
Learning to parse developer documentation

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 In this work, we propose a regular expression (regex) synthesizer for entity
2 linking in source code and API documentation which incorporates semantic
3 and relational features from the surrounding context. We demonstrate the
4 effectiveness of our synthesizer on a link prediction task using a corpus of
5 Java code and developer documentation, and demonstrate the effectiveness
6 of model-based representation learning for interpretable link prediction in
7 the source-to-source and doc-to-doc setting.

1 Introduction

9 Semantic information plays a key role in both natural and programming languages. In addition to its syntax, source code contains a rich denotational and operational semantics [Henkel et al., 2018]. To effectively reason about code in semantically similar but syntactically diverse settings requires models which incorporate features from the call graph [Gu et al., 2016, 2018, Liu et al., 2019] and surrounding typing context [Allamanis et al., 2017]. Many semantic features, such as data and control flow [Si et al., 2018] can be represented using a directed acyclic graph (DAG), which admits linear-time solutions to a variety of graph problems, including topological sorting, single-source shortest path and reachability queries.

17 The field of natural language has also developed a rich set of graph-based representations, including Reddy et al. [2016]’s and other typed attribute grammars which can be used to reason about syntactic and semantic relations between natural language entities. In the pointer network architecture, Vinyals et al. [2015b,a] emphasize the importance of constructing permutation-invariant representations and show SOTA improvements in semantic labeling tasks from dependency parsing [Ma et al., 2018], named-entity recognition [Lample et al., 2016], and coreference resolution where sequence-based techniques often struggle. Pointer networks have been recently extended with a copy-mechanism [Li et al., 2017] to handle out-of-vocabulary code tokens.

26 These tools can be used for studying both source code [Allamanis et al., 2017] and documentation [Yang et al., 2016]. Entity alignment in doc-to-doc (D2D) and source-to-source (S2S) is a straightforward application of existing link prediction [Zhang and Chen, 2018] and code embedding [Gu et al., 2018] techniques, but examples of cross-domain applications remain scarce. Robillard and Chhetri [2015], Robillard et al. [2017] first explore the task of suggesting reference API docs from source code using human feedback. Prior work also studies the relationship between comments and code entities [Iyer et al., 2018, Panthaplackel et al., 2020] using machine learning, but only within source code.

34 Maintainers of popular software projects often publish web-based developer docs, typically in markup languages like HTML or Markdown [Terrasa et al., 2018]. These documents contain natural language sentences, markup, and hyperlinks to other documents and source code artifacts. Both the document and link graph contain important semantic information. The

markup describes the text in relation to other entities in the document hierarchy [Yang et al., 2016], while the link graph describes relationships between related documents or source code entities. To compensate for the sparsity of hyperlinks between code and documentation, new techniques are likely required.

Unlike natural languages where polysemy is a common phenomenon [Ganea et al., 2016], most non-trivial tokens in source code are unique, even in large corpora. While the frequency of out-of-vocabulary (OOV) tokens presents a significant challenge for language modeling, it is an auspicious property for code search, where two lexical matches almost always refer to a single entity. Suppose we are given the string `AbstractSingletonFactoryBean`. We observe it has the following properties:

1. The string is camel-case, indicating it refers to an entity in a camel-case language.
2. The string contains the substring `Bean`, a common token in the Java language.
3. The string begins with a capital letter, indicating it refers to a class or interface.

Developers often use a tool called `grep` to locate files, which accepts queries written in the regular expression (regex) language, a domain specific language for string parsing and text search. Skilled `grep` users are able to rapidly construct a regex which retrieves the target entity with high probability whilst omitting irrelevant results. Assuming the entity exists on our filesystem, we can simply execute the following command to locate it:

```
$ grep -rE --include *.java "(class|interface) AbstractSingletonFactoryBean" .
```

