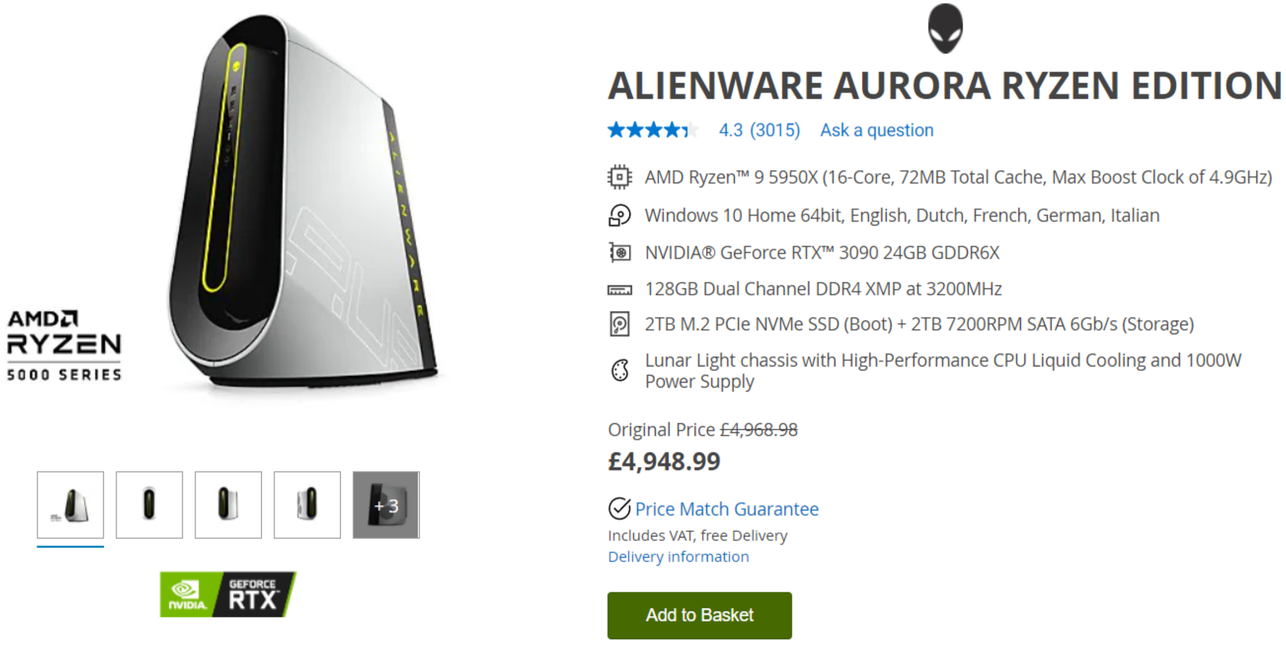
# Verifying Security in Embedded Software running in GPUs (WP4)

# Topics

- Admin (post-docs, equipment, etc)
- SMT-based Verification
- Incremental Verification of Fixed-Point Implementations of Neural Networks
- Efficient Formal Safety Analysis of Neural Networks

# Admin

- Edoardo Manino has officially started on February 1st
- Juliano Sales should officially start on April 1st, but we are still waiting for final approval from Robert and Alice about working remotely overseas
- Saumitra Mishra has not signed the contract yet
- Shall we advertise another PDRA position?
- We have the budget to buy a GPU server/host



# SMT-based Verification

# Satisfiability Modulo Theories

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories**

| **Theory** | **Example** |
|---|---|
| **Equality** | $\mathbf{x_1=x_2 \wedge \neg\ (x_1=x_3) \Rightarrow \neg(x_1=x_3)}$ |
| Bit-vectors | (b >> i) & 1 = 1 |
| Linear arithmetic | $(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)$ |
| Arrays | $(j = k \wedge a[k]=2) \Rightarrow a[j]=2$ |
| Combined theories | $(j \leq k \wedge a[j]=2) \Rightarrow a[i] < 3$ |

