Neurosymbolic Programming

Neurosymbolic Programming

Swarat Chaudhuri
UT Austin
swarat@cs.utexas.edu

Kevin Ellis
Cornell
kellis@cornell.edu

Oleksandr Polozov
Google
polozov@google.com

Rishabh Singh
Google
rising@google.com

Armando Solar-Lezama
MIT
asolar@csail.mit.edu

Yisong Yue
Caltech
yyue@caltech.edu

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Programming Languages

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

S. Chaudhuri et al.. *Neurosymbolic Programming*. Foundations and Trends® in Programming Languages, vol. 7, no. 3, pp. 158–243, 2021.

ISBN: 978-1-68083-935-7
© 2021 S. Chaudhuri et al.

Foundations and Trends® in Programming Languages

Volume 7, Issue 3, 2021

Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract Interpretation
- Compilation and Interpretation Techniques
- Domain Specific Languages
- Formal Semantics, including Lambda Calculi, Process Calculi, and Process Algebra
- Language Paradigms
- Mechanical Proof Checking
- Memory Management
- Partial Evaluation
- Program Logic
- Programming Language Implementation
- Programming Language Security
- Programming Languages for Concurrency
- Programming Languages for Parallelism
- Program Synthesis
- Program Transformations and Optimizations
- Program Verification
- Runtime Techniques for Programming Languages
- Software Model Checking
- Static and Dynamic Program Analysis
- Type Theory and Type Systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2021, Volume 7, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.

Contents

1 Introduction **3**
1.1 What is Neurosymbolic Programming? 5
1.2 Benefits Over Deep Learning 7
1.3 Why Now? 8
1.4 Algorithmic Approaches 9
1.5 Applications 10
1.6 Roadmap 12

2 The Landscape of Neurosymbolic Programming **14**
2.1 Neurosymbolic Learning Algorithms 14
2.2 Neurosymbolic Representations 21

3 Motivating Goals **25**
3.1 Generalization and Sample Efficiency 25
3.2 Transfer and Abstraction 28
3.3 Interpretability 30
3.4 Safety 34
3.5 Procedural Reasoning 37

4 Learning Algorithms **41**
4.1 Symbolic Search 41
4.2 Gradient-Based Search 47

4.3 Learning to Synthesize 50
4.4 Distillation 55
4.5 Neural Relaxations 58
4.6 Transfer of Neural Modules 60
4.7 Transferring Symbolic Code 62
4.8 Programmatic Weak Supervision 67

5 Conclusion **69**

References **72**

Neurosymbolic Programming

Swarat Chaudhuri[1], Kevin Ellis[2], Oleksandr Polozov[3], Rishabh Singh[4], Armando Solar-Lezama[5] and Yisong Yue[6]

[1] *The University of Texas at Austin; swarat@cs.utexas.edu*
[2] *Cornell University; kellis@cornell.edu*
[3] *Google; Work authored while at Microsoft Research; polozov@google.com*
[4] *Google; rising@google.com*
[5] *Massachusetts Institute of Technology (MIT); asolar@csail.mit.edu*
[6] *The California Institute of Technology (Caltech); yyue@caltech.edu*

ABSTRACT

We survey recent work on *neurosymbolic programming*, an emerging area that bridges the areas of deep learning and program synthesis. Like in classic machine learning, the goal here is to learn functions from data. However, these functions are represented as *programs* that can use neural modules in addition to symbolic primitives and are induced using a combination of symbolic search and gradient-based optimization.

Neurosymbolic programming can offer multiple advantages over end-to-end deep learning. Programs can sometimes naturally represent long-horizon, procedural tasks that are difficult to perform using deep networks. Neurosymbolic representations are also, commonly, easier to interpret and formally verify than neural networks. The restrictions of a programming language can serve as a form of regularization and lead to more generalizable and data-efficient

Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama and Yisong Yue (2021), “Neurosymbolic Programming”, Foundations and Trends® in Programming Languages: Vol. 7, No. 3, pp 158–243. DOI: 10.1561/2500000049.

learning. Compositional programming abstractions can also be a natural way of reusing learned modules across learning tasks.

In this monograph, we illustrate these potential benefits with concrete examples from recent work on neurosymbolic programming. We also categorize the main ways in which symbolic and neural learning techniques come together in this area. We conclude with a discussion of the open technical challenges in the field.

1

Introduction

The last decade has seen breathtaking progress in the field of deep learning. Every few months, the media is abuzz with tales of a deep neural network conquering yet another milestone in artificial intelligence (AI). Deep learning systems such as AlphaGo (Silver *et al.*, 2016), the deep reinforcement learning agent that defeated a world champion in the game of Go, and GPT-3 (Brown *et al.*, 2020), the neural language model with 175 billion trainable parameters and the ability to generate stunningly human-like text, are now part of folklore.

At the same time, concerns remain about the use of deep neural networks in real-world problem domains (Marcus and Davis, 2019). In applications such as autonomous robotics and the natural sciences, it is important for learning models to be *interpretable*. However, neural networks are black boxes for most practical purposes. While modern neural networks are obtained through the composition of many layers, it is nearly impossible to assign responsibility for a network's capabilities to specific layers. This makes it difficult to *reuse* components of a network in the way that is possible for traditional, human-written software. Finally, the training process for neural networks is entirely data-driven and must learn even the most basic forms of human-held knowledge from data. Such training usually takes very large volumes of data. Also, commonly, their results depend on low-level facets of the training environment, and the networks they learn can fare poorly on inputs that fall even slightly outside the training distribution.

More fundamentally, as Bengio argues in his Turing lecture (Bengio, 2019), deep learning primarily automates what Kahneman (Kahneman, 2011) calls System 1 thinking: intuitive, fast, and unconscious pattern recognition. However, human cognition also includes System 2 thinking, which is slow, logical, and algorithmic. Bengio points out that AI systems of the future would need to also automate such thought.