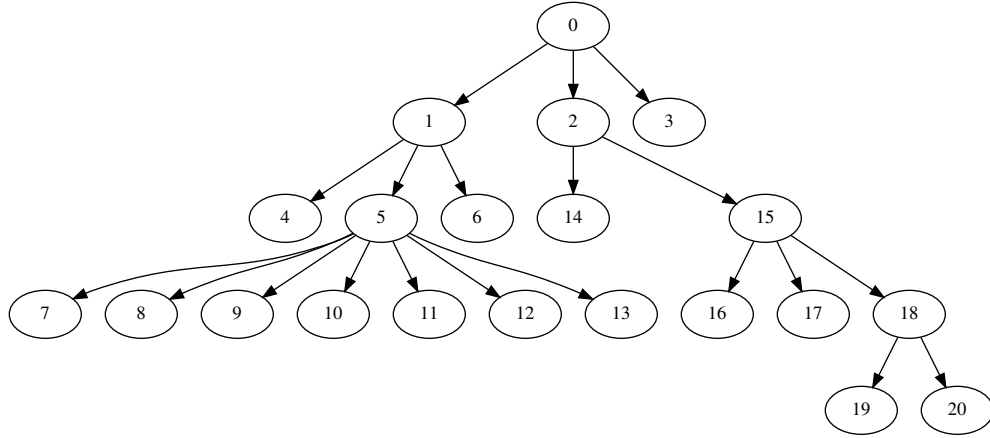
We hypothesize there exists a short regex which retrieves any named entity (assuming it exists) in a naturally occurring corpus of software artifacts. Given a named entity and its surrounding context, our goal is to synthesize a regex which selects only the link target, and as few other artifacts from the corpus as possible.

2 Dataset

Java is a statically-typed language with a high volume of API documentation. Offering a variety of tools for parsing source code [Parr, 2013, Hosseini and Brusilovsky, 2013, Kovalenko et al., 2019] and natural language [Manning et al., 2014, Grella and Cangialosi, 2018], it serves as a convenient language for both analysis and implementation. Our dataset consists of Java repositories on GitHub, and their accompanying documentation. All projects in our dataset have a collection of source code files and natural language documents.

We construct two datasets consisting of naturally-occurring links in developer docs, and a surrogate set of links constructed by matching lexical tokens in developer docs and source code. Our goal is recovery of ground truth links in the test set and surrogate links in the lexical matching graph. We evaluate our approach on both D2D and C2C link retrieval, as well as precision and recall on the surrogate link relations.

Our data consists of two complementary datasets: abstract syntax trees (ASTs) collected from Java source code and developer documentation. We use the `astminer` [Kovalenko et al., 2019] library to parse Java code, `jsoup` [Hedley, 2009] to parse HTML and Stanford’s `CoreNLP` [Manning et al., 2014] library to parse dependency graphs from developer docs. Consider the following AST, parsed from API documentation in the Eclipse Collections Java project:



78 The AST depicted above was generated by parsing the following code snippet:

```
public void lastKey_throws() {
    new ImmutableTreeMap<>(new TreeSortedMap<>()).lastKey();
}
```

79

80 Now consider the following dependency graph, taken from a Javadoc in the same project:

“The advantages to using this container over a `Map<K, Collection<V>>` is that all of the handling of the value collection can be done automatically. It also allows implementations to further specialize in how duplicate values will be handled. Value collections with list semantics would allow duplicate values for a key, while those implementing set semantics would not.”

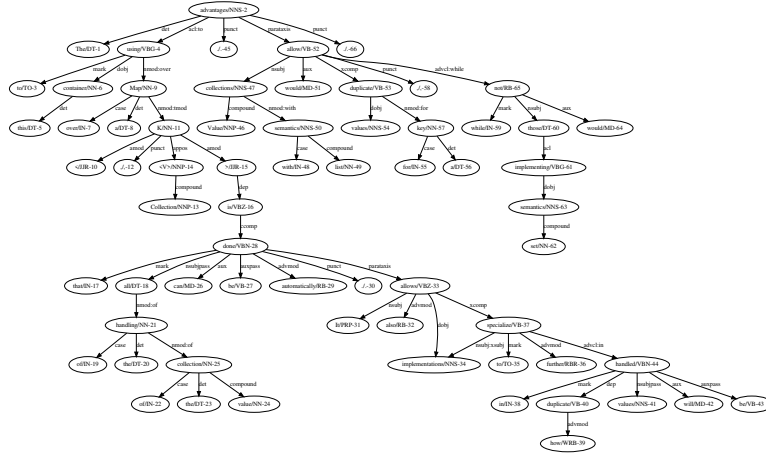


Figure 1: Dependency graph parsed from a Javadoc comment, shown above.

