and the constructed CLN. We denote the CLN model constructed from the formula template $\mathcal{S}(F)$ as M_F.

CLN Training. Once the CLN has been constructed based on a formula template, it is trained with the following optimization. Given a CLN model M constructed from an SMT template with learnable parameters $\mathbf{W}$, and a set $\mathbf{X}$ of valid assignments for the terms in the SMT template, the expected value of the CLN is maximized by minimizing a loss function $\mathcal{L}$ that penalizes model outputs that are less than one. A minimum scaling factor β is selected, and a hinge loss is applied to the scaling factors (B) to force the differentiable predicates to approach sharp cutoffs. The offset ϵ is also regularized to ensure precision. The overall optimization is formulated as:

$$\max_{\{\mathbf{W}, B, \epsilon\}} \mathbb{E}[M(\mathbf{X}; \mathbf{W}, B, \epsilon)] = \min_{\{\mathbf{W}, B, \epsilon\}} \sum_{\boldsymbol{x} \in \mathbf{X}} \mathcal{L}(M(\boldsymbol{x}; \mathbf{W}, B, \epsilon)) + \lambda \sum_{B \in \mathbf{B}} \mathcal{L}_{hinge}(\beta, B) + \gamma ||\epsilon||_2$$

where λ and γ are hyperparameters respectively governing the weight assigned to the scaling factor and offset regularization. $\mathcal{L}_{hinge}(\beta, B)$ is defined as $max(0, \beta - B)$, and $\mathcal{L}$ is any loss function strictly decreasing in domain $[0, 1]$.

Given a CLN that has been trained to a loss approaching 0 on a given set of valid assignments, we now show that the resulting continuous SMT formula learned by the CLN is consistent with an equivalent boolean SMT formula. In particular, we prove that continuous SMT formulas learned with CLNs are *sound* and *complete* with regard to SMT formulas on discrete logic. We further prove that a subset of SMT formulas are guaranteed to converge to a globally optimal solution.

Soundness. Given the SMT formula F, the CLN model M_F constructed from $\mathcal{S}(F)$ always preserves the truth value of F. It indicates that given a valid assignment to the terms $\boldsymbol{x}$ in F, $F(\boldsymbol{x}) = True \iff M_F(\boldsymbol{x}) = 1$ and $F(\boldsymbol{x}) = False \iff M_F(\boldsymbol{x}) = 0$.

Completeness. For any SMT formula F, a CLN model M can be constructed representing that formula. In other words, CLNs can express all SMT formulas on integers and reals.

We formally state these properties in Theorem 1 and provide a proof by induction on the constructor in the Appendix C. Before that we need to define a property for t-norms.

Property 1. $\forall t\ u, (t > 0)$ and $(u > 0)$ implies $(t \otimes u > 0)$.

The product t-norm and Godel t-norm have this property, while the Lukasiewicz t-norm does not.

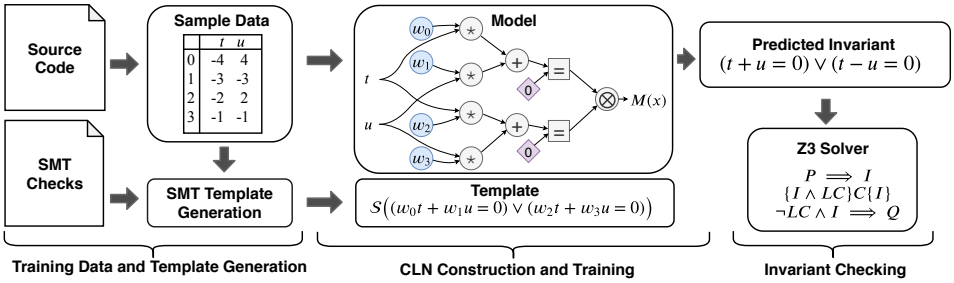


Figure 2: System architecture and CLN construction from SMT templates.

Theorem 1. *For any SMT formula F, there exists a CLN model M, such that*

$$\forall \boldsymbol{x}\ B\ \epsilon,\ 0 \leq M(\boldsymbol{x}; B, \epsilon) \leq 1$$

$$F(\boldsymbol{x}) = True \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1 \qquad F(\boldsymbol{x}) = False \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 0$$

as long as the t-norm used in building M satisfies Property 1.

Optimality. For a subset of SMT formulas (conjunctions of multiple linear equalities), CLNs are guaranteed to converge at the global minumum. We formally state this in Theorem 2 and the proof can be found in Appendix D. We first define another property similar to strict monotonicity.

Property 2. $\forall t_1\ t_2\ t_3,\ (t_1 < t_2)$ and $(t_3 > 0)$ implies $(t_1 \otimes t_3 < t_2 \otimes t_3)$.

Theorem 2. *For any CLN model M_F constructed from a formula, F, by the procedure shown in the proof of Theorem 1, if F is the conjunction of multiple linear equalities then any local minimum of M_F is the global minimum, as long as the t-norm used in building M_F satisfies Property 2.*

5 Loop Invariant Learning

We use CLNs to implement a new inference system for loop invariants, CLN2INV, which learns invariants directly from execution traces. Figure 2 provides an overview of the architecture.

Training Data Generation. We generate training data by running the program repeatedly on a set of randomly initialized inputs that satisfy the preconditions. Unconstrained variables are initialized from a uniform distribution, and variables with precondition constraints are initialized from a uniform distribution within their constraints. All program variables are recorded before each execution of the loop and after the loop terminates.

Template Generation. We encode the template using information gathered through static analysis. We collect useful information such as constants found in the program code along with the termination condition. Our analysis also strengthens the precondition and weakens the post-condition to constrain the problem as tightly as possible. For instance, unconstrained variables can be constrained to ensure the loop executes. In Appendix E, we prove this approach maintains soundness as solutions to the constrained problem can be used to reconstruct full solutions.

