A Common Graph Representation for Source Code and Natural Language

Breandan Consididne

Abstract

Semantic information plays a key role in the code search and synthesis settings. In this work, we propose a graph-based representation for code search and document recommendation which incorporates semantic and relational features from the type system and dataflow graph. We connect this graph to a natural language corpus of developer documents, and demonstrate the effectiveness of a graph-based representation on three downstream tasks: code search, document recommendation and entity linking within developer documentation.

1 Introduction

In addition to its syntactic features, source code contains a rich denotational and operational semantics [?]. We propose a representation of source code which incorporates semantic features from the call graph as well as hierarchical information about the typing context in which it occurs. This allows us to reason about program transformations in semantically similar but syntactically distinct fragments. Some semantic relationships, such as data or control flow may be represented by a directed acyclic graph (DAG) which admits linear-time solutions to a broad class of problems, including topological sorting, single-source shortest path and reachability queries.

Many programming languages allow the user to specify which types of values will inhabit a given variable, allowing tools to perform efficient static analyses such as nullity [Ekman and Hedin, 2007] and shape inference [con, 2019] without executing the program. Although types may not explicitly occur inside text, they can be often be inferred from the surrounding context using a DAG-based representation. Java, one of the most popular programming languages, recently adopted a feature called local variable type inference Liddell and Kim [2019], in which variable types can be omitted and inferred by the compiler in a straightforward manner.

DAGs have applications in natural languages as well [Sagae and Tsujii, 2008, Quernheim and Knight, 2012]. Consider the sentences in the previous paragraph, which can be rewritten in any order without changing its semantic meaning. In contrast, these sentences have a meaningful sequential entailment, and shuffling them would make the paragraph much less coherent. Thus, it is important to capture and represent permutation-invariant features when building language models. [?] Doing so allows us to model semantic relationships between and among related entities in natural language, such as co-references, semantic entailment and other features which are difficult to capture with a purely sequence-based approach.

Source code for widely used projects is often accompanied by web-based developer documentation. Such documentation often takes the form of HTML or Markdown-structured text, which is syntactically an AST and often contains links to other documents, which can be arranged to form a directed graph between documents. Both the link graph and the AST of the parent document contain useful information: the document tree contains information about the location of the text in relation to the other sections and defines the layout of text in a browser.

2 Prior work

Si et al. [2018] introduce a control-flow representation for source code which incorporates elements of the syntax tree and surrounding context.

3 Method

In order to relate the graph of documents to source code, a heuristic is needed. For source code, a good heuristic is the presence of an unambiguous token. This token can be a code-like fragment or other entity such as text.

It is often the case that two documents share a common token. If the token is rare, the co-occurence indicates they refer to a common entity. But which entity? In order to determine the referent, we need a representation of the surrounding context but a representation of the surrounding context. While many documents occassionally link to source code directly, source code very seldomly contains links to a HTML document.

We would like to infer which documents which are relevant to a particular section of code, based on the graph of documents and the graph of code. To infer links between these two domains requires building a multi-relational graph representation. We also need an AST of statically typed computer programs from GitHub. We choose Kotlin, which has a variety of parsing tools for source code [Kovalenko et al., 2019] and natural langauge [**?**].

3.1 Dataset

References

Kotlin∇: A shape-safe eDSL for differentiable programming. `https://github.com/breandan/kotlingrad`, 2019.

Torbjörn Ekman and Görel Hedin. Pluggable checking and inferencing of nonnull types for java. *Journal of Object Technology*, 6(9):455–475, 2007.

Vladimir Kovalenko, Egor Bogomolov, Timofey Bryksin, and Alberto Bacchelli. Pathminer: a library for mining of path-based representations of code. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 13–17. IEEE Press, 2019.

Clayton Liddell and Donghoon Kim. Analyzing the adoption rate of local variable type inference in open-source java 10 projects. *Journal of the Arkansas Academy of Science*, 73(1):51–54, 2019.

Daniel Quernheim and Kevin Knight. Dagger: A toolkit for automata on directed acyclic graphs. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 40–44, 2012.

Kenji Sagae and Jun'ichi Tsujii. Shift-reduce dependency dag parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 753–760. Association for Computational Linguistics, 2008.

Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Advances in Neural Information Processing Systems*, pages 7751–7762, 2018.