Of course, the symbolic AI tradition, which dominated AI for several decades, had invested heavily on automating System 2 thought. This tradition modeled the world using symbolic rules and logical assertions and used symbolic methods like heuristic search and automated deduction to construct inferences and plans. Unfortunately, capturing the world entirely using rules and logic proved difficult, not least because it was difficult to model uncertain and ambiguous knowledge in such notations. Also, classical symbolic methods did not have a mechanism to handle sensory inputs. Finally, discrete reasoning is an NP-hard problem, and algorithms for tasks such as planning and theorem-proving could only scale so far. As a result, bottom-up, data-driven methods ended up eclipsing these methods.

However, as integrating System 1 and System 2 thought becomes more important, an emerging body of work on *neurosymbolic machine learning* is seeking to couple classical symbolic algorithms with contemporary deep learning techniques. The resulting whole is often greater than the sum of its parts. For example, the neural component of such a method can help the method's symbolic component scale better, by *guiding* the latter's discrete decisions. The neural component can also handle lower-level sensory processing and allow the symbolic algorithm to operate on perceptual inputs. Conversely, the symbolic component can often serve as a *regularizer* that helps the neural component learn better, and provide a level of interpretability and trust that would be impossible with a purely neural method.

In particular, there is an emerging body of work on neurosymbolic learning that lies at the interface of deep learning and *program* *synthesis* (Gulwani *et al.*, 2017). We refer to this literature as *Neurosymbolic Programming*. The purpose of this monograph is to survey recent developments in this area.

1.1 What is Neurosymbolic Programming?

Neurosymbolic programming, as we define the term, is a generalization of classic program synthesis. The goal in program synthesis is to automatically discover programs from high-level task specifications. Traditionally, the specifications are hard logical constraints, for example, tests that need to be satisfied exactly, pre-postcondition pairs, or temporal logic formulas. The programs are structured, symbolic terms that follow the syntax of a domain-specific language (DSL). The discovery of programs is based on a combination of combinatorial search and automated reasoning (Gulwani *et al.*, 2017).

By contrast, programs in neurosymbolic programming can have both neural and symbolic elements. The synthesis objective may include hard constraints like in classic program synthesis. However, neurosymbolic programming also incorporates into the picture the standard objective of machine learning: finding a function that fits a given dataset *approximately* and *generalizes* to unseen inputs.

Now we delineate the boundaries of neurosymbolic programming more precisely. Let us define a *symbolic component* as a function that comes with a symbolic implementation, or at least, a (possibly incomplete) symbolic specification of its functionality. In contrast, a *neural component* is a (typically over-)parameterized, differentiable blackbox function that does not have an a priori specification.

Composition is a fundamental operation in both traditional programming and deep learning. However, there is a key difference between composition in the two settings. In traditional programming, composition requires that certain requirements hold at the interface of the components being composed. No such requirement exists at the interface of different layers in deep learning. Let us designate the former kind of composition as *symbolic*. We consider a *neurosymbolic program* to be a program that uses neural components and either symbolic components or symbolic compositions.

A *neurosymbolic learning algorithm* is a mechanism for program synthesis that uses deep representations and gradient-based optimization as well as symbolic methods such as search and automated deduction. Such an algorithm must discover the program's discrete structure, or *architecture*. In addition, the algorithm must discover the program's real-valued parameters (for example, parameters of the program's neural modules), if any. The task specification that directs this search could include hard constraints like in classic program synthesis. Commonly, however, it also includes a quantitative loss function derived from, for example, labeled data or reward functions. The goal of the algorithm is to find a program that optimizes the loss while obeying the hard constraints.

The sets of methods that target neurosymbolic programs and use neurosymbolic learning algorithms are overlapping but not identical. In particular, there are methods that use neurosymbolic algorithms to discover symbolic programs, and methods that synthesize neurosymbolic programs using purely symbolic or purely neural methods. In this paper, we take a broad perspective and consider *neurosymbolic programming* to be the study of the *union* of the two sets.

We leave out of the scope of this paper models such as Neural Turing Machines (Graves *et al.*, 2014) and Neural Stack Machines (Grefenstette *et al.*, 2015). These models are inspired by classic models of programming, such as Turing Machines and stack machines, and have certain program-like capabilities (for example, Neural Turing Machines can perform reads and writes to a differentiable memory). However, these models are not learned using neurosymbolic algorithms. Also, they do not impose human-comprehensible specifications on the interfaces between model components and cannot be naturally decomposed into high-level modules.

Neither do we consider Tensor Product Representations (Smolensky *et al.*, 2016), which provide a symbolically inspired inductive bias in neural networks, to be an example of neurosymbolic programming. This is because such a model cannot be naturally interpreted as a program even when the network implements a programmatic task. Also outside our scope are Neural Programmer Interpreters (Reed and De Freitas, 2015), which are recurrent neural networks that learn to execute programs, and neural models for combinatorial tasks such as MAX-SAT solving (Wang *et al.*, 2019). That said, all of these models are closely related to neurosymbolic programming, and future work may integrate them with approaches that we discuss in this monograph.

1.2 Benefits Over Deep Learning

Neurosymbolic programming has multiple potential benefits over end-to-end deep learning. In general, by virtue of their modularity and use of symbolic primitives, neurosymbolic programs are closer to human-written code than deep networks. Because of this, neurosymbolic programming can provide a means for interpretable learning, especially when the learning algorithm being used is biased towards models that are "more symbolic" and structurally simpler.