We generate bounds for individual variables (e.g., $t \geq b$) as well as multivariable polynomial constraints (e.g., $w_1 t_1 + w_2 t_2 = b$). Constants are optionally placed in constraints based on the static analysis and execution data (i.e. if a variable is initialized to a constant and never changes). We then compose template formulas from the collection of constraints by selecting a subset and joining them with conjunctions or disjunctions.

CLN Construction and Training. Once a template formula has been generated, a CLN is constructed from the template using the formulation in §4. As an optimization, we represent equality constraints as Gaussian-like functions that retain a global maximum when the constraint is satisfied as discussed in Appendix F. We then train the model using the collected execution traces.

Invariant Checking. Invariant checking is performed using SMT solvers such as Z3 (De Moura & Bjørner, 2008). After the CLN for a formula template has been trained, the SMT formula for the loop invariant is recovered by normalizing the learned parameters. The invariant is checked against the pre, post, and recursion conditions as described in §2.1.

6 EXPERIMENTS

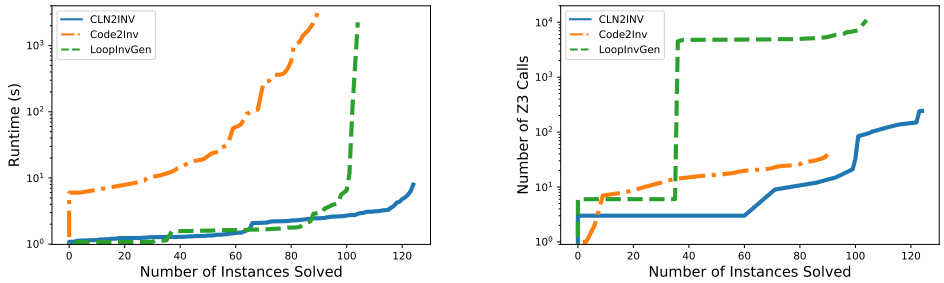
We compare the performance of CLN2INV with two existing methods and demonstrate the efficacy of the method on several more difficult problems. Finally, we conduct two ablation studies to justify our design choices.

Test Environment. All experiments are performed on an Ubuntu 18.04 server with an Intel Xeon E5-2623 v4 2.60GHz CPU, 256Gb of memory, and an Nvidia GTX 1080Ti GPU.

System Configuration. We implement CLNs in PyTorch and use the Adam optimizer for training with learning rate 0.01 (Paszke et al., 2017; Kingma & Ba, 2014). Because the performance of CLN is dependent on weight initialization, the CLN training randomly restart if the model does not reach termination within $2,000$ epochs. Learnable parameters are initialized from a uniform distribution in the range [-1, 1], which we found works well in practice.

Test Dataset. We use the same benchmark used in the evaluation of Code2Inv. We have removed nine invalid programs from Code2Inv's benchmark and test on the remaining 124. The removed programs are invalid because there are inputs which satisfy the precondition but result in a violation of the post-condition after the loop execution. The benchmark consists of loops expressed as C code and corresponding SMT files. Each loop can have nested if-then-else blocks (without nested loops). Programs in the benchmark may also have uninterpreted functions (emulating external function calls) in branches or loop termination conditions.

6.1 COMPARISON TO EXISTING SOLVERS



(a) Runtime performance.

(b) SMT solver calls.

Figure 3: Performance evaluation.

Performance Evaluation. We compare CLN2INV to two state-of-the-art methods: Code2Inv (based on neural code representation and reinforcement learning) and LoopInvGen (PAC learning over synthesized CNF formulas) (Si et al., 2018; Padhi et al., 2016). We limit each method to one hour per problem in the same format as the SyGuS Competition (Alur et al., 2019). Table 1 summarizes the results of the evaluation. CLN2INV is able to solve all **124** problems in the benchmark. LoopInvGen solves **104** problems while Code2inv solves **90**.[2]

Runtime Comparison. Figure 3a shows the measured runtime on each evaluated system. CLN2INV solves problems in **1.1** second on average, which is over **40**$\times$ faster than LoopInvGen, the second fastest system in the evaluation. In general, CLN2INV has similar performance to LoopInvGen on simple problems but is able to scale efficiently to complex problems.

Z3 Solver Utilization. Figure 3b shows the number of Z3 calls made by each method. For almost all problems, CLN2INV requires fewer Z3 calls than the other systems, although for some difficult problems it uses more Z3 calls than Code2Inv. While CLN2INV makes roughly twice as many Z3 solver calls as Code2Inv on average, it is able to generate and test candidate loop invariants over 250$\times$ faster on average.

Performance Summary Table 1 summarizes results of the performance evaluation. CLN2INV has the lowest time spent per problem making it the most practical approach. Code2Inv require

[2]The Code2Inv authors originally reported solving 92 problems with the same one hour timeout. We believe that the difference might be caused by changes in the testing environment or randomized model initialization.

Table 1: Results and summary statistics for performance evaluation.

Method	Number Solved	Avg Time (s)	Avg Z3 Calls	Time/Z3 Call (s)
Code2Inv	90	266.71	16.62	50.89
LoopInvGen	104	45.11	3,605.43	0.08
CLN2INV	124	1.07	31.77	0.17

more time on average per problem, but minimizes the number of calls made to an SMT solver. LoopInvGen is efficient at generating a large volume of guesses for the SMT solver. CLN2INV achieves a balance by producing quality guesses quickly allowing it to solve problems efficiently.