# Satisfiability Modulo Theories

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories**

| **Theory** | **Example** |
|---|---|
| Equality | $x_1=x_2 \wedge \neg (x_1=x_3) \Rightarrow \neg(x_1=x_3)$ |
| **Bit-vectors** | **(b >> i) & 1 = 1** |
| Linear arithmetic | $(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)$ |
| Arrays | $(j = k \wedge a[k]=2) \Rightarrow a[j]=2$ |
| Combined theories | $(j \leq k \wedge a[j]=2) \Rightarrow a[i] < 3$ |

$(a > 0) \wedge (b > 0) \Rightarrow (a + b > 0)$

# Satisfiability Modulo Theories

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories**

| **Theory** | **Example** |
|---|---|
| Equality | $x_1=x_2 \wedge \neg (x_1=x_3) \Rightarrow \neg(x_1=x_3)$ |
| Bit-vectors | (b >> i) & 1 = 1 |
| **Linear arithmetic** | $\mathbf{(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)}$ |
| Arrays | $(j = k \wedge a[k]=2) \Rightarrow a[j]=2$ |
| Combined theories | $(j \leq k \wedge a[j]=2) \Rightarrow a[i] < 3$ |

# Satisfiability Modulo Theories

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories**

| **Theory** | **Example** |
|---|---|
| Equality | $x_1=x_2 \wedge \neg (x_1=x_3) \Rightarrow \neg(x_1=x_3)$ |
| Bit-vectors | (b >> i) & 1 = 1 |
| Linear arithmetic | $(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)$ |
| **Arrays** | **$(j = k \wedge a[k]=2) \Rightarrow a[j]=2$** |
| Combined theories | $(j \leq k \wedge a[j]=2) \Rightarrow a[i] < 3$ |

$i = j \Rightarrow select(store\ (a, i, v), j) = v$

$i \neq j \Rightarrow select(store\ (a, i, v), j) = select(a, j)$

# Satisfiability Modulo Theories

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories**

| **Theory** | **Example** |
|---|---|
| Equality | $x_1=x_2 \wedge \neg (x_1=x_3) \Rightarrow \neg(x_1=x_3)$ |
| Bit-vectors | (b >> i) & 1 = 1 |
| Linear arithmetic | $(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)$ |
| Arrays | $(j = k \wedge a[k]=2) \Rightarrow a[j]=2$ |
| **Combined theories** | **$(j \leq k \wedge a[j]=2) \Rightarrow a[j] < 3$** |

# SMT-based Verification

- Given
  - a decidable $\sum$-theory T
  - a quantifier-free formula $\varphi$

  **$\varphi$ is T-satisfiable** iff $T \cup \{\varphi\}$ is satisfiable, i.e., there exists a structure that satisfies both formula and sentences of T
- Given
  - a set $\Gamma \cup \{\varphi\}$ of first-order formulae over T

  **$\varphi$ is a T-consequence of $\Gamma$** ($\Gamma \models_T \varphi$) iff every model of $T \cup \Gamma$ is also a model of $\varphi$
- Checking $\Gamma \models_T \varphi$ can be reduced in the usual way to checking the T-satisfiability of $\Gamma \cup \{\neg\varphi\}$