Symbolic abstractions can also simplify the automated *analysis* of models. Over the last few years, there have been many efforts on algorithmic analysis of the safety and robustness of neural networks (Katz *et al.*, 2017; Gehr *et al.*, 2018; Anderson *et al.*, 2019). To a significant extent, these methods are adaptations of methods for quality assurance of traditional software. Unfortunately, analyzing even simple properties of deep neural networks is NP-hard, and scaling these analysis methods to real-world neural networks is difficult. In contrast, recent work on certifiable learning reduces (Anderson *et al.*, 2020) the safety and robustness analysis of certain specialized neurosymbolic models to the analysis of its (simpler, more compact) symbolic components. The latter task can be done relatively easily with existing software analysis techniques.

Neurosymbolic programming gives human users a mechanism to guide the learning process, and this can lead to more reliable learning. For example, in recent work in the reinforcement learning setting (Cheng *et al.*, 2019), a (deep) learning algorithm is given a human-provided function (program) that performs a task, albeit not optimally, and tasked with improving the performance of this program. The resulting learning process has lower statistical variance than one in which the task is entirely learned from data. If the prior is of reasonable quality, this strategy can lead to greater overall performance as well.

Also, a key advantage of high-level programming abstractions is that they tend to be compositional, i.e., allow the structured creation of larger programs using smaller modules. This enables a principled mechanism of *transferring* knowledge (Valkov *et al.*, 2018) across learning tasks: one can train a model in one task and reuse it in another task in the way a human programmer would use a library module. This mechanism is a generalization of a popular family of transfer learning techniques in deep learning, which commonly reuse network layers with frozen weights across tasks (Yosinski *et al.*, 2014).

Finally, higher-level programming abstractions can reduce the supervision effort needed for learning. In supervised learning settings such as image classification, one requires human users to label training inputs, and this can get expensive. In contrast, in the *data programming* paradigm (Ratner *et al.*, 2016), the user writes *labeling programs* that can automatically produce labels for inputs. Such programs are easy to write or automatically synthesize in many domains (Zhan *et al.*, 2020; Sun *et al.*, 2020). When they are available, they can drastically reduce the cost of learning.

1.3 Why Now?

The idea of combining neural and symbolic methods has a long history in AI research (Sun and Alexandre, 2013; Garcez *et al.*, 2002). In *knowledge-based neural networks*, an early example of a neurosymbolic model (Towell *et al.*, 1990), a set of hand-written symbolic rules were compiled into a neural network, which is then refined using data. It was shown that such neural networks are more data-efficient and tend to generalize better than classic neural networks. There were complementary efforts that extracted symbolic models, such as rules (Towell and Shavlik, 1993) and finite automata (Giles *et al.*, 1992), out of neural networks, essentially performing a form of program synthesis. Unifying these two strands of work, Shavlik, 1994 proposed a general learning framework in which an initial neural network, constructed using symbolic knowledge, is refined using data, and new symbolic knowledge is extracted from this refined network. These ideas were embodied, and taken further, in the Connectionist Inductive Learning and Logic

Programming system (Garcez and Zaverucha, 1999), which integrated logic programming and neural networks. These approaches certainly fit the category of neurosymbolic programming as defined in this paper.

While the first wave of neurosymbolic programming produced many interesting ideas, the practical impact of this line of work was limited. However, the current moment feels especially appropriate for resurrecting this area. For one, AI is increasingly deployed in real-world problems in which safety, reliability, and interpretability are important, and there is growing awareness about the limitations of pure deep learning in these problems. This opens up a window of opportunity for neurosymbolic methods. (Indeed, as we describe in Section 1.5, neurosymbolic methods are already making inroads into these tasks.)

Second, we now have access to much larger datasets and computational power than we did in the 1990s. This fact was key to the revival of neural networks as a research area, and it can help neurosymbolic programming as well. Finally, in the recent past, there has been significant progress on symbolic program synthesis, and new ways of coupling gradient-based and combinatorial search have emerged. As we show in this monograph, a new wave of research on neurosymbolic programming is already beginning to build on this progress.

1.4 Algorithmic Approaches

The fundamental challenge in neurosymbolic programming is that here, one must search through a combinatorial, and quickly exploding, space of program architectures. Worse, for each architecture explored in such a search, one must often perform high-dimensional continuous optimization to find optimal parameters for the neural modules appearing in the architecture. Nevertheless, over the last few years, researchers have discovered multiple new lines of attack on this problem, and some unifying themes are beginning to emerge in this area.

For example, one set of methods for neurosymbolic programming uses a neural network to *learn to synthesize programs*, i.e., direct a search process over program architectures (Balog *et al.*, 2016; Murali *et al.*, 2018). The network is trained using metalearning, from data that relates a set of tasks to programs that solve the tasks. Once a program architecture is generated, its lower-level parameters can be found using neural or symbolic methods.

A second category of methods, which we call *learning to specify* (Ellis *et al.*, 2018b), determine how to generalize incomplete or ambiguous task specifications to more complete specifications. These complete specifications are then used to direct a program synthesis process. A third category uses *neural relaxations* of a nonsmooth set of programs (Shah *et al.*, 2020). This space could consist of programs with completely different architectures. However, since a program is ultimately a representation of a function, the parameters of a neural network can (approximately) represent it. A final body of methods goes in the other direction, *distilling* a smooth neural function (Verma *et al.*, 2018) into a discrete program whose behavior approximately matches the network's. We discuss all of these methods in more depth in Section 2 and Section 4.

1.5 Applications

The algorithmic innovations sketched above are already beginning to impact real-world applications. Now we sketch some of these applications. Given the increased deployment of machine learning in domains in which trust and procedural reasoning are important, we expect many more such applications to emerge in the coming years.

Scientific Discovery. Building learning algorithms that discover new scientific hypotheses and guide experiments is a grand challenge in AI. Such algorithms must respect constraints known to hold in the world and produce outputs that scientists can interpret. This makes neurosymbolic programming a natural fit to this space.

