Neural Abstract Interpretation

Shaurya Gomber
University of Illinois Urbana-Champaign
Champaign, Illinois, USA

1 PROBLEM AND MOTIVATION

Abstract Interpretation [3] is a popular technique for formally analyzing the properties of programs [4, 5], neural networks [10, 21], and complex real-world systems [8] by soundly approximating their *concrete* semantics in an *abstract domain*. However, designing efficient abstract transformers for expressive relational domains such as Octagon [16] and Polyhedra [6] is hard as one needs to carefully balance the fundamental tradeoff between the cost, soundness, and the precision of the transformer for downstream task [16]. Further, scalable implementations involve intricate performance optimizations [22, 23]. We propose a data-driven learning approach to generate efficient abstract transformers to ease development costs. Given the inherent complexity of abstract transformers and the proven capability of neural networks to effectively approximate complex functions [7, 12], we envision and propose the Neural Abstract Interpretation (`NeurAbs`) framework (Section 3) to learn **neural abstract transformers**: neural networks that serve as the abstract transformers. These neural transformers can then act as a fast and sometimes even more precise replacement for slow and imprecise hand-crafted transformers. Additionally, these neural transformers are *differentiable* as opposed to the hand-crafted ones, enabling their use with gradient-guided learning methods, which can be beneficial for tasks that can be posed as learning problems (like invariant generation).

In Section 4, we illustrate how `NeurAbs` facilitates data-driven learning of various neural transformers for octagon join, achieving different levels of soundness and precision by fine-tuning the framework's inputs. We also highlight the benefits of the differentiability of these neural transformers by framing the search for octagon invariants in a loop program as a differentiable optimization task.

2 BACKGROUND AND RELATED WORKS

Abstract transformer corresponding to an operation op in the concrete domain (C) is a function $f_{op} : \mathcal{A} \rightarrow \mathcal{A}$ that captures the effect of applying op to concrete states corresponding to an abstract state in $\mathcal{A}$. f_{op} is *sound* if it over-approximates the output of op, i.e. $\forall a \in \mathcal{A}.\ op(\gamma(a)) \sqsubseteq_C \gamma(f_{op}(a))$ where $\gamma : \mathcal{A} \rightarrow C$ is the concretization function. The *precision* of f_{op} is essentially indicative of the degree of over-approximation due to f_{op} and can be quantified by some measure of the size of the abstract element computed by f_{op}. Precision is important as any abstract transformer that returns $\top$ is technically sound but is not very useful for practical settings. We ideally need sound transformers that are as precise as possible.

2.1 Related Works

Learning Abstract Transformers. Works like [13, 14] have been proposed to synthesize abstract transformers automatically. These expect the concrete domain and semantics as inputs and use symbolic methods to find exact abstract transformers. In contrast, our `NeurAbs` framework proposes a data-driven approach to learning neural transformers with varying soundness and precision. [2] uses a data-driven, counter-example guided learning method to learn static analyzers, but the learned static analyzers are symbolic and not differentiable. [11] uses a data-driven approach to learn a neural policy that allows it to remove redundant constraints from abstract states to achieve order of magnitude speedups, but it does not use neural networks to replace the hand-crafted transformers.

Neural Surrogates. In recent years, significant advancements have been made in developing and applying *neural surrogates for programs* [9, 19], focusing on their potential to speed up program execution [15, 17] and estimate program gradients [18, 20]. Our framework `NeurAbs` lifts this idea from concrete programs to abstract programs. The idea is that the learned neural abstract transformers for each program operation can be composed to obtain a *neural surrogate of the abstract program.*