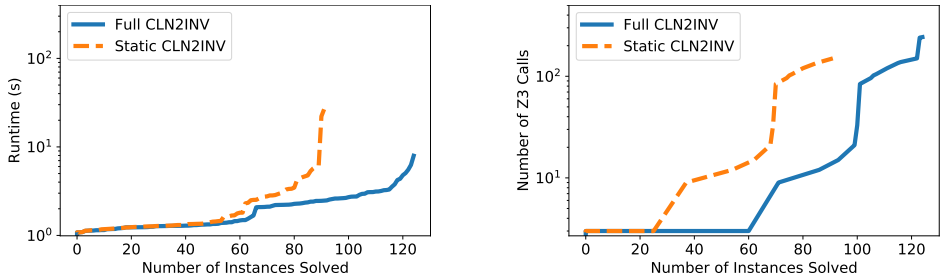
6.2 More Difficult Loop Invariants

We construct 12 additional problems to demonstrate CLN2INV's ability to infer complex loop invariants. We design these problems to have two key characteristics, which are absent in the Code2Inv dataset: (i) they require invariants involving conjunctions and disjunctions of multivariable constraints, and (ii) the invariant cannot easily be identified by inspecting the precondition, termination condition, or post-condition. CLN2INV is able to find correct invariants for all 12 problems in less than 20 seconds, while Code2Inv and LoopInvGen time out after an hour (see Appendix G).

6.3 Ablation Studies

Comparing T-norms. We compare t-norms (Godel, Lukasiewicz, and product) and conclude that product t-norm has the best performance. See Appendix H for more details.

Effect of CLN Training on Performance. CLN2INV relies on a combination of heuristics using static analysis and learning formulas from execution traces to correctly infer loop invariants. In this ablation we disable model training and limit CLN2INV to static models with no learnable parameters. The static CLN2INV solves 91 problems in the dataset. Figure 4 shows a comparison of full CLN2INV with one limited to static models. CLN2INV's performance with training disabled shows that a large number of problems in the dataset are relatively simple and invariants can be inferred from basic heuristics. However, for more difficult problems, the ability of CLNs to learn SMT formulas is key to successfully finding correct invariants.



(a) Runtime performance.

(b) SMT solver calls.

Figure 4: Ablation study comparing static vs trained models.

7 Conclusion

We develop a novel neural architecture that explicitly and precisely learns SMT formulas by construction. We achieve this by introducing a new sound and complete semantic mapping for SMT that enables learning formulas through backpropagation. We use CLNs to implement a loop invariant inference system, CLN2INV, that is the first to solve all theoretically solvable problems in the Code2Inv benchmark and takes only 1.1 second on average. We believe that the CLN architecture will also be beneficial for other domains that require learning SMT formulas.

REFERENCES

Rajeev Alur, Dana Fisman, Saswat Padhi, Rishabh Singh, and Abhishek Udupa. Sygus-comp 2018: Results and analysis. *CoRR*, abs/1904.07146, 2019. URL `http://arxiv.org/abs/1904.07146`.

M. Gethsiyal Augasta and T. Kathirvalavakumar. Rule extraction from neural networks a comparative study. *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*, pp. 404–408, 2012.

Matthias Baaz et al. Infinite-valued gödel logics with 0-1-projections and relativizations. In *Gödel'96: Logical foundations of mathematics, computer science and physics—Kurt Gödel's legacy, Brno, Czech Republic, August 1996, proceedings*, pp. 23–33. Association for Symbolic Logic, 1996.

A. Biere, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009. ISBN 1586039296, 9781586039295.

Andreas Blass and Yuri Gurevich. Inadequacy of computable loop invariants. *ACM Transactions on Computational Logic (TOCL)*, 2(1):1–11, 2001.

Tej Chajed, Joseph Tassarotti, M. Frans Kaashoek, and Nickolai Zeldovich. Verifying concurrent, crash-safe systems with perennial. In *Proceedings of the 27th Symposium on Operating Systems Principles*. ACM, 2019.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*, pp. 1931–1941, 2019.

Carlo A Furia, Bertrand Meyer, and Sergey Velder. Loop invariants: Analysis, classification, and examples. *ACM Computing Surveys (CSUR)*, 46(3):34, 2014.

Juan P Galeotti, Carlo A Furia, Eva May, Gordon Fraser, and Andreas Zeller. Inferring loop invariants by mutation, dynamic analysis, and static checking. *IEEE Transactions on Software Engineering*, 41(10):1019–1037, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan Newman Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. Certikos: An extensible architecture for building certified concurrent os kernels. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 653–669, 2016.

Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.

Petr Hájek, Lluís Godo, and Francesc Esteva. A complete many-valued logic with product-conjunction. *Archive for mathematical logic*, 35(3):191–208, 1996.

Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Michael J Kearns, Umesh Virkumar Vazirani, and Umesh Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, pp. 1–4, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jan Lukasiewicz. Untersuchungen uber den aussagenkalkul. *CR des seances de la Societe des Sciences et des Letters de Varsovie, cl. III*, 23, 1930.

Luke Nelson, Helgi Sigurbjarnarson, Kaiyuan Zhang, Dylan Johnson, James Bornholt, Emina Torlak, and Xi Wang. Hyperkernel: Push-button verification of an os kernel. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 252–269. ACM, 2017.

Luke Nelson, James Bornholt, Ronghui Gu, Andrew Baumann, Emina Torlak, and Xi Wang. Scaling symbolic evaluation for automated verification of systems code with serval. In *Symposium on Operating Systems Principles*, 2019.

ThanhVu Nguyen, Timos Antonopoulos, Andrew Ruef, and Michael Hicks. Counterexample-guided approach to finding numerical invariants. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pp. 605–615. ACM, 2017.

Saswat Padhi and Todd D. Millstein. Data-driven loop invariant inference with automatic feature synthesis. *CoRR*, abs/1707.02029, 2017. URL `http://arxiv.org/abs/1707.02029`.

Saswat Padhi, Rahul Sharma, and Todd Millstein. Data-driven precondition inference with learned features. *ACM SIGPLAN Notices*, 51(6):42–56, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Ali Payani and Faramarz Fekri. Inductive logic programming via differentiable deep neural logic networks. *arXiv preprint arXiv:1906.03523*, 2019.