81 Our goal is to connect these two graphs using a common representation for source code and
82 natural language. Absent any explicit `@link` or `@see` annotations, in order to relate these
83 two graphs, we must somehow infer the shared semantic entities. We describe the problem
84 in the following section.

85 3 Problem

86 Suppose we have a set of hyperlinks $H = \{h_0, \dots, h_m\}$ and documents $D = \{d_0, \dots, d_n\}$.
87 Each document contains a sequence of tokens $T = \{t_0, \dots, t_p\}$. Each hyperlink h_i is a tuple

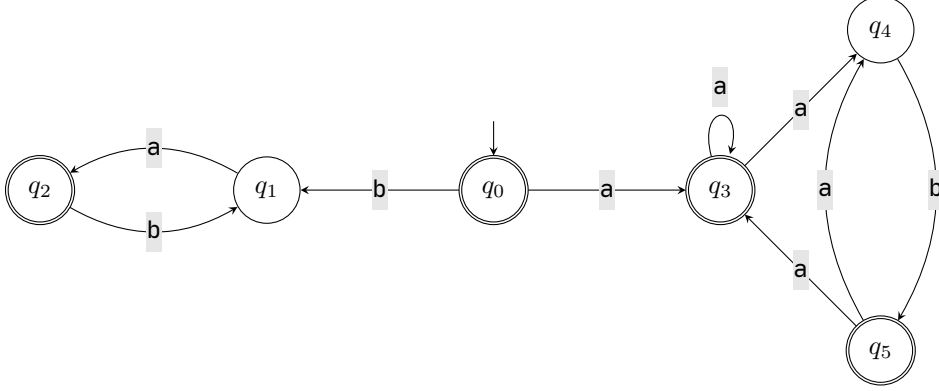


Figure 2: NFA corresponding to the regex $(a(ab)^*)^*(ba)^*$, where $q_{\{0,2,3,5\}} \in F$.

88 $\langle t_{anchor}, d_{source}, d_{target} \rangle$. We want a function $\mathcal{G} : (t_{anchor}, d_{source}; \theta) \mapsto \mathcal{R}$, parameterized
 89 by θ , taking a token and its parent document, which produces a regex $\mathcal{R} : \mathcal{D} \rightarrow \mathcal{B}$. This
 90 function tells us whether to accept or reject a given document \mathcal{D} . We seek θ minimizing
 91 $\sum_{d \in \mathcal{D}} \mathcal{L}_{test}(\mathcal{G}(t_{anchor}, d_{source}; \theta), d_{target})$ for all links in our test set.

92 4 Method

93 Let $\Sigma := \mathbf{A} \mid \mathbf{a} \mid \dots \mid \mathbf{Z} \mid \mathbf{z} \mid \mathbf{0} \mid \mathbf{1} \mid \dots \mid \mathbf{9} \mid \mid \mid \mathbf{\wedge}$. Our language $J_{<}$ has the following productions:

$$\langle exp \rangle := \langle exp \rangle \cdot \langle exp \rangle \mid \langle exp \rangle \mid \langle exp \rangle \mid \Sigma \mid \langle exp \rangle^* \mid !\langle exp \rangle \mid \cdot \quad (1)$$

94 Where ‘ \cdot ’ indicates concatenation. $J_{<}$ is a regular language, reducible to a non-deterministic
 95 finite automaton (NFA) using the Glushkov [1961] algorithm as shown in Figure 2 (inde-
 96 pendently discovered by McNaughton-Yamada-Thompson). NFA are reducible to both de-
 97 terministic finite automata (DFA) using the powerset construction [Rabin and Scott, 1959]
 98 and regular expressions using Arden’s Lemma [Arden, 1961]. Regular expressions can also
 99 be converted directly to DFA as described by Brzozowski [1964] and Berry and Sethi [1986].