As a concrete example, Cranmer *et al.* (2020) propose a method for *symbolic regression* — the automatic discovery of symbolic equations from data — and apply it to a task in cosmology (dark matter prediction). Also, several recent efforts use neurosymbolic programming in behavior analysis of lab animals. For instance, Sun *et al.* (2020) use a neurosymbolic method to embed videos of lab animals into lower-dimensional representations. Shah *et al.* (2020) use neurosymbolic programming to classify sequential animal behaviors. Zhan *et al.* (2021)

use neurosymbolic representation learning for interpretable clustering of such behaviors. Tjandrasuwita *et al.* (2021) learn interpretable programs that describe divergences between different human experts annotating behaviors.

Programming Systems. There has been significant recent interest in machine-learning-based assistants for software developers. Purely neural tools often struggle with understanding the complex, logical semantics of software, which are sensitive to even the smallest changes in syntax. Neurosymbolic programming is a natural way to overcome this issue given that symbolic methods have long been used successfully in program analysis.

For example, the BAYOU system (Murali *et al.*, 2018) automatically completes Java methods given a few keywords that appear in the method. The PATOIS system (Shin *et al.*, 2019a) uses neurosymbolic programming to semantically parse text into code. Ellis *et al.* (2018b) simplifies graphics programming with a method to synthesize graphics code from a given picture.

Dialog Systems. Task-oriented dialog systems assist users with specific goals through a natural language interface. As digital assistants, they facilitate travel booking, database question answering, scheduling, and much more. The key challenge of task-oriented dialog is *state tracking* — identifying the user's intent and parameters in each dialog act, and using them to drive the system's actions. Fundamentally, dialog state is an intermediate symbolic representation that depends on complex, high-dimensional semantic context, namely dialog history and the underlying knowledge base or API. Thereby, neurosymbolic programming is a natural choice for modeling dialog state, successfully applied in many domains. For example, Andreas *et al.* (2020) design a calendar assistant in which scheduling actions, dialog corrections, and exceptions are represented as compositional programs, synthesized by neurosymbolic models in context.

Process Automation. The field of *robotic process automation (RPA)* aims to automate procedural GUI workflows to facilitate business digitization and software testing. RPA agents interact with Web browsers, GUI applications, and APIs to accomplish the user's parameterized tasks. They are typically pretrained for each task using natural language commands, UI-grounded demonstrations, task completion rewards, or some combination thereof.

In RPA, the agent's state and action spaces are enormous – the current screen or Web page defines the state and the action space includes all possible interactions with its elements. Learning a robust and interpretable RPA agent is challenging even from grounded demonstrations as supervision. Instead, recent approaches leverage neurosymbolic programming and model the agent as a neurosymbolic task program. For example, Srivastava *et al.* (2020) combine neural language modeling with inductive program synthesis Gulwani *et al.*, 2017 to learn a generative model of programs that both guarantees consistency with the demonstrations and optimizes natural language alignment.

Robotics and Control. When designing policies or controllers for autonomous embodied systems, factors such as safety and data efficiency become paramount. For both low-level control and high-level planning problems, the standard practice has been to leverage symbolic domain knowledge (e.g., the governing equations of motion for the system, or an automaton representation of the high-level states) to design structured models that have certifiable guarantees, good generalization, or both (*e.g.*, Verma *et al.* (2019)). An emerging research direction is to automatically learn or discover the structure of the symbolic knowledge (*e.g.*, Xu *et al.* (2018)), which can be viewed as an instance of neurosymbolic programming.

1.6 Roadmap

The rest of this monograph is organized as follows. In Section 2, we give an overview of the landscape of recent research on neurosymbolic programming. Section 3 describes in some depth the main motivating goals for research in this area, along with concrete examples of how recent research is addressing these goals. In Section 4, we discuss some of the common themes in learning algorithms for neurosymbolic programs.

We conclude with a discussion of future challenges in the area in Section 5.

References

Achiam, J., D. Held, A. Tamar, and P. Abbeel. (2017). “Constrained Policy Optimization”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. *Proceedings of Machine Learning Research*. PMLR. 22–31.

Alet, F., T. Lozano-Perez, and L. P. Kaelbling. (2018a). “Modular meta-learning”. In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by A. Billard, A. Dragan, J. Peters, and J. Morimoto. Vol. 87. *Proceedings of Machine Learning Research*. PMLR. 856–868. URL: http://proceedings.mlr.press/v87/alet18a.html.

Alet, F., T. Lozano-Pérez, and L. P. Kaelbling. (2018b). “Modular meta-learning”. *Conference on Robot Learning*.

Alshiekh, M., R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. (2018). “Safe Reinforcement Learning via Shielding”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*. Ed. by S. A. McIlraith and K. Q. Weinberger. AAAI Press. 2669–2678.

Alur, R., R. Bodík, E. Dallal, D. Fisman, P. Garg, G. Juniwal, H. Kress-Gazit, P. Madhusudan, M. M. K. Martin, M. Raghothaman, S. Saha, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. (2015). “Syntax-Guided Synthesis”. In: *Dependable Software Systems Engineering*. 1–25. DOI: 10.3233/978-1-61499-495-4-1.

Amini, A., S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi. (2019). “MathQA: Towards interpretable math word problem solving with operation-based formalisms”. *arXiv preprint arXiv:1905.13319*.

Amizadeh, S., H. Palangi, O. Polozov, Y. Huang, and K. Koishida. (2020). “Neuro-Symbolic Visual Reasoning: Disentangling “Visual” from “Reasoning””. In: *International Conference on Machine Learning*.

Anderson, G., S. Pailoor, I. Dillig, and S. Chaudhuri. (2019). “Optimization and abstraction: a synergistic approach for analyzing neural network robustness”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM. 731–744.