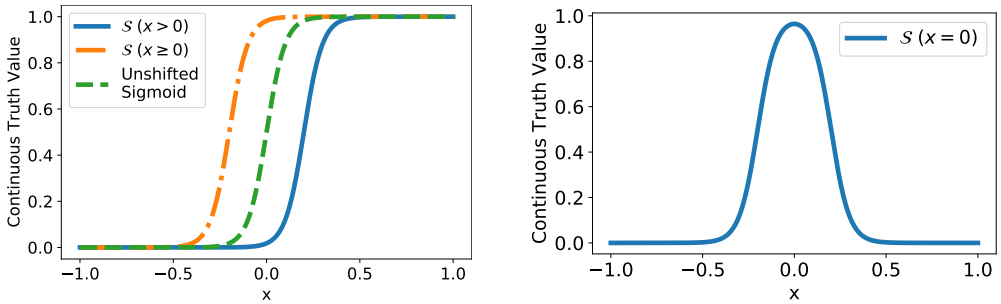
Rahul Sharma and Alex Aiken. From invariant checking to invariant inference using randomized search. *Formal Methods in System Design*, 48(3):235–256, 2016.

Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Advances in Neural Information Processing Systems*, pp. 7751–7762, 2018.

James R Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D Ernst, and Thomas Anderson. Verdi: a framework for implementing and formally verifying distributed systems. *ACM SIGPLAN Notices*, 50(6):357–368, 2015.

A CONTINUOUS PREDICATES

Figure 5 shows examples of shifted sigmoids for $\mathcal{S}(>)$, $\mathcal{S}(\geq)$, and $\mathcal{S}(=)$.



(a) Plot of $\mathcal{S}(x \geq 0)$, $\mathcal{S}(x > 0)$ with sigmoid

(b) Plot of $\mathcal{S}(x = 0)$ with product t-norm

Figure 5: Illustration of the mapping $\mathcal{S}$ on $>, \geq, =$ when $B = 20$ and $\epsilon = 0.2$

B PROOF OF LIMIT OF $\mathcal{S}(t = u)$

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} \mathcal{S}(t = u) \quad = \quad \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} \frac{1}{1 + e^{-B(t-u+\epsilon)}} \otimes \frac{1}{1 + e^{B(t-u-\epsilon)}} = \begin{cases} 1 & t = u \\ 0 & t \neq u \end{cases}$$

Proof. Let $f(t, u; B, \epsilon) = \frac{1}{1+e^{-B(t-u+\epsilon)}}$ and $g(t, u; B, \epsilon) = \frac{1}{1+e^{B(t-u-\epsilon)}}$. Then what we want to prove becomes

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} (f(t, u; B, \epsilon) \ \otimes \ g(t, u; B, \epsilon)) = \begin{cases} 1 & t = u \\ 0 & t \neq u \end{cases}$$

Because all $f, g, \otimes$ are continuous in their domain, we have

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} (f(t, u; B, \epsilon) \ \otimes \ g(t, u; B, \epsilon)) = \left(\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} f(t, u; B, \epsilon) \right) \otimes \left(\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} g(t, u; B, \epsilon) \right)$$

Using basic algebra, we get

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} f(t, u; B, \epsilon) = \begin{cases} 1 & t \geq u \\ 0 & t < u \end{cases} \qquad \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} g(t, u; B, \epsilon) = \begin{cases} 1 & t \leq u \\ 0 & t > u \end{cases}$$

Combing these results, we have

$$\left(\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} f(t, u; B, \epsilon) \right) \otimes \left(\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} g(t, u; B, \epsilon) \right) = \begin{cases} 0 \otimes 1 & t < u \\ 1 \otimes 1 & t = u \\ 1 \otimes 0 & t > u \end{cases}$$

For any t-norm, we have $0 \otimes 1 = 0$, $1 \otimes 1 = 1$, and $1 \otimes 0 = 0$. Put it altogether, we have

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to +\infty}} (f(t, u; B, \epsilon) \ \otimes \ g(t, u; B, \epsilon)) = \begin{cases} 1 & t = u \\ 0 & t \neq u \end{cases}$$

which concludes the proof. □

C PROOF OF THEOREM 1

Theorem 1. For any quantifier-free linear SMT formula F, there exists CLN model M, such that

$$\forall \boldsymbol{x}\ B\ \epsilon,\ 0 \leq M(\boldsymbol{x}; B, \epsilon) \leq 1 \tag{1}$$

$$\forall \boldsymbol{x},\ F(\boldsymbol{x}) = True \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1 \tag{2}$$

$$\forall \boldsymbol{x},\ F(\boldsymbol{x}) = False \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 0 \tag{3}$$

as long as the t-norm used in building M satisfies Property 1.

Proof. For convenience of the proof, we first remove all $<$, $\leq$, $=$ and $\neq$ in F, by transforming $t < u$ into $\neg(t \geq u)$, $t \leq u$ into $\neg(t > u)$, $t = u$ into $(t \geq u) \wedge \neg(t > u)$, and $t \neq u$ into $(t > u) \vee \neg(t \geq u)$. Now the only operators that F may contain are $>, \geq, \wedge, \vee, \neg$. We prove Theorem 1 by induction on the constructor of formula F. In the following proof, we construct model M given F and show it satisfied Eq.(1)(2). We leave the proof for why M also satisfied Eq.(3) to readers.

Atomic Case. When F is an atomic clause, then F will be in the form of $\boldsymbol{x} * W + b > 0$ or $\boldsymbol{x} * W + b \geq 0$. For the first case, we construct a linear layer with weight W and bias b followed by a sigmoid function scaled with factor B and right-shifted with distance ϵ. For the second case, we construct the same linear layer followed by a sigmoid function scaled with factor B and left-shifted with distance ϵ. Simply evaluating the limits for each we arrive at

$$\forall \boldsymbol{x}, F(\boldsymbol{x}) = True \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1$$

And from the definition of sigmoid function we know $0 \leq M(\boldsymbol{x}; B, \epsilon) \leq 1$.