100 Formally, an NFA is a 5-tuple $\langle Q, \Sigma, \Delta, q_0, F \rangle$, where Q is a finite set of states, Σ is the
 101 alphabet, $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$ is the transition function, $q_0 \in Q$ is the initial
 102 state and $F \subseteq Q$ are the terminal states. An NFA can be represented as a directed graph
 103 whose adjacency matrix is defined by the transition function, with edge labels representing
 104 symbols from the alphabet and binary node labels indicating whether the node is a terminal
 105 or nonterminal state.

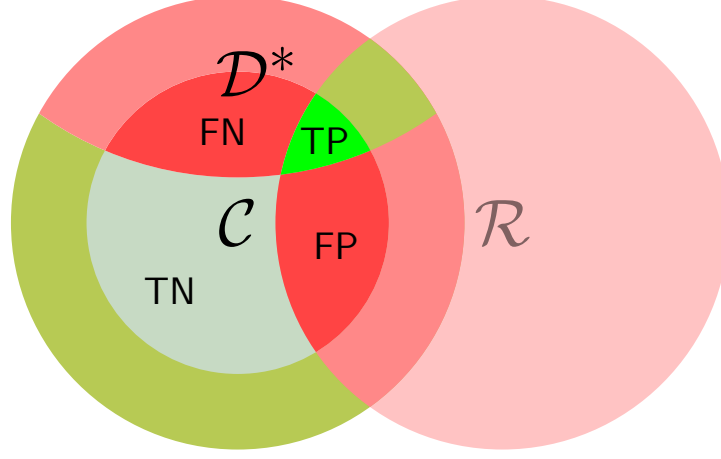
106 We pose the problem as a few-shot generative modeling task, where the input is a node
 107 embedding consisting of the query text and local graph context, and the output is a regular
 108 expression retrieving the link target with high precision.

109 Let \mathcal{D} be a document graph, constructed by semantically parsing the document’s contents,
 110 and neighboring documents from the link graph. Let \mathcal{T} be a code token, corresponding to
 111 a node in the document graph \mathcal{D} .

112 Our synthesizer is trained on a node embedding from the local graph context, a semantic
 113 graph parsed from the parent document and neighboring documents in the link graph. In-
 114 stead of directly performing link prediction, we train our generative model to output an
 115 NFA. We then compare precision and recall over the meta-test set.

116 Brzozowski [1964] defines the derivative of a language, \mathcal{Q} with respect to a string \mathbf{t} as follows:

$$\frac{\partial}{\partial \mathbf{t}} \mathcal{Q} = \{s \mid \mathbf{t}s \in \mathcal{Q}\} \quad (2)$$



117 As noted by Brzozowski, if we interpret the operators ‘ $|$ ’ and ‘ \cdot ’ from Equation 1 as ‘ $+$ ’
 118 and ‘ \times ’ respectively, we recover the standard rules from differential calculus:

$$\frac{\partial \mathbf{t}}{\partial \mathbf{t}} = \epsilon \quad (3)$$

$$\frac{\partial \epsilon}{\partial \mathbf{t}} = \emptyset \quad (4)$$

$$\frac{\partial Q^*}{\partial \mathbf{t}} = \frac{\partial Q}{\partial \mathbf{t}} Q^* \quad (5)$$

$$\frac{\partial \neg Q}{\partial \mathbf{t}} = \neg \frac{\partial Q}{\partial \mathbf{t}} \quad (6)$$

$$\frac{\partial Q \cdot S}{\partial \mathbf{t}} = \frac{\partial Q}{\partial \mathbf{t}} \cdot S \cup \frac{\partial S}{\partial \mathbf{t}} \cdot Q \quad (7)$$