# Verification of Adversarial Cases

# Technical Report

**Computer Science > Logic in Computer Science**

*[Submitted on 21 Dec 2020]*

## Incremental Verification of Fixed-Point Implementations of Neural Networks

Luiz Sena, Erickson Alves, Iury Bessa, Eddie Filho, Lucas Cordeiro

Implementations of artificial neural networks (ANNs) might lead to failures, which are hardly predicted in the design phase since ANNs are highly parallel and their parameters are barely interpretable. Here, we develop and evaluate a novel symbolic verification framework using incremental bounded model checking (BMC), satisfiability modulo theories (SMT), and invariant inference, to obtain adversarial cases and validate coverage methods in a multi-layer perceptron (MLP). We exploit incremental BMC based on interval analysis to compute boundaries from a neuron's input. Then, the latter are propagated to effectively find a neuron's output since it is the input of the next one. This paper describes the first bit-precise symbolic verification framework to reason over actual implementations of ANNs in CUDA, based on invariant inference, therefore providing further guarantees about finite-precision arithmetic and its rounding errors, which are routinely ignored in the existing literature. We have implemented the proposed approach on top of the efficient SMT-based bounded model checker (ESBMC), and its experimental results show that it can successfully verify safety properties, in actual implementations of ANNs, and generate real adversarial cases in MLPs. Our approach was able to verify and produce adversarial examples for 85.8% of 21 test cases considering different input images, and 100% of the properties related to covering methods. Although our verification time is higher than existing approaches, our methodology can consider fixed-point implementation aspects that are disregarded by the state-of-the-art verification methodologies.

Comments: arXiv admin note: text overlap with arXiv:1907.12933
Subjects: **Logic in Computer Science (cs.LO)**; Artificial Intelligence (cs.AI)
Cite as: arXiv:2012.11220 **[cs.LO]**
(or arXiv:2012.11220v1 **[cs.LO]** for this version)

### Submission history

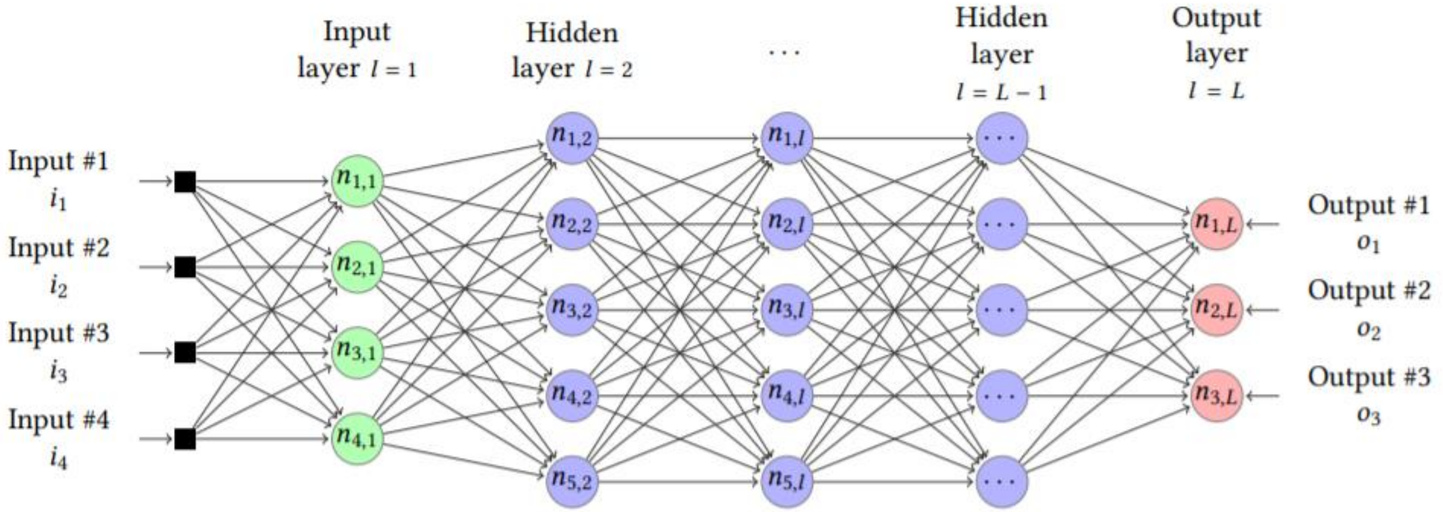
From: Lucas Carvalho Cordeiro [view email]
**[v1]** Mon, 21 Dec 2020 10:03:44 UTC (655 KB)