Negation Case. If $F = \neg F'$, from the induction hypothesis, F' can be represented by models M' satisfying Eq.(1)(2)(3). Let p' output node of M'. We add a final output node $p = 1 - p'$. So $M(\boldsymbol{x}; B, \epsilon) = 1 - M'(\boldsymbol{x}; B, \epsilon)$. Using the induction hypothesis $0 \leq M'(\boldsymbol{x}; B, \epsilon) \leq 1$, we conclude Eq.(1) $0 \leq M(\boldsymbol{x}; B, \epsilon) \leq 1$.

Now we prove the " $\implies$ " side of Eq.(2). If $F(\boldsymbol{x}) = True$, then $F'(\boldsymbol{x}) = False$. From the induction hypothesis, we know $lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M'(\boldsymbol{x}; B, \epsilon) = 0$. So

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} 1 - M'(\boldsymbol{x}; B, \epsilon) = 1 - 0 = 1$$

Next we prove the " $\impliedby$ " side. If $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1$, we have

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M'(\boldsymbol{x}; B, \epsilon) = \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} 1 - M(\boldsymbol{x}; B, \epsilon) = 1 - 1 = 0$$

From the induction hypothesis we know that $F'(\boldsymbol{x}) = False$. So $F(\boldsymbol{x}) = \neg F'(\boldsymbol{x}) = True$.

Conjunction Case. If $F = F_1 \wedge F_2$, from the induction hypothesis, F_1 and F_2 can be represented by models M_1 and M_2, such that both (F_1, M_1) and (F_2, M_2) satisfy Eq.(1)(2)(3). Let p_1 and p_2 be the output nodes of M_1 and M_2. We add a final output node $p = p_1 \otimes p_2$. So $M(\boldsymbol{x}; B, \epsilon) = M_1(\boldsymbol{x}; B, \epsilon) \otimes M_2(\boldsymbol{x}; B, \epsilon)$. Since $(\otimes)$ is continuous and so are $M_1(\boldsymbol{x}; B, \epsilon)$ and $M_2(\boldsymbol{x}; B, \epsilon)$, we know their composition $M(\boldsymbol{x}; B, \epsilon)$ is also continuous. (Readers may wonder why $M1(\boldsymbol{x}; B, \epsilon)$ is continuous. Actually the continuity of $M(\boldsymbol{x}; B, \epsilon)$ should be proved inductively like this proof itself, and we omit it for brevity.) From the definition of $(\otimes)$, we have Eq.(1) $0 \leq M(\boldsymbol{x}; B, \epsilon) \leq 1$.

Now we prove the $\implies$ side of Eq.(2). For any $\boldsymbol{x}$, if $F(\boldsymbol{x}) = True$ which means both $F_1(\boldsymbol{x}) = True$ and $F_2(\boldsymbol{x}) = True$, from the induction hypothesis we know that $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon) = 1$ and $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_2(\boldsymbol{x}; B, \epsilon) = 1$. Then

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon) \otimes M_2(\boldsymbol{x}; B, \epsilon) = 1 \otimes 1 = 1$$

Then we prove the $\Longleftarrow$ side. From the induction hypothesis we know that $M_1(\boldsymbol{x}; B, \epsilon) \leq 1$ and $M_2(\boldsymbol{x}; B, \epsilon) \leq 1$. From the non-decreasing property of t-norms (see §2.3), we have

$$M_1(\boldsymbol{x}; B, \epsilon) \otimes M_2(\boldsymbol{x}; B, \epsilon) \quad \leq \quad M_1(\boldsymbol{x}; B, \epsilon) \otimes 1$$

Then from the consistency property and the commutative property, we have

$$M_1(\boldsymbol{x}; B, \epsilon) \otimes 1 = M_1(\boldsymbol{x}; B, \epsilon)$$

Put them altogether we get

$$M(\boldsymbol{x}; B, \epsilon) \quad \leq \quad M_1(\boldsymbol{x}; B, \epsilon) \quad \leq \quad 1$$

Because we know $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1$, according to the squeeze theorem in calculus, we get

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon) = 1$$

From the induction hypothesis, we know that $F_1(\boldsymbol{x}) = True$. We can prove $F_2(\boldsymbol{x}) = True$ in the same manner. Finally we have $F(\boldsymbol{x}) = F_1(\boldsymbol{x}) \wedge F_2(\boldsymbol{x}) = True$.

Disjunction Case. For the case $F = F_1 \vee F_2$, we construct M from M_1 and M_2 as we did in the conjunctive case. This time we let the final output node be $p = p_1 \oplus p_2$. From the continuity of $(\otimes)$ and the definition of $(\oplus)$ ($t \oplus u = 1 - (1 - t) \otimes (1 - u)$), $(\oplus)$ is also continuous. We conclude $M(\boldsymbol{x}; B, \epsilon)$ is also continuous and $0 \leq M(\boldsymbol{x}; B, \epsilon) \leq 1$ by the same argument as $F = F_1 \wedge F_2$.

Now we prove the " $\Longrightarrow$ " side of Eq.(2). For any assignment $\boldsymbol{x}$, if $F(\boldsymbol{x}) = True$ which means $F_1(\boldsymbol{x}) = True$ or $F_2(\boldsymbol{x}) = True$. Without loss of generality, we assume $F_1(\boldsymbol{x}) = True$. From the induction hypothesis, we know $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon) = 1$.

For any $(\oplus)$ and any $0 \leq t, t' \leq 1$, if $t \leq t'$, then

$$t \oplus u = 1 - (1 - t) \otimes (1 - u) \quad \leq \quad 1 - (1 - t') \otimes (1 - u) = t' \oplus u$$

Using this property and the induction hypothesis $M_2(\boldsymbol{x}; B, \epsilon) \geq 0$, we have

$$M_1(\boldsymbol{x}; B, \epsilon) \oplus 0 \quad \leq \quad M_1(\boldsymbol{x}; B, \epsilon) \oplus M_2(\boldsymbol{x}; B, \epsilon) = M(\boldsymbol{x}; B, \epsilon)$$

From the induction hypothesis we also have $M_1(\boldsymbol{x}; B, \epsilon) \leq 1$. Using the definition of $(\oplus)$ and the consistency of $(\otimes)$ ($0 \otimes x = 0$), we get $M_1(\boldsymbol{x}; B, \epsilon) \oplus 0 = M_1(\boldsymbol{x}; B, \epsilon)$. Put them altogether we get

$$M_1(\boldsymbol{x}; B, \epsilon) \quad \leq \quad M(\boldsymbol{x}; B, \epsilon) \leq 1$$

Because we know $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon) = 1$, according to the squeeze theorem in calculus, we get $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1$.

