From Typestate Verification to Interpretable Deep Models (Invited Talk Abstract)

Eran Yahav
Technion, Israel
yahave@cs.technion.ac.il

Stephen J. Fink
Facebook, USA
stephenfink@fb.com

Nurit Dor
Kayhut, Israel
nurit.dor@gmail.com

G. Ramalingam
Microsoft Research, USA
grama@microsoft.com

Emmanuel Geay
Wayfair, USA
emlgeay@gmail.com

ABSTRACT

The paper "Effective Typestate Verification in the Presence of Aliasing" was published in the International Symposium on Software Testing and Analysis (ISSTA) 2006 Proceedings, and has now been selected to receive the ISSTA 2019 Retrospective Impact Paper Award. The paper described a scalable framework for verification of typestate properties in real-world Java programs. The paper introduced several techniques that have been used widely in the static analysis of real-world programs. Specifically, it introduced an abstract domain combining access-paths, aliasing information, and typestate that turned out to be simple, powerful, and useful. We review the original paper and show the evolution of the ideas over the years. We show how some of these ideas have evolved into work on machine learning for code completion, and discuss recent general results in machine learning for programming.

CCS CONCEPTS

• **Software and its engineering → Automated static analysis**;
• **Computing methodologies → Neural networks**.

KEYWORDS

program analysis, program synthesis, machine learning

ACM Reference Format:
Eran Yahav, Stephen J. Fink, Nurit Dor, G. Ramalingam, and Emmanuel Geay. 2019. From Typestate Verification to Interpretable Deep Models (Invited Talk Abstract). In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3293882.3338992

1 OVERVIEW

The goal of the original paper was to verify typestate properties over real-world Java programs [6]. This idea then evolved into the task of *static specification mining*, learning specifications from individual programs [10] and then into learning from partial programs for the purpose of code completion [7].

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07.
https://doi.org/10.1145/3293882.3338992

```
String[] f(final String[] array) {
 final String[] newArray = new String[array.length];
 for (int index = 0; index < array.length; index++) {
  newArray[array.length - index - 1] = array[index];
 }
 return newArray;
}
```

Predictions	
reverseArray	77.34%
reverse	18.18%
subArray	1.45%
copyArray	0.74%

Figure 1: A code snippet and its predicted labels as computed by our model.

The static analysis that powered these works, as well as many others, used a combination of local flow-sensitive access-paths and typestate together with global flow-insensitive aliasing information. This abstract domain borrowed ideas from shape analysis [9].

In this talk, we will describe the journey from deep analysis of individual programs, into the realm of a more shallow analysis over a large number of programs, accompanied by deep learning. We will discuss the tradeoffs between analysis effort and learning effort, as well as recent results on machine learning for programming [3–5]. Then, we will show how to extract an automaton from a given recurrent neural network, providing some insight on what a network has actually learned [11, 12].

1.1 Motivating Tasks

Semantic labeling of code snippets. Consider the snippet of Figure 1. This snippet only contains low-level assignments to arrays, but a human reading the code may (correctly) label it as performing the *reverse* operation. Our goal is to predict such labels automatically. The right hand side of Figure 1 shows the labels predicted using our approach. The most likely prediction (77.34%) is *reverseArray*. Alon et al. [5] provide additional examples. Intuitively, this problem is hard because it requires *learning a correspondence* between the *entire content of a code snippet* and a semantic label. That is, it requires aggregating possibly hundreds of expressions and statements into a single, descriptive label.

Captioning code snippets. The goal of *code captioning* is to assign a natural language caption that describes the snippet. For the code in Figure 2 our approach predicts the caption *"get the text of a pdf file in C#"*. Intuitively, this task is harder than semantic labeling, as it requires the generation of a natural language sentence in addition to capturing the meaning of the code snippet.

```
iTextSharp.text.pdf.PdfReader reader = new iTextSharp.text.pdf.PdfReader(
    new iTextSharp.text.pdf.RandomAccessFileOrArray(@"C:\PDFFile.pdf"), null);
```

Prediction: *get the text of a pdf file in C#*

Figure 2: A code snippet and its predicted caption as computed by our model.

```
OkHttpClient ok = new OkHttpClient();
Request request = new Builder().url(url).build();
Response response =
```

Prediction: `ok.newCall(request).execute()`

Figure 3: A snippet and its predicted completion as computed by our model.