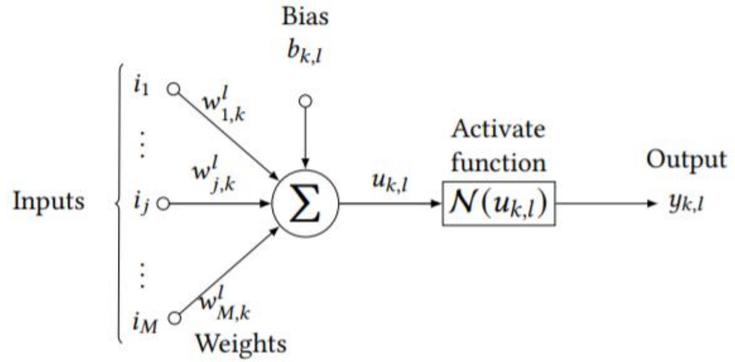
# SMT-based Verification for NB Architectures

- State-of-the-art verification framework
  - Bounded Model Checking
  - Incremental Bounded Model Checking
  - Proof by induction (*k*-induction)
  - Abstract interpretation (interval analysis)
- We verify two classes of bugs:
  1) generic implementation errors, e.g., invalid memory access or division-by-zero, which can cause the implementation of the DNN to crash
  2) failure of the implementation to behave according to the high-level rules, which may cause miss-classifications (adversarial cases)
- Validation of covering methods

# ANN Architecture

- Multilayer Perceptrons (MLP)
  - Consists of at least three layers: **input**, **hidden**, which typically employs a non-linear activation function, and **output** layers




  - Activation functions: **ReLU** and **Sigmoid**

$$ReLU : \mathbb{R} \rightarrow \mathbb{R}^+$$

$$ReLU(x) = max(0, x).$$

$$y_{k,l} = \mathcal{N}(u_{k,l}) = \frac{1}{1 + e^{-u_{k,l}}}.$$

# Efficient Formal Safety Analysis of Neural Networks

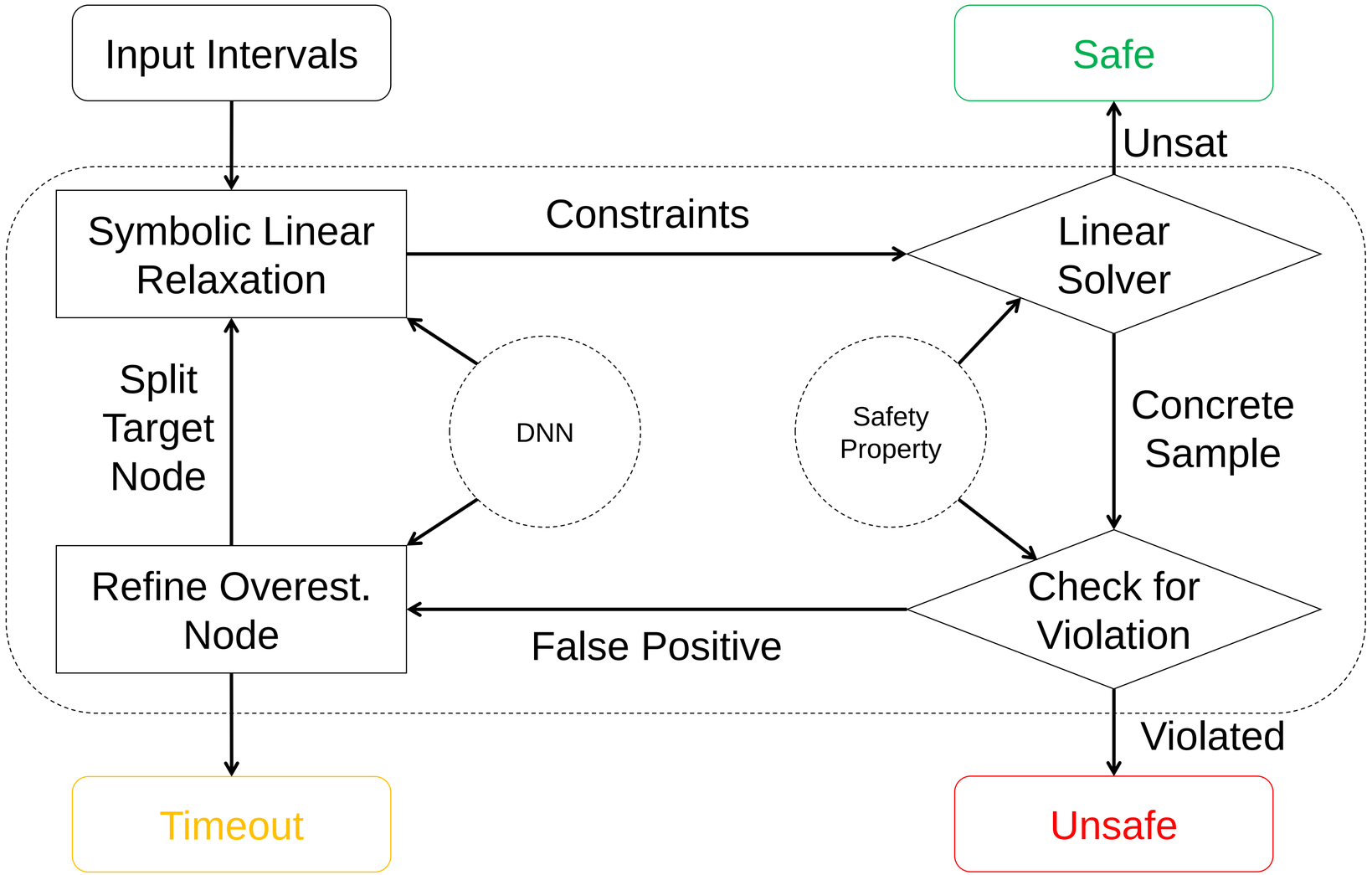
# Efficient Formal Safety Analysis of Neural Networks

**Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, Suman Jana**
Columbia University, NYC, NY 10027, USA
{tcwangshiqi, kpei, jaw2228, junfeng, suman}@cs.columbia.edu

## Abstract

Neural networks are increasingly deployed in real-world safety-critical domains such as autonomous driving, aircraft collision avoidance, and malware detection. However, these networks have been shown to often mispredict on inputs with minor adversarial or even accidental perturbations. Consequences of such errors can be disastrous and even potentially fatal as shown by the recent Tesla autopilot crashes. Thus, there is an urgent need for formal analysis systems that can rigorously check neural networks for violations of different safety properties such as robustness against adversarial perturbations within a certain $L$-norm of a given image. An effective safety analysis system for a neural network must be able to either ensure that a safety property is satisfied by the network or find a counterexample, i.e., an input for which the network will violate the property. Unfortunately, most existing techniques for performing such analysis struggle to scale beyond very small networks and the ones that can scale to larger networks suffer from high false positives and cannot produce concrete counterexamples in case of a property violation. In this paper, we present a new efficient approach for rigorously checking different safety properties of neural networks that significantly outperforms existing approaches by multiple orders of magnitude. Our approach can check different safety properties and find concrete counterexamples for networks that are $10\times$ larger than the ones supported by existing analysis techniques. We believe that our approach to estimating tight output bounds of a network for a given input range can also help improve the explainability of neural networks and guide the training process of more robust neural networks.