Then we prove the " $\Longleftarrow$ " side. Here we need to use the existence of limit:

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon)$$

This property can be proved by induction like this proof itself, thus omitted for brevity.

Let

$$c_1 = \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon)$$

and

$$c_2 = \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_2(\boldsymbol{x}; B, \epsilon)$$

Then

$$\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M_1(\boldsymbol{x}; B, \epsilon) \oplus M_2(\boldsymbol{x}; B, \epsilon) = c_1 \oplus c_2 = 1 - (1 - c_1) \otimes (1 - c_2)$$

Since we have $\lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} M(\boldsymbol{x}; B, \epsilon) = 1$, we get

$$(1 - c_1) \otimes (1 - c_2) = 0$$

Using Property 1 of ($\otimes$) (defined in §4), we have $c_1 = 1 \vee c_2 = 1$. Without loss of generality, we assume $c_1 = 1$. From the induction hypothesis, we know that $F_1(\boldsymbol{x}) = True$. Finally, $F(\boldsymbol{x}) = F_1(\boldsymbol{x}) \vee F_2(\boldsymbol{x}) = True$.

□

Careful readers may have found that if we use the continuous mapping function $\mathcal{S}$ in §3, then we have another perspective of the proof above, which can be viewed as two interwoven parts. The first part is that we proved the following lemma.

Corollary 1. *For any quantifier-free linear SMT formula F,*

$$\forall \boldsymbol{x}\ B\ \epsilon,\ 0 \leq \mathcal{S}(F; B, \epsilon)(\boldsymbol{x}) \leq 1$$

$$\forall \boldsymbol{x},\ F(\boldsymbol{x}) = True \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} \mathcal{S}(F; B, \epsilon)(\boldsymbol{x}) = 1$$

$$\forall \boldsymbol{x},\ F(\boldsymbol{x}) = False \iff \lim_{\substack{\epsilon \to 0^+ \\ B \cdot \epsilon \to \infty}} \mathcal{S}(F; B, \epsilon)(\boldsymbol{x}) = 0$$

Corollary 1 indicates the soundness of $\mathcal{S}$. The second part is that we construct a CLN model given $\mathcal{S}(F)$. In other words, we translate $\mathcal{S}(F)$ into vertices in a computational graph composed of differentiable operations on continuous truth values.

D Proof of Theorem 2

Theorem 2. For any CLN model M_F constructed from a formula, F, by the procedure shown in the proof of Theorem 1, if F is the conjunction of multiple linear equalities then any local minima of M_F is the global minima, as long as the t-norm used in building M_F satisfies Property 2.

Proof. Since F is the conjunction of linear equalities, it has the form

$$F = \bigwedge_{i=1}^{n} (\sum_{j=1}^{l_i} w_{ij} t_{ij} = 0)$$

Here $\mathbf{W} = \{w_{ij}\}$ are the learnable weights, and $\{t_{ij}\}$ are terms (variables). We omit the bias b_i in the linear equalities, as the bias can always be transformed into a weight by adding a constant of 1 as a term. For convenience, we define $f(x) = \mathcal{S}(x = 0) = \frac{1}{1+e^{-B(x+\epsilon)}} \otimes \frac{1}{1+e^{B(x-\epsilon)}}$.

Given an assignment $\boldsymbol{x}$ of the terms $\{t_{ij}\}$, if we construct our CLN model M_F following the procedure shown in the proof of Theorem 1, the output of the model will be

$$M(\boldsymbol{x}; \mathbf{W}, B, \epsilon) = \bigotimes_{i=1}^{n} f(\sum_{j=1}^{l_i} w_{ij} t_{ij})$$

When we train our CLN model, we have a collection of m data points $\{t_{ij1}\}, \{t_{ij2}\}, ..., \{t_{ijm}\}$, which satisfy formula F. If B and ϵ are fixed (unlearnable), then the loss function will be

$$L(\mathbf{W}) = \sum_{k=1}^{m} \mathcal{L}(M(\boldsymbol{x}; \mathbf{W}, B, \epsilon)) = \sum_{k=1}^{m} \mathcal{L}(\bigotimes_{i=1}^{n} f(\sum_{j=1}^{l_i} w_{ij} t_{ijk})) \tag{4}$$

Suppose $\mathbf{W}^* = \{w_{ij}^*\}$ is a local minima of $L(\mathbf{W})$. We need to prove $\mathbf{W}^*$ is also the global minima. To prove this, we use the definition of a local minima. That is,

$$\exists \delta > 0,\ \forall \mathbf{W},\ ||\mathbf{W} - \mathbf{W}^*|| \leq \delta \implies L(\mathbf{W}) \geq L(\mathbf{W}^*) \tag{5}$$

For convenience, we denote $u_{ik} = \sum_{j=1}^{l_i} w_{ij} t_{ijk}$. Then we rewrite Eq.(4) as

$$L(\mathbf{W}) = \sum_{k=1}^{m} \mathcal{L}(\bigotimes_{i=1}^{n} f(u_{ik}))$$

If we can prove at $\mathbf{W}^*$, $\forall\ 1 \leq i \leq n,\ 1 \leq k \leq m,\ u_{ik} = 0$. Then because (i) f reaches its global maximum at 0, (ii) the t-norm $(\otimes)$ is monotonically increasing, (iii) $\mathcal{L}$ is monotonically decreasing, we can conclude that $\mathbf{W}^*$ is the global minima.

