Semantic Representation of Data Science Programs

Evan Patterson[1,2], **Ioana Baldini**[2], **Aleksandra Mojsilović**[2], **Kush R. Varshney**[2]
[1] Stanford University
[2] IBM Research AI
epatters@stanford.edu, {ioana, aleksand, krvarshn}@us.ibm.com,

1 Introduction

Your computer—through which you are, in all likelihood, reading this paper—is continuously, efficiently, and reliably executing computer programs, but does it really understand them? Not in any meaningful sense. That burden falls upon human knowledge workers, who are increasingly asked to write and understand code. They would benefit greatly from intelligent tools that reveal the connections between their code, their colleagues' code, and the subject-matter concepts to which the code implicitly refers and to which their real enthusiasm belongs. By teaching machines to comprehend code, we could create artificial agents that empower human knowledge workers or perhaps even generate useful programs of their own.

One computational domain undergoing rapid growth is data science. Besides the usual problems facing the scientist-turned-programmer, the data scientist must contend with a proliferation of programming languages (like Python, R, and Julia) and frameworks (too numerous to recount). Data science therefore presents an especially compelling target for machine understanding of computer code. An AI agent that simultaneously comprehends the generic concepts of computing and the specialized concepts of data science could prove enormously useful, for example to debug and visualize machine learning workflows or automatically summarize data analyses as natural text for human readers.

Towards this vision, we propose and implement an AI system that forms semantic representations of computer programs in a particular subject-matter domain. We will focus on applications to data science because we, the authors, are all data scientists of various stripes. Nevertheless, we think that our methodology could be fruitfully applied to other scientific domains with a heavy computational focus, such as bioinformatics or computational linguistics.

2 An example

Before explaining how our method works, we illustrate it with a simple example. Two versions of a toy data analysis, both written in Python, are shown in Listings 1 and 2. The first is implemented using the scientific computing packages NumPy and SciPy; the second using the data science packages Pandas and Scikit-learn. The two programs perform the same computation: they read the Iris dataset from a CSV file, drop the last column (labeling the flower species), fit a k-means clustering model with three clusters to the remaining columns, and return the cluster assignments and centroids. The programs are thus syntactically distinct but semantically equivalent.

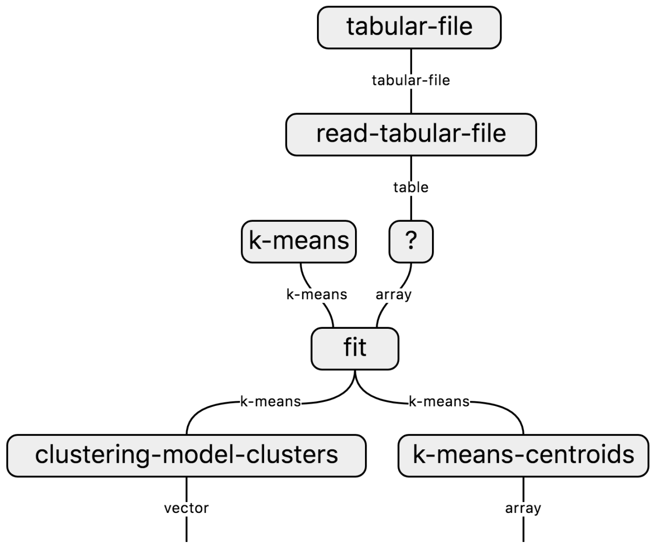

Figure 1: Semantic flow graph for both versions of k-means clustering analysis (Listings 1 and 2)

Identifying this semantic equivalence, our system furnishes the same semantic representation for both programs, the dataflow graph shown in Figure 1. The labeled nodes and edges refer to concepts in an ontology. The node tagged with a question mark refers to code with unknown semantics.

We restrict ourselves to this simple example for reasons of space and good pedagogy. However, we emphasize that our system is capable of treating complex, real-world programs. For an early example, see Figure 1 in [Patterson *et al.*, 2017].

3 Ideas and techniques

We now explain, as fully as space permits, our method of constructing semantic representations of computer programs. It is summarized in Figure 2.

Our method outputs two major artifacts, the raw and semantic flow graphs. Both dataflow graphs capture the execution of a computer program doing data analysis, but at

```
import numpy as np
from scipy.cluster.vq import kmeans2

iris = np.genfromtxt('iris.csv',
  dtype='f8', delimiter=',', skip_header=1)
iris = np.delete(iris, 4, axis=1)

centroids, clusters = kmeans2(iris, 3)
```

Listing 1: k-means clustering via NumPy and SciPy

```
import pandas as pd
from sklearn.cluster import KMeans

iris = pd.read_csv('iris.csv')
iris = iris.drop('Species', 1)

kmeans = KMeans(n_clusters=3)
kmeans.fit(iris.values)
centroids = kmeans.cluster_centers_
clusters = kmeans.labels_
```

Listing 2: k-means clustering via Pandas and Scikit-learn

different levels of abstraction. The *raw flow graph* records the concrete function calls made by the program. This graph is language and library dependent. The *semantic flow graph* describes the same program in terms of abstract concepts belonging to a formal ontology about data science. This graph is language and library independent. In our example, Figure 1 is a semantic flow graph. The raw flow graphs for Listings 1 and 2 are not shown.

Our method generates these artifacts in two major steps (Figure 2). First, dynamic *program analysis* distills the raw flow graph from a computer program. The *semantic enrichment* algorithm then transforms the raw flow graph into the semantic flow graph. This algorithm, and its associated ontology and ontology language, are the main contributions of our work.