123 Given some regex corresponding to a regular language \mathcal{R}^1 , this tells us how symbolic changes
 124 to the regex will change the language it recognizes. Suppose we have a corpus \mathcal{C} and a set
 125 of target documents $\mathcal{D}^* \subseteq \mathcal{C}$. We define a loss, $\mathcal{L}_{\mathcal{R}} : \langle \mathcal{R}, \mathcal{C}, \mathcal{D}^* \rangle \mapsto \mathbb{R}$ as follows:

$$\mathcal{L}(\mathcal{R}, \mathcal{C}, \mathcal{D}^*) = \mathbb{E}_{\mathcal{D}^* \sim \mathcal{C}} \frac{\overbrace{|\mathcal{R} \cap \mathcal{C}|}^P + \overbrace{|\mathcal{C} \setminus \mathcal{R}|}^N}{\underbrace{|\mathcal{R} \cap \mathcal{D}^*|}_{TP} + \underbrace{|\mathcal{C} \setminus \mathcal{D}^* \setminus \mathcal{R}|}_{TN}} \quad (8)$$

126 Given some set of strings we want to recognize, this ratio tells us how many documents from
 127 the corpus \mathcal{C} does the regex accept $|\mathcal{R} \cap \mathcal{C}|$ and reject $|\mathcal{C} \setminus \mathcal{R}|$, over how many documents it
 128 should accept and reject. We need this ratio to be as low as possible for effective retrieval.

129 Suppose we have a function $\mathcal{G}_{\theta} : \mathbb{R}^n \times \mathbb{R}^{|\theta|} \rightarrow \mathcal{R}$, which takes a contextual entity embedding
 130 \mathbb{R}^n , a set of parameters θ , and returns a regular expression \mathcal{R} . Our goal is to minimize
 131 $\mathcal{L}(\mathcal{G}_{\theta}, \mathbf{X}, \mathbf{Y})$, where \mathbf{X} is a set of unlabeled context embeddings and \mathbf{Y} is the ground truth
 132 link target. To compute $\nabla_{\theta} \mathcal{L}(\mathcal{G})$, we need $\nabla_{\Sigma} \mathcal{R}$, the vector of partial derivatives of \mathcal{R} with
 133 respect to every symbol in the alphabet, Σ . Brzozowski [1964] shows us how to backpro-
 134 propagate loss through \mathcal{R} , into parameter space $\mathbb{R}^{|\theta|}$.

¹Hereafter, we use \mathcal{R} to denote the regex and the language it recognizes interchangeably.

5 Preliminary Results and Discussion

We compare precision on our benchmark using top-K rank retrieval with respect to various baselines. For instance, to compute the count-search baseline, we search our corpus for the hyperlink anchor text, and rank the resulting documents by frequency of the anchor text. Results are shown in Figure 3.

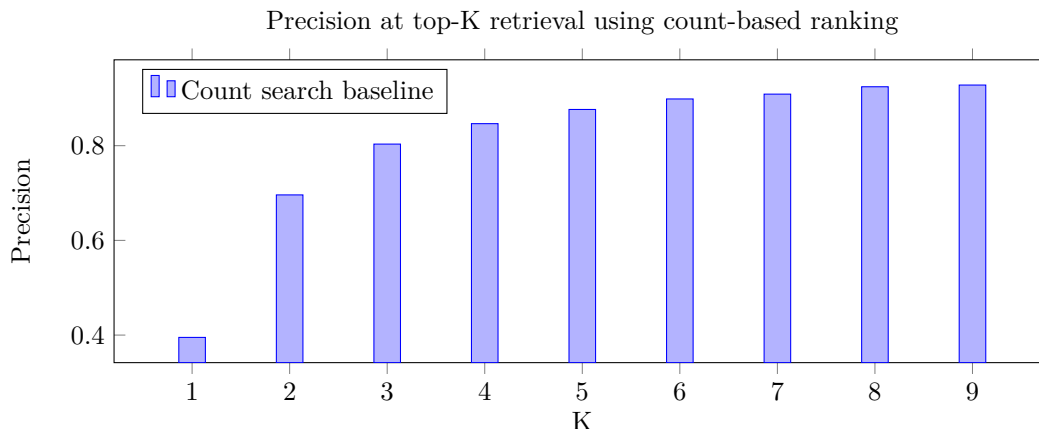


Figure 3: Results for count-based lexical matching baseline.