Now we prove $\forall\ 1 \leq i \leq n,\ 1 \leq k \leq m,\ u_{ik} = 0$. Here we just show the case $i = 1$. The proof for $i > 1$ can be directly derived using the associativity of $(\otimes)$.

Let $\alpha_k = \bigotimes_{i=2}^{n} f(u_{ik})$. Since $f(x) > 0$ for all $x \in R$, using Property 2 of our t-norm $(\otimes)$, we know that $\alpha_k > 0$. Now the loss function becomes

$$L(\mathbf{W}) = \sum_{k=1}^{m} \mathcal{L}(f(u_{1k}) \otimes \alpha_k)$$

From Eq.(5), we have

$$\exists 0 < \delta' < 1,\ \forall \gamma,\ |\gamma| \leq \delta' \implies \sum_{k=1}^{m} \mathcal{L}(f(u_{ik}(1+\gamma)) \otimes \alpha_k) \geq \sum_{k=1}^{m} \mathcal{L}(f(u_{ik}) \otimes \alpha_k) \tag{6}$$

Because (i) $f(x)$ is an even function decreasing on $x > 0$ (which can be easily proved), (ii) $(\otimes)$ is monotonically increasing, (iii) $\mathcal{L}$ is monotonically decreasing, for $-\delta' < \gamma < 0$, we have

$$\sum_{k=1}^{m} \mathcal{L}(f(u_{ik}(1+\gamma)) \otimes \alpha_k) = \sum_{k=1}^{m} \mathcal{L}(f(|u_{ik}(1+\gamma)|) \otimes \alpha_k) \leq$$
$$\sum_{k=1}^{m} \mathcal{L}(f(|u_{ik}|) \otimes \alpha_k) = \sum_{k=1}^{m} \mathcal{L}(f(u_{ik}) \otimes \alpha_k) \tag{7}$$

Combing Eq.(6) and Eq.(7), we have

$$\sum_{k=1}^{m} \mathcal{L}(f(u_{ik}(1+\gamma)) \otimes \alpha_k) = \sum_{k=1}^{m} \mathcal{L}(f(u_{ik}) \otimes \alpha_k)$$

Now we look back on Eq.(7). Since (i) $\mathcal{L}$ is strictly decreasing, (ii) the t-norm we used here has Property 2 (see §4 for definition), (iii) $\alpha_k > 0$, the only case when $(=)$ holds is that for all $1 \leq k \leq m$, we have $f(|u_{ik}(1+\gamma)|) = f(|u_{ik}|)$. Since $f(x)$ is strictly decreasing for $x \geq 0$, we have $|u_{ik}(1+\gamma)| = |u_{ik}|$. Finally because $-1 < -\delta' < \gamma < 0$, we have $u_{ik} = 0$. □

E THEOREM 3 AND THE PROOF

Theorem 3. Given a program C: assume(P); while (LC) $\{C\}$ assert(Q);
If we can find a loop invariant I' for program C': assume($P \wedge LC$); while (LC) $\{C\}$ assert(Q);
and $P \wedge \neg LC \implies Q$, then $I' \vee (P \wedge \neg LC)$ is a correct loop invariant for program C.

Proof. Since I' is a loop invariant of C', we have

$$(P \wedge LC) \wedge LC \implies I' \quad (a) \qquad \{I' \wedge LC\}C\{I'\} \quad (b) \qquad I' \wedge \neg LC \implies Q \quad (c)$$

We want to prove $I' \vee (P \wedge \neg LC)$ is a valid loop invariant of C, which means

$$P \wedge LC \implies I' \vee (P \wedge \neg LC) \qquad \{(I' \vee (P \wedge \neg LC)) \wedge LC\}C\{I' \vee (P \wedge \neg LC)\}$$
$$(I' \vee (P \wedge \neg LC)) \wedge \neg LC \implies Q$$

We prove the three propositions separately. To prove $P \wedge LC \implies I' \vee (P \wedge \neg LC)$, we transform it into a stronger proposition $P \wedge LC \implies I'$, which directly comes from (a).

For $\{(I' \vee (P \wedge \neg LC)) \wedge LC\}C\{I' \vee (P \wedge \neg LC)\}$, after simplification it becomes $\{I' \wedge LC\}C\{I' \vee (P \wedge \neg LC)\}$, which is a direct corollary of (b).

For $(I' \vee (P \wedge \neg LC)) \wedge \neg LC \implies Q$, after simplification it will become two separate propositions, $I' \wedge \neg LC \implies Q$ and $P \wedge \neg LC \implies Q$. The former is exactly (c), and the latter is a known condition in the theorem. □

F PROPERTIES OF GAUSSIAN FUNCTION

We use a Gaussian-like function $S(t = u) = \exp(-\frac{(t-u)^2}{2\sigma_2})$ to represent equalities in our experiments. It has the following two properties. First, it preserves the original semantic of $=$ when $\sigma \to 0$, similar to the mapping $S(t = u) = \frac{1}{1+e^{-B(t-u+\epsilon)}} \otimes \frac{1}{1+e^{B(t-u-\epsilon)}}$ we defined in §3.

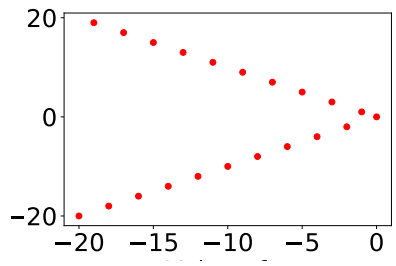
$$\lim_{\sigma \to 0^+} \exp(-\frac{(t-u)^2}{2\sigma^2}) = \begin{cases} 1 & t = u \\ 0 & t \neq u \end{cases}$$

Second, if we view $S(t = u)$ as a function over $t - u$, then it reaches its only local maximum at $t - u = 0$, which means the equality is satisfied.