Anderson, G., A. Verma, I. Dillig, and S. Chaudhuri. (2020). “Neurosymbolic Reinforcement Learning with Formally Verified Exploration”. In: *Neural Information Processesing Systems (NeurIPS)*.

Andreas, J., J. Bufe, D. Burkett, C. Chen, J. Clausman, J. Crawford, K. Crim, J. DeLoach, L. Dorner, J. Eisner, and et al. (2020). “Task-Oriented Dialogue as Dataflow Synthesis”. *Transactions of the Association for Computational Linguistics*. 8(Dec.): 556–571.

Andreas, J., M. Rohrbach, T. Darrell, and D. Klein. (2016a). “Learning to Compose Neural Networks for Question Answering”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California. 1545–1554.

Andreas, J., M. Rohrbach, T. Darrell, and D. Klein. (2016b). “Neural module networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 39–48.

Balog, M., A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. (2016). “DeepCoder: Learning to Write Programs”. *CoRR*. abs/1611.01989. arXiv: 1611.01989.

Barnat, J., L. Brim, M. Češka, and P. Ročkai. (2010). “Divine: Parallel distributed model checker”. In: *Ninth international workshop on parallel and distributed methods in verification, and second international workshop on high performance computational systems biology*. IEEE. 4–7.

Barrett, C., P. Fontaine, and C. Tinelli. (2010). *The SMT-LIB Standard Version 2.6.*

Bastani, O., Y. Pu, and A. Solar-Lezama. (2018). "Verifiable Reinforcement Learning via Policy Extraction". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada.* Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. 2499–2509.

Beery, S., G. Van Horn, and P. Perona. (2018). "Recognition in Terra Incognita". In: *Proceedings of the European Conference on Computer Vision (ECCV).* 456–473.

Bengio, Y. (2019). "From System 1 Deep Learning to System 2 Deep Learning". Neural Information Processing Systems.

Bickel, S., M. Brückner, and T. Scheffer. (2009). "Discriminative learning under covariate shift." *Journal of Machine Learning Research.* 10(9).

Bingham, E., J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. (2019). "Pyro: Deep Universal Probabilistic Programming". *J. Mach. Learn. Res.* 20(1): 973–978. ISSN: 1532-4435.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. (2020). "Language Models are Few-Shot Learners". *CoRR.* abs/2005.14165. arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

Bunel, R., M. J. Hausknecht, J. Devlin, R. Singh, and P. Kohli. (2018). "Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net.

Carpenter, B., A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. (2017). "Stan: A probabilistic programming language". *Journal of statistical software*. 76(1).

Chaudhuri, S., M. Clochard, and A. Solar-Lezama. (2014). "Bridging boolean and quantitative synthesis using smoothed proof search". In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 207–220.

Chaudhuri, S. and A. Solar-Lezama. (2010). "Smooth interpretation". In: *PLDI*. 279–291.

Chen, T., S. Kornblith, M. Norouzi, and G. Hinton. (2020a). "A simple framework for contrastive learning of visual representations". In: *International Conference on Machine Learning*.

Chen, X., M. Monfort, A. Liu, and B. D. Ziebart. (2016). "Robust covariate shift regression". In: *Artificial Intelligence and Statistics*. 1270–1279.

Chen, X., C. Liang, A. W. Yu, D. Zhou, D. Song, and Q. V. Le. (2019). "Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension". In: *International Conference on Learning Representations*.

Chen, Y., C. Wang, O. Bastani, I. Dillig, and Y. Feng. (2020b). "Program Synthesis Using Deduction-Guided Reinforcement Learning". In: *International Conference on Computer Aided Verification*. Springer. 587–610.

Cheng, R., A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. W. Burdick. (2019). "Control regularization for reduced variance reinforcement learning". In: *ICML*.

Cheung, A., A. Solar-Lezama, and S. Madden. (2012). "Using program synthesis for social recommendations". In: *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*. Ed. by X. Chen, G. Lebanon, H. Wang, and M. J. Zaki. ACM. 1732–1736.

Chow, Y., O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. (2018). "A Lyapunov-based approach to safe reinforcement learning". In: *Advances in neural information processing systems*. 8092–8101.

Cranmer, M., A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho. (2020). "Discovering symbolic models from deep learning with inductive biases". In: *Neural Information Processing Systems*.

Cui, G. and H. Zhu. (2021). "Differentiable Synthesis of Program Architectures". *Advances in Neural Information Processing Systems*. 34.

Dal Palù, A., A. Dovier, A. Formisano, and E. Pontelli. (2015). "Cud@ sat: Sat solving on gpus". *Journal of Experimental & Theoretical Artificial Intelligence*. 27(3): 293–316.

Daumé III, H. (2008). "Cross-Task Knowledge-Constrained Self Training". In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. 680–688.

De Raedt, L., A. Kimmig, and H. Toivonen. (2007). "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery." In: *IJCAI*. Vol. 7. Hyderabad. 2462–2467.

Dechter, E., J. Malmaud, R. P. Adams, and J. B. Tenenbaum. (2013). "Bootstrap learning via modular concept discovery". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI Press/International Joint Conferences on Artificial Intelligence.

Dennis, L., M. Fisher, M. Slavkovik, and M. Webster. (2016). "Formal verification of ethical choices in autonomous systems". *Robotics and Autonomous Systems*. 77: 1–14.

Devlin, J., J. Uesato, S. Bhupatiraju, R. Singh, A. Mohamed, and P. Kohli. (2017). "RobustFill: Neural Program Learning under Noisy I/O". In: *ICML*.

Dua, D., Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. (2019). "DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs". *arXiv preprint arXiv:1903.00161*.

Dudík, M., D. Erhan, J. Langford, L. Li, *et al.* (2014). "Doubly robust policy evaluation and optimization". *Statistical Science*. 29(4): 485–511.