# Architecture



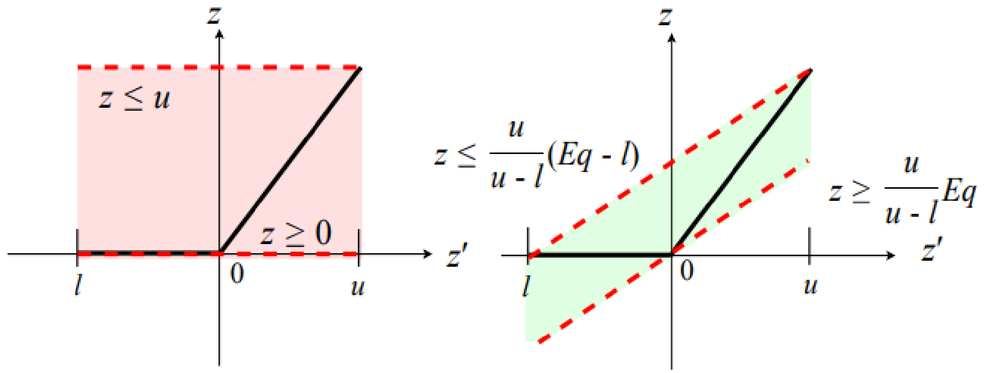

**Given**: Input Ranges, Target Network, and Predefined Safety Property

**Neurify**: Locate Overestimated Nodes with Symbolic Intervals and Iteratively Refine Approximated Output Ranges with Linear Solver

**Output**: Proved Safe, Unsafe with Counterexamples, or Timeout

# Symbolic Linear Relaxation

- Tighter interval analysis for ReLU propagations
- Partial input dependencies are preserved
- Used to identify crucial overestimated nodes, i.e., nodes that perform nonlinearity
- In practice, it can cut 59.64% more overestimation error than symbolic interval analysis