Preliminary results indicate 40% of all links in our dataset point to the document in which the anchor text occurs most frequently in the corpus, and 93% of all links refer to documents where the ground truth link occurs in the top-10 results ranked by frequency of the anchor text. In future work, we will compare performance against a graph neural network using a word-embedding approach, trained to minimize cosine distance between the source document and target document’s context, using a set of candidate links returned by the count-search baseline. Finally, we will compare our approach against the count-search baseline. To do so, we will need to train a synthesizer using our loss function over the regex. Furthermore, we will need to construct the semantic graph embedding over the surrounding context.

References

- Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017. URL <https://arxiv.org/pdf/1711.00740.pdf>.
- Dean N Arden. Delayed-logic and finite-state machines. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pages 133–151. IEEE, 1961.
- Gerard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theoretical computer science*, 48:117–126, 1986. URL <https://core.ac.uk/reader/82374120>.
- Avishek Joey Bose, Ankit Jain, Piero Molino, and William L Hamilton. Meta-graph: Few shot link prediction via meta learning. *arXiv preprint arXiv:1912.09867*, 2019. URL <https://arxiv.org/pdf/1912.09867.pdf>.
- Janusz A Brzozowski. Derivatives of regular expressions. *Journal of the ACM (JACM)*, 11(4):481–494, 1964. URL http://maveric.uwaterloo.ca/reports/1964_JACM_Brzozowski.pdf.
- Octavian-Eugen Ganea, Marina Ganea, Aurelien Lucchi, Carsten Eickhoff, and Thomas Hofmann. Probabilistic bag-of-hyperlinks model for entity linking. In *Proceedings of the 25th International Conference on World Wide Web*, pages 927–938, 2016. URL <https://arxiv.org/pdf/1509.02301.pdf>.
- Victor Mikhaylovich Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1, 1961.
- Matteo Grella and Simone Cangialosi. Non-projective dependency parsing via latent heads representation LHR. *arXiv preprint arXiv:1802.02116*, 2018. URL <https://arxiv.org/pdf/1802.02116.pdf>.
- Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642, 2016. URL <https://arxiv.org/pdf/1605.08535.pdf>.
- Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE, 2018. URL <https://guxd.github.io/papers/deepcs.pdf>.
- Jonathan Hedley. jsoup: Java HTML parser. 2009. URL <https://jsoup.org>.
- Jordan Henkel, Shuvendu K Lahiri, Ben Liblit, and Thomas Reps. Code vectors: Understanding programs through embedded abstracted symbolic traces. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 163–174, 2018. URL <https://arxiv.org/pdf/1803.06686.pdf>.
- Roya Hosseini and Peter Brusilovsky. JavaParser: A fine-grain concept indexing tool for Java problems. In *CEUR Workshop Proceedings*, volume 1009, pages 60–63. University of Pittsburgh, 2013.
- Srinivasan Iyer, Ioannis Konostas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. *arXiv preprint arXiv:1808.09588*, 2018. URL <https://arxiv.org/pdf/1808.09588.pdf>.
- 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. URL <https://github.com/JetBrains-Research/astminer>.

197 Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and
198 Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint*
199 *arXiv:1603.01360*, 2016. URL <https://arxiv.org/pdf/1603.01360.pdf>.

200 Jian Li, Yue Wang, Michael R Lyu, and Irwin King. Code completion with neural attention
201 and pointer networks. *arXiv preprint arXiv:1711.09573*, 2017. URL <https://www.ijcai.org/Proceedings/2018/0578.pdf>.