Ellis, K., L. Morales, M. Sablé-Meyer, A. Solar-Lezama, and J. Tenenbaum. (2018a). "Learning libraries of subroutines for neurally–guided Bayesian program induction". In: *Advances in Neural Information Processing Systems*. 7805–7815.

Ellis, K., D. Ritchie, A. Solar-Lezama, and J. Tenenbaum. (2018b). “Learning to infer graphics programs from hand-drawn images”. In: *Advances in neural information processing systems*. 6059–6068.

Ellis, K., C. Wong, M. Nye, M. Sable-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum. (2020). “Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning”. *arXiv preprint arXiv:2006.08381*.

Fichte, J. K., M. Hecher, and M. Zisser. (2019). “An improved GPU-based SAT model counter”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 491–509.

Finn, C., P. Abbeel, and S. Levine. (2017). “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *ICML*.

Fulton, N. and A. Platzer. (2018). “Safe reinforcement learning via formal methods: Toward safe control through proof and learning”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Garcez, A. S. d., K. B. Broda, and D. M. Gabbay. (2002). *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.

Garcez, A. S. d’Avila and G. Zaverucha. (1999). “The Connectionist Inductive Learning and Logic Programming System”. *Appl. Intell.* 11(1): 59–77. DOI: 10.1023/A:1008328630915.

Garcıa, J. and F. Fernández. (2015). “A comprehensive survey on safe reinforcement learning”. *Journal of Machine Learning Research*. 16(1): 1437–1480.

Gaunt, A. L., M. Brockschmidt, R. Singh, N. Kushman, P. Kohli, J. Taylor, and D. Tarlow. (2016). “Terpret: A probabilistic programming language for program induction”. *arXiv preprint arXiv:1608.04428*.

Gaunt, A. L., M. Brockschmidt, N. Kushman, and D. Tarlow. (2017). “Differentiable Programs with Neural Libraries”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 1213–1222.

Gehr, T., M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. (2018). “AI^2: Safety and robustness certification of neural networks with abstract interpretation”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 3–18.

Giles, C. L., C. B. Miller, D. Chen, H.-H. Chen, G.-Z. Sun, and Y.-C. Lee. (1992). "Learning and extracting finite state automata with second-order recurrent neural networks". *Neural Computation*. 4(3): 393–405.

Graves, A., G. Wayne, and I. Danihelka. (2014). "Neural turing machines". *arXiv preprint arXiv:1410.5401*.

Grefenstette, E., K. M. Hermann, M. Suleyman, and P. Blunsom. (2015). "Learning to transduce with unbounded memory". *Advances in neural information processing systems*. 28: 1828–1836.

Gulwani, S., O. Polozov, and R. Singh. (2017). "Program synthesis". *Foundations and Trends in Programming Languages*. 4(1-2): 1–119.

Hamadi, Y. and C. Wintersteiger. (2012). "Seven challenges in parallel SAT solving". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. No. 1.

Harnad, S. (1990). "The symbol grounding problem". *Physica D: Nonlinear Phenomena*. 42(1-3): 335–346.

Hendrycks, D., M. Mazeika, S. Kadavath, and D. Song. (2019). "Using self-supervised learning can improve model robustness and uncertainty". In: *Advances in Neural Information Processing Systems*. 15663–15674.

Herzig, J., P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. (2020). "TAPAS: Weakly Supervised Table Parsing via Pre-training". *arXiv preprint arXiv:2004.02349*.

Hewitt, L. B., T. A. Le, and J. B. Tenenbaum. (2020). "Learning to learn generative programs with Memoised Wake-Sleep". *UAI*.

Hinton, G. E., O. Vinyals, and J. Dean. (2015). "Distilling the Knowledge in a Neural Network". *CoRR*. abs/1503.02531. arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531.

Hu, R., J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. (2017). "Learning to reason: End-to-end module networks for visual question answering". In: *Proceedings of the IEEE International Conference on Computer Vision*. 804–813.

Hutter, F., D. Babic, H. H. Hoos, and A. J. Hu. (2007). "Boosting Verification by Automatic Tuning of Decision Procedures". In: *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*. IEEE Computer Society. 27–34.

Iyer, A., M. Jonnalagedda, S. Parthasarathy, A. Radhakrishna, and S. K. Rajamani. (2019). "Synthesis and machine learning for heterogeneous extraction". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 301–315.

James, S., P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. (2019). "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 12627–12637.

Jha, S., S. Gulwani, S. A. Seshia, and A. Tiwari. (2010). "Oracle-guided component-based program synthesis". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1. ICSE '10*. Cape Town, South Africa: ACM. 215–224.

Ji, R., Y. Sun, Y. Xiong, and Z. Hu. (2020). "Guiding Dynamic Programing via Structural Probability for Accelerating Programming by Example". *Proc. ACM Program. Lang.* 4(OOPSLA).

Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.

Kalyan, A., A. Mohta, O. Polozov, D. Batra, P. Jain, and S. Gulwani. (2018). "Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Karimi, A. and P. S. Duggirala. (2020). "Formalizing traffic rules for uncontrolled intersections". In: *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. 41–50.

Katz, G., C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. (2017). "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". *CoRR*. abs/1702.01135.

Keysers, D., N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. van Zee, and O. Bousquet. (2020). "Measuring Compositional Generalization: A Comprehensive Method on Realistic Data". In: *International Conference on Learning Representations*.

Kim, B., M. Wattenberg, J. Gilmer, C. J. Cai, J. Wexler, F. B. Viégas, and R. Sayres. (2018). "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy and A. Krause. Vol. 80. *Proceedings of Machine Learning Research*. PMLR. 2673–2682.

Kingma, D. P. and M. Welling. (2013). "Auto-encoding variational Bayes". *arXiv preprint arXiv:1312.6114*.

Krawiec, K. (2016). *Behavioral program synthesis with genetic programming*. Vol. 618. Springer.