(a) Naive concretizaion

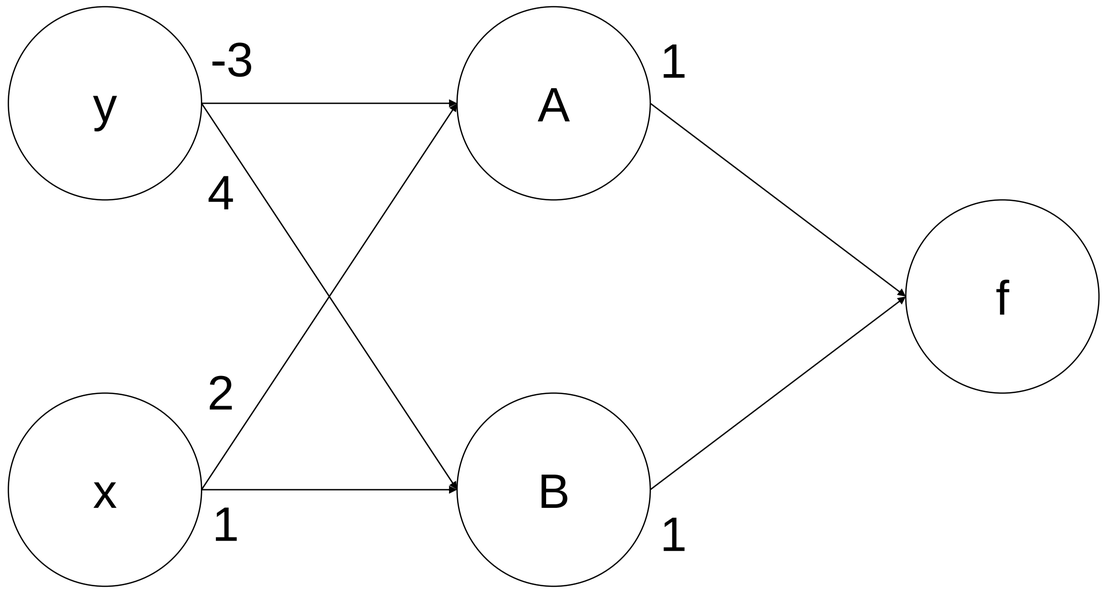
Dependencies ignored

(b) Symbolic linear relaxation

Partial dependencies

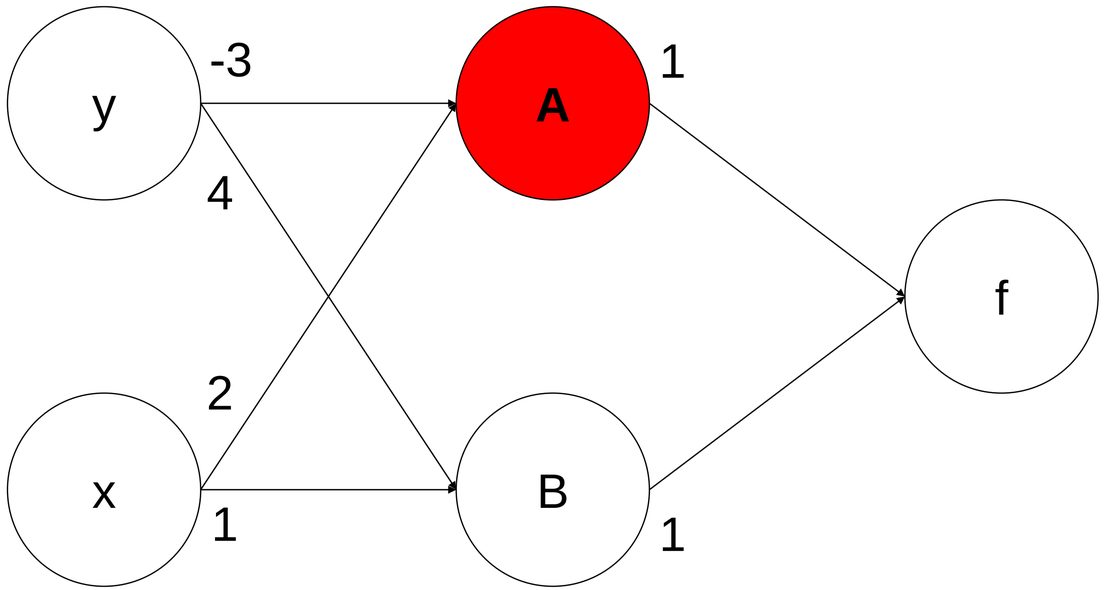
# Illustrative Example

| Neuron | Interval |
|---|---|
| x | [0, 1] |
| y | [0, 1] |
| A | ? |
| B | ? |



# Illustrative Example

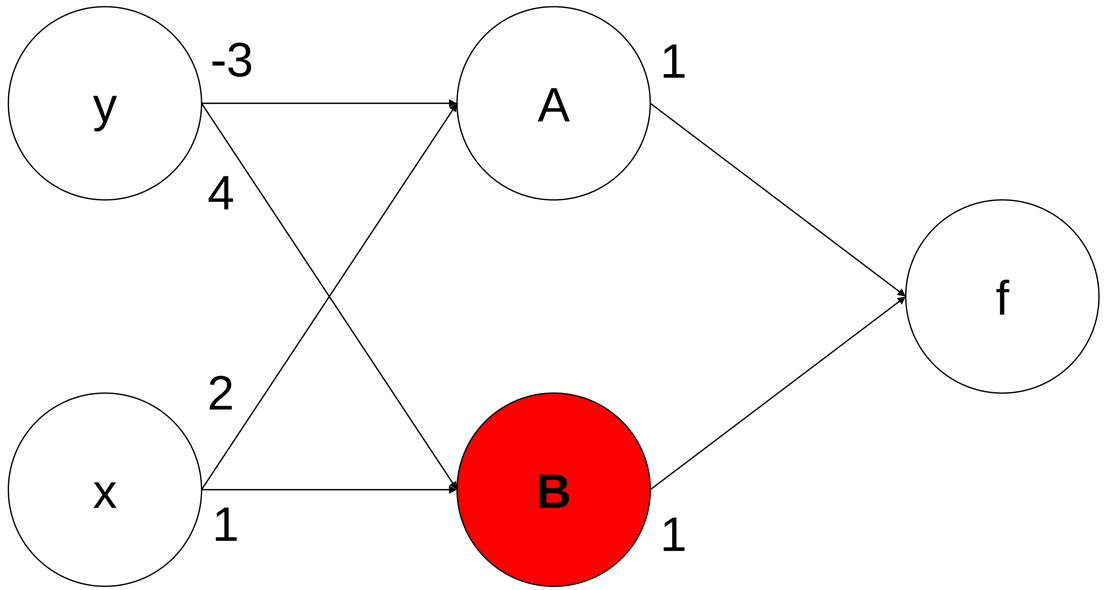
| Neuron | Interval |
|---|---|
| x | [0, 1] |
| y | [0, 1] |
| A | [-3, 2] |
| B | ? |




$$A = 2x - 3y$$
$$A_{lower} = 2 \times 0 - 3 \times 1 = -3$$
$$A_{upper} = 2 \times 1 - 3 \times 0 = 2$$