3 APPROACH AND UNIQUENESS

The `NeurAbs` framework combines a supervised learning approach with a specialized loss function to fine-tune and balance the soundness-precision tradeoff while learning the neural abstract transformers. The general problem of learning abstract transformers that are simultaneously sound and precise is notably difficult. For instance, consider the task of learning the *sound* and *most-precise* octagon abstract transformer f_{op} from a set of functions $\mathcal{F}$ for an operator op. As defined in [3], we represent the "most-precise abstract transformer" for op as $f^{\#}_{op}$. [3] just provides a specification for $f^{\#}_{op}$, and in general, there is no way to compute it. Our f_{op} should be such that its output is sound for all possible inputs in the abstract domain, and the output is as close to $f^{\#}_{op}$. The optimization problem thus becomes:

$$\min_{f \in \mathcal{F}} \sum_{a \in \mathcal{A}} \mathcal{L}_P(f^{\#}_{op}(a),\ f(a)) \ \text{ s.t. } \sum_{a \in \mathcal{A}} \mathcal{L}_S(a,\ f(a)) = 0 \qquad (1)$$

where $\mathcal{L}_S(a, f(a))$ is 0 if $f(a)$ is a sound approximation of the effect op on a, i.e $op(\gamma(a)) \sqsubseteq_C \gamma(f(a))$ and 1 otherwise and $\mathcal{L}_P(o_1, o_2)$ can be any metric to measure how "big" is o_2 compared to o_1. Solving this optimization problem is complex due to the following reasons:

(1) The set $\mathcal{A}$ of all possible abstract elements has infinite size.
(2) As discussed above, there is no way to compute $f^{\#}_{op}$ in general, and thus computing $\mathcal{L}_P$ is not trivial.
(3) Computing $\mathcal{L}_S$ is usually done by finding counter-example e to soundness such that $op(\gamma(e)) \not\sqsubseteq_C \gamma(f(e))$. This is done by encoding this condition, along with the semantics of op and γ in SMT [1], and then using an SMT solver. This makes computing $\mathcal{L}_S$ expensive and makes the use of gradient-guided learning methods to solve the optimization problem infeasible due to the non-differentiability of such solvers.

To overcome the above challenges and make the learning problem feasible, we propose a **supervised approach for learning**

abstract transformers as a relaxation of the above problem. To the best of our knowledge, we are the first work to propose such relaxation, thus enabling the learning and use of neural abstract transformers. Given a dataset $\mathcal{D} = \{X_i, y_i\}$ representing input-output of an abstract transformer f in the octagon domain, we pose the learning of the neural abstract transformer f^* as the following optimization problem:

$$\min_{\theta} \mathbb{E}_{(X,y)\sim D}\ [\alpha * \mathcal{L}_S(y,\ f^*(x;\theta)) + \beta * \mathcal{L}_P(y,\ f^*(x;\theta))] \quad (2)$$

where if octagon o_1 is represented by the constraints $\pm v_i \pm v_j \leq c_{ij}$ and if the octagon o_2 is represented by the constraints $\pm v_i \pm v_j \leq c'_{ij}$, then:

(1) $\mathcal{L}_S(o_1, o_2) = \sum_{i,j} ite(c_{ij} - c'_{ij} > 0, c_{ij} - c'_{ij}, 0)$ is the sum of difference of inequality constants of o_1 and o_2 for all i, j where $c_{ij} > c'_{ij}$. This forces the c'_{ij}s to be greater than c_{ij}s and thus enforces soundness as this ensures that the output produced by the model is an over-approximation of the ground truth.
(2) $\mathcal{L}_P(o_1, o_2) = \sum_{i,j} |c_{ij} - c'_{ij}|$ is the sum of absolute differences of the corresponding inequality constants, which forces the output octagon constants to be as close as the ground truth and thus enforce precision.
(3) α, β are the soundness and precision weights, respectively, and let us control the degree of soundness and precision required for the transformer.

Soundness-Precision tradeoff. Note that solely having a soundness loss can guide the model to generate octagons with exceedingly high (approaching infinity) inequality constants. While this maintains soundness, it lacks precision. Conversely, exclusively incorporating precision loss may result in octagons with constants almost identical to the desired output but slightly less, thereby impacting soundness. By appropriately tuning the values of α and β, we can achieve neural transformers with varying degrees of soundness and precision. In settings such as verification, where soundness is very important, we can always resort to hand-crafted transformers' outputs if the output of the neural transformer is unsound (checking the soundness of an output is comparably simpler).