Semantic enrichment is supported by a new *ontology* (or *knowledge base*) about data science, called the Data Science Ontology. It contains two types of knowledge: concepts and annotations. *Concepts* formalize the abstract ideas of machine learning, statistics, and computing on data. The semantic flow graph has semantics, as its name suggests, because its nodes and edges are linked to concepts. *Annotations* map code from data science libraries, such as Pandas and Scikit-learn, onto concepts. During semantic enrichment, annotations translate concrete functions in the raw flow graph into abstract functions in the semantic flow graph.

The ontology itself is written in a new *ontology language* called the MONoidal Ontology and Computing Language (Monocl). To cleanly model computer programs, our language combines ideas from category theory, such as string diagrams [Selinger, 2010], and ideas from type theory, such as implicit conversions [Reynolds, 1980], thus exploiting the close connection between the two subjects [Crole, 1993; Jacobs, 1999]. We see the ontology language as belonging to an emerging paradigm of categorical knowledge representation [Spivak and Kent, 2012; Patterson, 2017].

We raise an important methodological point about the role of annotations in semantic enrichment. Our system is fully automated inasmuch as it expects no special input from end users. It does depend on annotations of commonly used software packages. While that requires some human effort, it is negligible compared to the usual effort of creating, maintaining, and documenting software packages.

4 Related work

The history of AI is replete with interactions between knowledge representation and computer program analysis. Methods as diverse as automated planning, expert systems, description logic, and graph parsing have all featured in "knowledge-based program analysis" [Johnson and Soloway, 1985; Harandi and Ning, 1990; Devanbu *et al.*, 1991; Wills, 1992; Welty, 2007]. These projects are supposed to help software developers maintain large codebases in specialized industrial domains like telecommunications.

Our research goals are less ambitious in scale but also, we hope, more tractable. We focus on knowledge workers who write short, semantically rich scripts, without the endless layers of abstraction found in large codebases. In data science, the code tends to be much shorter, the control flow more linear, and the underlying concepts better defined, than in large-scale industrial software. Our methodology is accordingly quite different from that of the older literature.

This work extends our earlier paper [Patterson *et al.*, 2017] by introducing a new ontology, ontology language, and semantic enrichment algorithm.

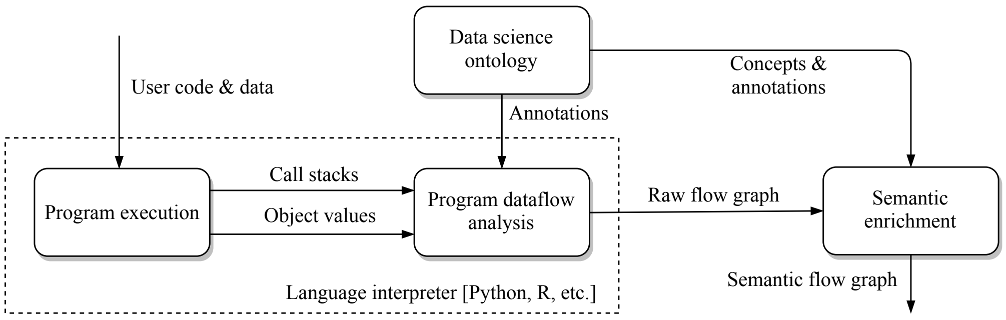

Figure 2: System architecture

References

[Crole, 1993] Roy L. Crole. *Categories for Types*. Cambridge University Press, 1993.

[Devanbu *et al.*, 1991] Prem Devanbu, Ron Brachman, Peter G. Selfridge, and Bruce W. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 34(5):34–49, 1991.

[Harandi and Ning, 1990] Mehdi T. Harandi and Jim Q. Ning. Knowledge-based program analysis. *IEEE Software*, 7(1):74–81, 1990.

[Jacobs, 1999] Bart Jacobs. *Categorical logic and type theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999.

[Johnson and Soloway, 1985] W. Lewis Johnson and Elliot Soloway. PROUST: Knowledge-based program understanding. *IEEE Transactions on Software Engineering*, (3):267–275, 1985.

[Patterson *et al.*, 2017] Evan Patterson, Robert McBurney, Holly Schmidt, Ioana Baldini, Aleksandra Mojsilović, and Kush R. Varshney. Dataflow representation of data analyses: Towards a platform for collaborative data science. *IBM Journal of Research and Development*, 61(6):9:1–9:13, 2017.

[Patterson, 2017] Evan Patterson. Knowledge representation in bicategories of relations. arXiv:1706.00526, 2017.

[Reynolds, 1980] John C. Reynolds. Using category theory to design implicit conversions and generic operators. In *International Workshop on Semantics-Directed Compiler Generation*, pages 211–258, 1980.

[Selinger, 2010] Peter Selinger. A survey of graphical languages for monoidal categories. In *New Structures for Physics*, pages 289–355. Springer, 2010.

[Spivak and Kent, 2012] David Spivak and Robert Kent. Ologs: a categorical framework for knowledge representation. *PLoS One*, 7(1), 2012.

[Welty, 2007] Christopher A. Welty. Software engineering. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 11, pages 402–416. Cambridge University Press, 2 edition, 2007.

[Wills, 1992] Linda M. Wills. Automated program recognition by graph parsing. Technical report, MIT Artificial Intelligence Laboratory, 1992.