Code Completion. Consider the code of Figure 3. Our code completion automatically predicts the next steps in the code: `ok.newCall(request).execute()`. This task requires prediction of the missing part of the code based on a given context. Technically, this can be expressed as predicting a completion of a partial abstract syntax tree.

In the talk, we show how techniques based on neural networks address all of these tasks, as well as other programming tasks.

2 FROM PROGRAMS TO DEEP MODELS

Leveraging machine learning models for predicting program properties such as variable names, method names, and expression types is a topic of much recent interest (e.g.,[1, 2, 8]). These techniques are based on learning a statistical model from a large amount of code and using the model to make predictions in new programs. A major challenge in these techniques is how to represent instances of the input space to facilitate learning. Designing a program representation that enables effective learning is a critical task that is *often done manually for each task and programming language.*

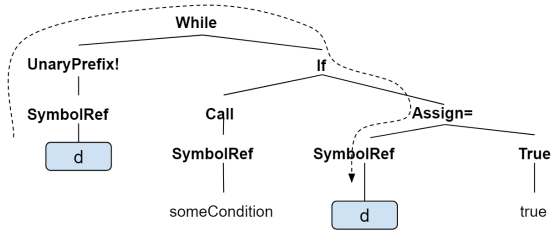
Our approach We present a program representation for learning from programs. Our approach uses different *path-based abstractions of the program's abstract syntax tree.* This family of path-based representations is natural, general, fully automatic, and works well across different tasks and programming languages.

AST paths We define AST paths as paths between nodes in a program's abstract syntax tree (AST). To automatically generate paths, we first parse the program to produce an AST, and then extract paths between nodes in the tree. We represent a path in the AST as a sequence of nodes connected by up and down movements, and represent a program element as the set of paths that its occurrences participate in. Fig. 4a shows an example JavaScript program. Fig. 4b shows its AST, and one of the extracted paths. The path from the first occurrence of the variable `d` to its second occurrence can be represented as: SymbolRef ↑ UnaryPrefix! ↑ While ↓ If ↓ Assign= ↓ SymbolRef . This is an example of a pairwise path between leaves in the AST, but in general the family of path-based representations is much wider. We consider several choices of subsets of this family in [4].

In the talk, I will describe our recent work on CODE2SEQ [3] which generates natural language sequences from structured representations of code.

```
while (!d) {
  if (someCondition()) {
    d = true;
  }
}
```

(a) A simple JavaScript program.



(b) The program's AST, and a path.

Figure 4: A program and its AST, and an example of one of the paths.

REFERENCES

[1] Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. 2015. Suggesting Accurate Method and Class Names. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015).*

[2] Miltiadis Allamanis, Hao Peng, and Charles A. Sutton. 2016. A Convolutional Attention Network for Extreme Summarization of Source Code. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016.* http://jmlr.org/proceedings/papers/v48/allamanis16.html

[3] Uri Alon, Omer Levy, and Eran Yahav. 2019. Code2Seq: Generating Sequences from Structured Representations of Code. In ***ICLR'19****: International Conference on Learning Representations.*

[4] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2018. A General Path-Based Representation for Predicting Program Properties. In *PLDI'18: Proceedings of the Conference on Programming Language Design and Implementation.*

[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: learning distributed representations of code. *PACMPL* 3, POPL (2019), 40:1–40:29. https://doi.org/10.1145/3290353

[6] Stephen J. Fink, Eran Yahav, Nurit Dor, G. Ramalingam, and Emmanuel Geay. [n.d.]. Effective typestate verification in the presence of aliasing. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2006.* https://doi.org/10.1145/1146238.1146254

[7] Alon Mishne, Sharon Shoham, and Eran Yahav. 2012. Typestate-based semantic code search over partial programs. In *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012.* https://doi.org/10.1145/2384616.2384689

[8] Veselin Raychev, Martin Vechev, and Andreas Krause. [n.d.]. Predicting Program Properties from "Big Code". In *Proceedings of the SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15).*

[9] Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. 2002. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.* 24, 3 (2002).

[10] Sharon Shoham, Eran Yahav, Stephen Fink, and Marco Pistoia. [n.d.]. Static specification mining using automata-based abstractions. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2007.* https://doi.org/10.1145/1273463.1273487

[11] Gail Weiss, Yoav Goldberg, and Eran Yahav. [n.d.]. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In *Proceedings of the International Conference on Machine Learning, ICML 2018.*

[12] Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the Practical Computational Power of Finite Precision RNNs for Language Recognition. In *ACL'18: 56th Annual Meeting of the Association for Computational Linguistics.*