4 RESULTS AND CONTRIBUTIONS

4.1 Neural Octagon Transformers

We use the `NeurAbs` framework to train neural transformers for various octagon abstract transformers like join, meet, affine etc. We randomly generate octagons (with 3 variables) to compute the training and testing sets. For real-world use cases, these datasets can be sampled from the programs being analyzed. Table 1 shows the soundness and precision results for the trained transformer for octagon join. Soundness (%) denotes the percentage of sound outputs generated when the transformer is run on a test dataset of 1000 octagons. The imprecision measure is the average difference between the inequality constants of the model's output and the ground truth. As expected, increasing the soundness weight guides the model to learn more sound transformers, but this makes the model less precise. This shows the effectiveness of our framework in learning multiple neural transformers for the same abstract transformer with varying soundness and precision.

Soundness Weight (α)	Precision Weight (β)	Soundness Measure (%)	Imprecision Measure
1	1	9.6	49.96
10	1	36.6	85.30
20	1	49.0	110.51
50	1	64.5	129.26
100	1	79.0	184.58
250	1	86.3	291.23

Table 1: Soundness and Precision of Neural Octagon Join

4.2 Differentiable Learning of Loop Invariants

In this section, we highlight the advantages of our neural abstract transformers being differentiable by employing them in learning loop invariants. We frame this task of finding valid inductive octagonal invariants for a loop program $\mathcal{P}$ as a learning problem. Let's consider a typical loop program $\mathcal{P} = \texttt{while}(\beta)\ \texttt{do}\ C\ \texttt{od}$.
Let $f_C = f_n \circ f_{n-1} ... \circ f_1$ represent the effective abstract transformer for C where f_i represents the abstract transformer for $stmt_i$ in C. If O_{init} approximates the initial states ($init$) of $\mathcal{P}$, then the octagon O_{inv} is a valid octagonal invariant of the program if it satisfies:

$$(O_{init} \in O_{inv})\ \wedge\ (f_C(f_{conj}(O_{inv},\ \beta)) \in O_{inv}) \quad (3)$$

where f_{conj} represents the abstract transformer for taking the conjunction of an octagon with a condition like β and so $f_{conj}(O_{inv},\ \beta)$ is an approximation for points in O_{inv} that satisfy β.

Now, using the `NeurAbs` framework, we can derive the neural approximation f_i^* for each transformer f_i. These neural transformers can then be composed to get the *effective neural transformer for the loop body* (f_C^*) as $f_C^* = f_n^* \circ f_{n-1}^* ... \circ f_1^*$. Exploiting the differentiability of f_C^*, we can now pose the search for a valid octagonal invariant O_{inv} as the following optimization problem:

$$\min_{o \in O}\ \mathcal{L}_S(O_{init},\ o) + \mathcal{L}_S(f_C^*(f_{conj}^*(o, B)),\ o) \quad (4)$$

where O represents all possible octagons, f_{conj}^* represents the neural transformer for conjunction, and as described above, $\mathcal{L}_S(o_1, o_2)$ measures if o_2 is a sound approximation of o_1 ($o_1 \in o_2$). Starting with random octagons, we can now apply gradient descent guided by the loss described in Eq. 4 to find candidate octagon invariants, which can then be validated by verifying the correctness constraints described in Eq. 3 using an SMT solver.

As a concrete example, consider the following loop program:

```
x = 100; y = 150;
while (y <= 600) {
    x = x + y;
    y = 2*y;
}
```

Using a neural transformer trained for affine assignment in the octagon domain, the above method synthesizes non-trivial valid octagonal invariants like $\{y >= 65.514, x - y <= -49.951, -x - y <= 74.897\}$ and $\{x - y <= 13.239\}$ for this program. More precise invariants can be learned through better initialization strategies and by integrating the learned octagon's precision into the learning goal. However, this example highlights the effectiveness of our neural transformers in a practical setting of finding invariants.

REFERENCES

[1] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. 2009. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). Vol. 185. IOS Press, Chapter 26, 825–885. https://doi.org/10.3233/978-1-58603-929-5-825

[2] Pavol Bielik, Veselin Raychev, and Martin Vechev. 2017. Learning a Static Analyzer from Data. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 233–253.