G MORE DIFFICULT LOOP INVARIANT

```
//pre: t=-20/\u=-20
while (u != 0) {
  t++;
  if (u > 0)
    t = -u + 1;
  else
    t = -u - 1;
}
//post: t=0
```

(a) Example Pseudocode



(b) Plotted trace of program

Figure 6: Finding the Loop Invariant of Problem 1.

Description of Problems. In this section, we discuss in detail two of the more difficult loop invariant problems we have created. Consider the first example loop in Fig.6. For a loop invariant to be usable in verifying this loop, it must be valid for the precondition $t = -20, u = -20$, the recursion step representing loop execution when the loop condition is satisfied $(u \neq 0)$, and the post condition $t = 0$ when the loop condition is no longer satisfied $(u = 0)$. The correct and precise invariant for the program is $((t + u = 0) \vee (t - u = 0)) \wedge (t \leq 0)$. The plot of the trace in Fig.6b shows that the points lie on one of two lines expressible as linear equality constraints. These constraints along with the inequality can be learned from the execution trace in under 20 seconds. The invariant that is inferred is $((t + u = 0) \vee (t - u = 0)) \wedge (t \leq 0)$, which Z3 verifies is a sufficient loop invariant.

Figure 7 shows the pseudocode for the second program. The correct and precise loop invariant is $(t + u = 0) \wedge (v + w = 0) \wedge (u + w \geq 0)$. Our CLN can learn the this invariant with the t-norm in under 20 seconds. Both Code2inv and LoopInvGen time out within one hour without finding a solution.

```
//pre: t=-10 /\ u=10 /\ v=-10 /\ w=10
while (u + w > 0) {
    if (unknown()) {
        t++; u--;
    } else {
        v++; w--;
    }
}
//post: t=w /\ u=v
```

Figure 7: Pseudocode for Problem 2, which involves a conjunction of equalities

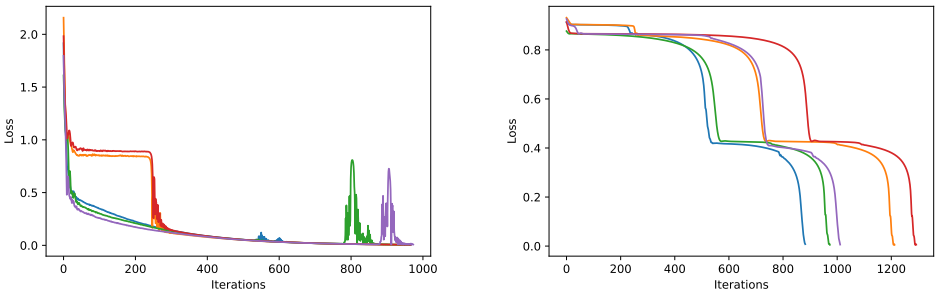
H T-NORMS AND T-CONORMS

Here we provide more details on the convergence of t-norms and T-conorms on our models. Table 2 summarizes the results of the comparison.

Convergence Traces. Figures 8, 9, and 10 show plots of the training loss for Godel, Lukasiewicz, and product t-norms and t-conorms respectively. Variations between individual traces are a result of random parameter initialization. CLNs with t-norms usually converge rapidly but sometimes will temporarily plateau when one clause converges before the other. In contrast, t-conorms always train one clause at a time, resulting in a curved staircase shape in the loss traces. We observe no significant difference between the t-norms used for conjunctions but observe that the product co-tnorms converges faster than the other t-conorms.

Table 2: Table with average iterations to convegence (average taken over 5 runs)

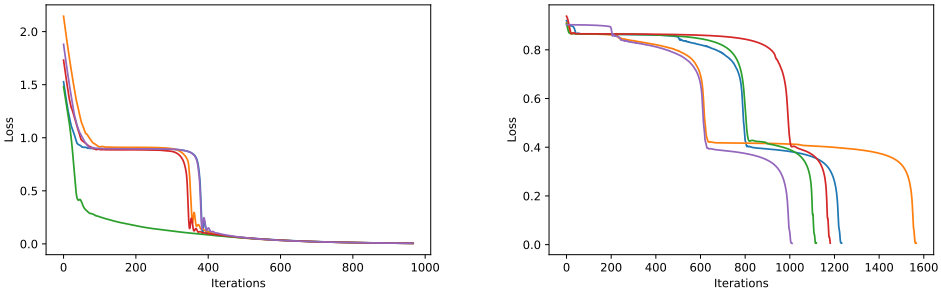
Problem	**Godel (iterations)**	**Lukasiewicz (iterations)**	**Product (iterations)**
Conjunction	967	966	966
Disjunction	1,074	1,221	984



(a) Godel t-norm

(b) Godel t-conorm

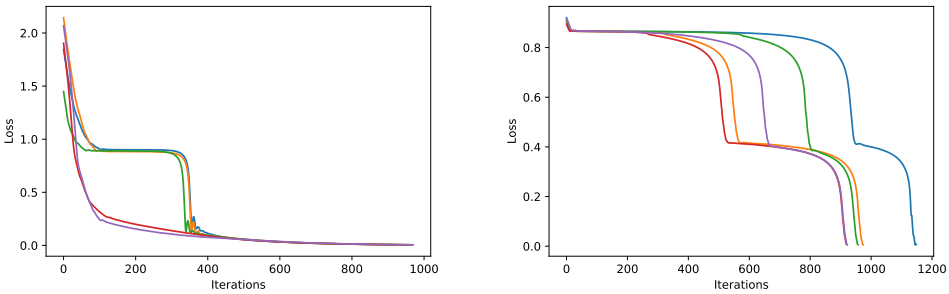
Figure 8: Godel t-norm and t-conorm



(a) Lukasiewicz t-norm

(b) Lukasiewicz t-conorm

Figure 9: Lukasiewicz t-norm and t-conorm



(a) Product t-norm

(b) Product t-conorm

Figure 10: Product t-norm and t-conorm