Le Goues, C., T. Nguyen, S. Forrest, and W. Weimer. (2011). "Genprog: A generic method for automatic software repair". *IEEE transactions on software engineering*. 38(1): 54–72.

Lerda, F. and R. Sisto. (1999). "Distributed-memory model checking with SPIN". In: *International SPIN Workshop on Model Checking of Software*. Springer. 22–39.

Levine, S. and V. Koltun. (2013). "Guided policy search". In: *International Conference on Machine Learning*. 1–9.

Liang, P., M. I. Jordan, and D. Klein. (2010). "Learning programs: A hierarchical Bayesian approach". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 639–646.

Liu, A., G. Shi, S.-J. Chung, A. Anandkumar, and Y. Yue. (2020). "Robust regression for safe exploration in control". In: *Learning for Dynamics and Control*. PMLR. 608–619.

Liu, D., H. Zhang, F. Wu, and Z.-J. Zha. (2019a). "Learning to Assemble Neural Module Tree Networks for Visual Grounding". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Liu, H., K. Simonyan, and Y. Yang. (2019b). “DARTS: Differentiable Architecture Search”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=S1eYHoC5FX.

Liu, Y., Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan. (2021). “A survey on evolutionary neural architecture search”. *IEEE Transactions on Neural Networks and Learning Systems*.

Manhaeve, R., S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. (2018). “DeepProbLog: Neural probabilistic logic programming”. *Advances in Neural Information Processing Systems*. 31: 3749–3759.

Manna, Z. and R. J. Waldinger. (1971). “Toward Automatic Program Synthesis”. *Communications of the ACM*. 14(3): 151–165.

Mao, J., X. Zhang, Y. Li, W. T. Freeman, J. B. Tenenbaum, and J. Wu. (2019). “Program-guided image manipulators”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 4030–4039.

Marcus, G. and E. Davis. (2019). *Rebooting AI: Building artificial intelligence we can trust*. Pantheon.

Moldovan, T. M. and P. Abbeel. (2012). “Safe Exploration in Markov Decision Processes”. In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress. URL: http://icml.cc/2012/papers/838.pdf.

Mooney, R. J. (2008). “Learning to Connect Language and Perception.” In: *AAAI*. 1598–1601.

Moura, L. M. de and N. Bjørner. (2008). “Z3: An Efficient SMT Solver”. In: *TACAS*. 337–340.

Murali, A. and P. Madhusudan. (2019). “Augmenting Neural Nets with Symbolic Synthesis: Applications to Few-Shot Learning”. *CoRR*. abs/1907.05878. arXiv: 1907.05878. URL: http://arxiv.org/abs/1907.05878.

Murali, V., L. Qi, S. Chaudhuri, and C. Jermaine. (2018). “Neural Sketch Learning for Conditional Program Generation”. *ICLR*.

Neal, R. M. (2011). “MCMC Using Hamiltonian Dynamics”. In: *Handbook of Markov Chain Monte Carlo*. CRC Press. Chap. 5. DOI: 10.1201/b10905-7.

Parisotto, E., A. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli. (2017). "Neuro-Symbolic Program Synthesis". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Polozov, O. and S. Gulwani. (2015). "FlashMeta: a framework for inductive program synthesis". In: *ACM SIGPLAN Notices*. Vol. 50. No. 10. ACM. 107–126.

Ratner, A. J., C. M. De Sa, S. Wu, D. Selsam, and C. Ré. (2016). "Data programming: Creating large training sets, quickly". In: *Advances in neural information processing systems*. 3567–3575.

Real, E., C. Liang, D. So, and Q. Le. (2020). "Automl-zero: Evolving machine learning algorithms from scratch". In: *International Conference on Machine Learning*. PMLR. 8007–8019.

Reed, S. and N. De Freitas. (2015). "Neural programmer-interpreters". *arXiv preprint arXiv:1511.06279*.

Riedel, S., M. Bosnjak, and T. Rocktäschel. (2016). "Programming with a differentiable Forth interpreter". *CoRR, abs/1605.06640*.

Sachan, M. and E. Xing. (2018). "Self-training for jointly learning to ask and answer questions". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 629–640.

Sahoo, S. S., S. Venugopalan, L. Li, R. Singh, and P. F. Riley. (2020). "Scaling Symbolic Methods using Gradients for Neural Model Explanation". *CoRR*. abs/2006.16322. URL: https://arxiv.org/abs/2006.16322.

Shah, A., E. Zhan, J. J. Sun, A. Verma, Y. Yue, and S. Chaudhuri. (2020). "Learning Differentiable Programs with Admissible Neural Heuristics". In: *Advances in Neural Information Processing Systems*.

Shavlik, J. W. (1994). "Combining symbolic and neural learning". *Machine Learning*. 14(3): 321–331.

Shin, R., M. Allamanis, M. Brockschmidt, and O. Polozov. (2019a). "Program synthesis and semantic parsing with learned code idioms". In: *Advances in Neural Information Processing Systems*. 10825–10835.

Shin, R., N. Kant, K. Gupta, C. Bender, B. Trabucco, R. Singh, and D. Song. (2019b). "Synthetic Datasets for Neural Program Synthesis". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=ryeOSnAqYm.

Si, X., M. Raghothaman, K. Heo, and M. Naik. (2019). "Synthesizing datalog programs using numerical relaxation". In: *IJCAI*.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. (2016). "Mastering the Game of Go with Deep Neural Networks and Tree Search". *Nature*. 529(7587): 484–489. DOI: 10.1038/nature16961.

Smolensky, P., M. Lee, X. He, W. Yih, J. Gao, and L. Deng. (2016). "Basic Reasoning with Tensor Product Representations". *CoRR*. abs/1601.02745. arXiv: 1601.02745. URL: http://arxiv.org/abs/1601.02745.