# Illustrative Example

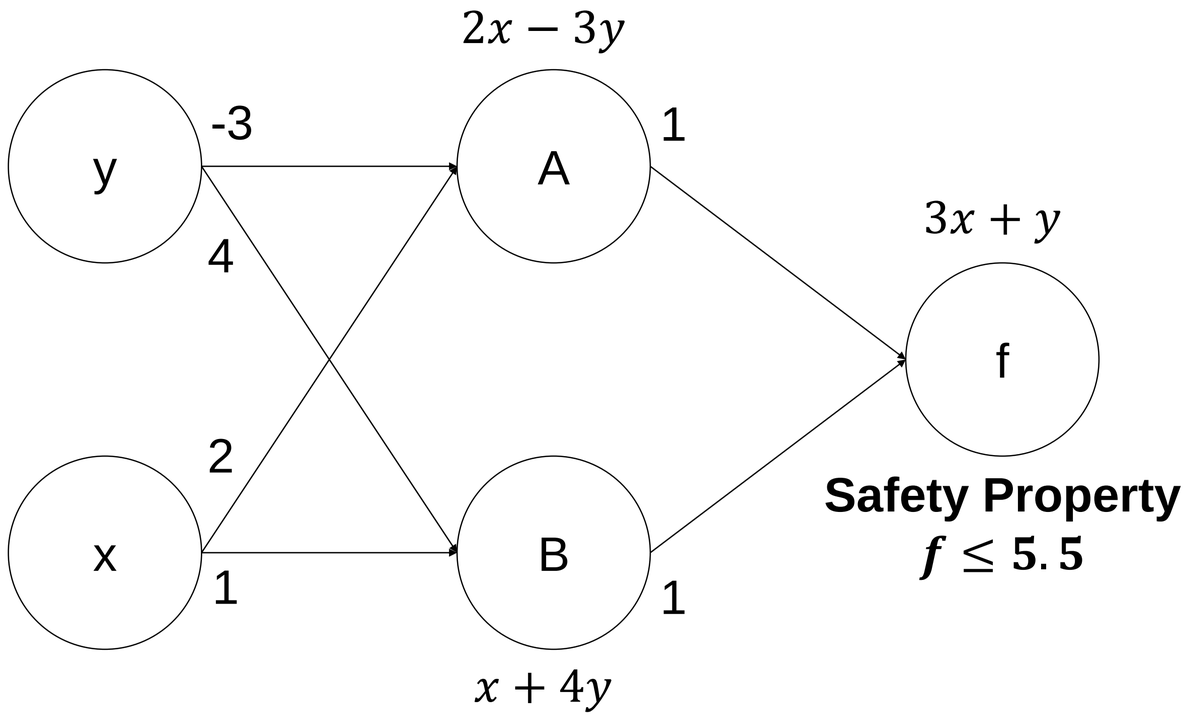
| Neuron | Interval |
|---|---|
| x | [0, 1] |
| y | [0, 1] |
| A | [-3, 2] |
| B | [0, 5] |




$$\boldsymbol{B = x + 4y}$$
$$\boldsymbol{B_{lower} = 0 + 4 \times 0 = 0}$$
$$\boldsymbol{B_{upper} = 1 + 4 \times 1 = 5}$$

# Illustrative Example

| Neuron | Interval |
|---|---|
| x | [0, 1] |
| y | [0, 1] |
| A | [-3, 2] |
| B | [0, 5] |




$$\mathbf{P = 3x + y - 5.5 \leq 0}$$

# Thank you