[3] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Los Angeles, California) *(POPL '77)*. Association for Computing Machinery, New York, NY, USA, 238–252. https://doi.org/10.1145/512950.512973

[4] P. Cousot and R. Cousot. 2000. Abstract Interpretation Based Program Testing. In *Proceedings of the SSGRR 2000 Computer & eBusiness International Conference*. Scuola Superiore G. Reiss Romoli, Compact disk paper 248 and electronic proceedings http://www.ssgrr.it/en/ssgrr2000/proceedings.htm, L'Aquila, Italy.

[5] Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2005. The ASTREÉ Analyzer. In *Programming Languages and Systems*, Mooly Sagiv (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 21–30.

[6] Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Tucson, Arizona) *(POPL '78)*. Association for Computing Machinery, New York, NY, USA, 84–96. https://doi.org/10.1145/512760.512770

[7] George V. Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2 (1989), 303–314. https://api.semanticscholar.org/CorpusID:3958369

[8] Dennis Dams, Rob Gerth, and Orna Grumberg. 1997. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.* 19, 2 (mar 1997), 253–291. https://doi.org/10.1145/244795.244800

[9] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural Acceleration for General-Purpose Approximate Programs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 449–460. https://doi.org/10.1109/MICRO.2012.48

[10] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. 3–18. https://doi.org/10.1109/SP.2018.00058

[11] Jingxuan He, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2020. Learning fast and precise numerical analysis. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 1112–1127. https://doi.org/10.1145/3385412.3386016

[12] K. Hornik, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Netw.* 2, 5 (jul 1989), 359–366.

[13] Pankaj Kumar Kalita, Sujit Kumar Muduli, Loris D'Antoni, Thomas Reps, and Subhajit Roy. 2022. Synthesizing abstract transformers. 6, OOPSLA2, Article 171 (oct 2022), 29 pages. https://doi.org/10.1145/3563334

[14] Junghee Lim and Thomas Reps. 2013. TSL: A System for Generating Abstract Interpreters and its Application to Machine-Code Analysis. 35, 1, Article 4 (apr 2013), 59 pages. https://doi.org/10.1145/2450136.2450139

[15] Charith Mendis, Cambridge Yang, Yewen Pu, Dr.Saman Amarasinghe, and Michael Carbin. 2019. Compiler Auto-Vectorization with Imitation Learning. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/d1d5923fc822531bbfd9d87d4760914b-Paper.pdf

[16] A. Mine. 2001. The octagon abstract domain. In *Proceedings Eighth Working Conference on Reverse Engineering*. 310–319. https://doi.org/10.1109/WCRE.2001.957836

[17] Andreas Munk, Adam Ščibior, AtÄ±lÄ±m GÃ¼neÅŸ Baydin, Andrew Stewart, Goran Fernlund, Anoush Poursartip, and Frank Wood. 2020. Deep Probabilistic Surrogate Networks for Universal Simulator Approximation. In *International Conference on Probabilistic Programming (PROBPROG 2020), Cambridge, MA, United States*. https://probprog.cc/

[18] A. Renda, Y. Chen, C. Mendis, and M. Carbin. 2020. DiffTune: Optimizing CPU Simulator Parameters with Learned Differentiable Surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, Los Alamitos, CA, USA, 442–455. https://doi.org/10.1109/MICRO50266.2020.00045

[19] Alex Renda, Yi Ding, and Michael Carbin. 2021. Programming with neural surrogates of programs. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Chicago, IL, USA) *(Onward! 2021)*. Association for Computing Machinery, New York, NY, USA, 18–38. https://doi.org/10.1145/3486607.3486748

[20] Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, and Suman Jana. 2019. NEUZZ: Efficient Fuzzing with Neural Program Smoothing. 803–817. https://doi.org/10.1109/SP.2019.00052

[21] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3, POPL, Article 41 (jan 2019), 30 pages. https://doi.org/10.1145/3290354

[22] Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2015. Making numerical program analysis fast. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, David Grove and Stephen M. Blackburn (Eds.). ACM, 303–313.

[23] Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2017. Fast polyhedra abstract domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 46–59.