Solar-Lezama, A. (2009). "The Sketching Approach to Program Synthesis". In: *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings*. 4–13. DOI: 10.1007/978-3-642-10672-9_3.

Solar-Lezama, A., L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. (2006). "Combinatorial sketching for finite programs". In: *ASPLOS*. 404–415.

Srivastava, S., O. Polozov, N. Jojic, and C. Meek. (2020). "Learning Web-based Procedures by Reasoning over Explanations and Demonstrations in Context". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7652–7662.

Sun, J. J., A. Kennedy, E. Zhan, D. J. Anderson, Y. Yue, and P. Perona. (2021). "Task programming: Learning data efficient behavior representations". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2876–2885.

Sun, J. J., A. Kennedy, E. Zhan, Y. Yue, and P. Perona. (2020). "Task Programming: Learning Data Efficient Behavior Representations". *arXiv preprint arXiv:2011.13917*.

Sun, R. and F. Alexandre. (2013). *Connectionist-symbolic integration: From unified to hybrid approaches*. Psychology Press.

Sundararajan, M., A. Taly, and Q. Yan. (2017). "Axiomatic Attribution for Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. *Proceedings of Machine Learning Research*. PMLR. 3319–3328.

Sutton, R. S., D. Precup, and S. Singh. (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". *Artificial intelligence*. 112(1-2): 181–211.

Taori, R., A. Dave, V. Shankar, N. Carlini, B. Recht, and L. Schmidt. (2020). "Measuring robustness to natural distribution shifts in image classification". *Advances in Neural Information Processing Systems*. 33.

Tavares, Z., J. Burroni, E. Minasyan, A. Solar-Lezama, and R. Ranganath. (2019). "Predicate Exchange: Inference with Declarative Knowledge". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. *Proceedings of Machine Learning Research*. PMLR. 6186–6195. URL: http://proceedings.mlr.press/v97/tavares19a.html.

Taylor, L. and G. Nitschke. (2017). "Improving deep learning using generic data augmentation". *arXiv preprint arXiv:1708.06020*.

Tjandrasuwita, M., J. J. Sun, A. Kennedy, S. Chaudhuri, and Y. Yue. (2021). "Interpreting Expert Annotation Differences in Animal Behavior". *CoRR*. abs/2106.06114. arXiv: 2106.06114. URL: https://arxiv.org/abs/2106.06114.

Torlak, E. and R. Bodik. (2013). "Growing solver-aided languages with Rosette". In: *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*. ACM. 135–152.

Towell, G. G. and J. W. Shavlik. (1993). "Extracting refined rules from knowledge-based neural networks". *Machine learning*. 13(1): 71–101.

Towell, G. G., J. W. Shavlik, and M. O. Noordewier. (1990). "Refinement of approximate domain theories by knowledge-based neural networks". In: *Proceedings of the eighth National conference on Artificial intelligence*. Vol. 861866. Boston, MA.

Udrescu, S.-M. and M. Tegmark. (2020). "AI Feynman: A physics-inspired method for symbolic regression". *Science Advances*. 6(16): eaay2631.

Udupa, A., A. Raghavan, J. V. Deshmukh, S. Mador-Haim, M. M. Martin, and R. Alur. (2013). "TRANSIT: specifying protocols with concolic snippets". *ACM SIGPLAN Notices*. 48(6): 287–296.

Valiant, L. G. (1984). "A theory of the learnable". *Communications of the ACM*. 27(11): 1134–1142.

Valkov, L., D. Chaudhari, A. Srivastava, C. Sutton, and S. Chaudhuri. (2018). "HOUDINI: Lifelong Learning as Program Synthesis". In: *Advances in Neural Information Processing Systems*. 8701–8712.

Vedantam, R., K. Desai, S. Lee, M. Rohrbach, D. Batra, and D. Parikh. (2019). "Probabilistic Neural-symbolic Models for Interpretable Visual Question Answering". In: *ICML*.

Verma, A., H. M. Le, Y. Yue, and S. Chaudhuri. (2019). "Imitation-Projected Programmatic Reinforcement Learning". In: *Neural Information Processing Systems (NeurIPS)*.

Verma, A., V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. (2018). "Programmatically Interpretable Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 5052–5061.

Wang, P., P. L. Donti, B. Wilder, and J. Z. Kolter. (2019). "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 6545–6554.

Wang, X. and J. Schneider. (2014). "Flexible transfer learning under support and model shift". *Advances in Neural Information Processing Systems*. 27: 1898–1906.

Xu, D., S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. (2018). "Neural task programming: Learning to generalize across hierarchical tasks". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 3795–3802.

Yosinski, J., J. Clune, Y. Bengio, and H. Lipson. (2014). "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 3320–3328.

Young, H., O. Bastani, and M. Naik. (2019). "Learning neurosymbolic generative models via program synthesis". *arXiv preprint arXiv:1901.08565*.

Zhan, E., J. J. Sun, A. Kennedy, Y. Yue, and S. Chaudhuri. (2021). "Unsupervised Learning of Neurosymbolic Encoders". *CoRR*. abs/2107.13132. arXiv: 2107.13132. URL: https://arxiv.org/abs/2107.13132.

Zhan, E., A. Tseng, Y. Yue, A. Swaminathan, and M. Hausknecht. (2020). "Learning Calibratable Policies using Programmatic Style-Consistency". In: *International Conference on Machine Learning*. PMLR. 11001–11011.

Zhang, X., A. Solar-Lezama, and R. Singh. (2018). "Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. 4879–4890.

Zhu, H., Z. Xiong, S. Magill, and S. Jagannathan. (2019). "An inductive synthesis framework for verifiable reinforcement learning". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*. Ed. by K. S. McKinley and K. Fisher. ACM. 686–701.