203 Bohong Liu, Tao Wang, Xunhui Zhang, Qiang Fan, Gang Yin, and Jinsheng Deng. A
204 neural-network based code summarization approach by using source code and its call
205 dependencies. In *Proceedings of the 11th Asia-Pacific Symposium on Internetwork, Inter-*
206 *network '19*, New York, NY, USA, 2019. Association for Computing Machinery. ISBN
207 9781450377010. doi: 10.1145/3361242.3362774. URL [https://doi.org/10.1145/](https://doi.org/10.1145/3361242.3362774)
208 [3361242.3362774](https://doi.org/10.1145/3361242.3362774).

209 Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy.
210 Stack-pointer networks for dependency parsing. *arXiv preprint arXiv:1805.01087*, 2018.
211 URL <https://arxiv.org/pdf/1805.01087.pdf>.

212 Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard,
213 and David McClosky. The stanford coreNLP natural language processing toolkit. In
214 *Proceedings of 52nd annual meeting of the association for computational linguistics: sys-*
215 *tem demonstrations*, pages 55–60, 2014. URL [https://nlp.stanford.edu/pubs/](https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf)
216 [StanfordCoreNlp2014.pdf](https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf).

217 Sheena Panthaplackel, Milos Gligoric, Raymond J. Mooney, and Junyi Jessy Li. Associating
218 natural language comment and source code entities. In *AAAI*, 2020. URL [https://](https://arxiv.org/pdf/1912.06728.pdf)
219 arxiv.org/pdf/1912.06728.pdf.

220 Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

221 Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM*
222 *journal of research and development*, 3(2):114–125, 1959.

223 Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark
224 Steedman, and Mirella Lapata. Transforming dependency structures to logical forms
225 for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:
226 127–140, 2016. URL [https://www.mitpressjournals.org/doi/pdf/10.1162/](https://www.mitpressjournals.org/doi/pdf/10.1162/tacL_a_00088)
227 [tacL_a_00088](https://www.mitpressjournals.org/doi/pdf/10.1162/tacL_a_00088).

228 Martin P Robillard and Yam B Chhetri. Recommending reference API documentation. *Em-*
229 *pirical Software Engineering*, 20(6):1558–1586, 2015. URL [https://www.cs.mcgill.](https://www.cs.mcgill.ca/~martin/papers/cr2014a.pdf)
230 [ca/~martin/papers/cr2014a.pdf](https://www.cs.mcgill.ca/~martin/papers/cr2014a.pdf).

231 Martin P Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Cha-
232 parro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-
233 Vázquez, et al. On-demand developer documentation. In *2017 IEEE International Con-*
234 *ference on Software Maintenance and Evolution (ICSME)*, pages 479–483. IEEE, 2017.

235 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and
236 Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning*
237 *Research*, 12(Sep):2539–2561, 2011.

238 Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning
239 loop invariants for program verification. In *Advances in Neural Information Pro-*
240 *cessing Systems*, pages 7751–7762, 2018. URL [https://papers.nips.cc/paper/](https://papers.nips.cc/paper/8001-learning-loop-invariants-for-program-verification.pdf)
241 [8001-learning-loop-invariants-for-program-verification.pdf](https://papers.nips.cc/paper/8001-learning-loop-invariants-for-program-verification.pdf).

242 Alexandre Terrasa, Jean Privat, and Guy Tremblay. Using natural language processing for
243 documentation assist. In *Workshops at the Thirty-Second AAAI Conference on Artificial*
244 *Intelligence*, 2018.

- 245 Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence
246 for sets. *arXiv preprint arXiv:1511.06391*, 2015a. URL [https://arxiv.org/pdf/](https://arxiv.org/pdf/1511.06391.pdf)
247 [1511.06391.pdf](https://arxiv.org/pdf/1511.06391.pdf).
- 248 Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in*
249 *neural information processing systems*, pages 2692–2700, 2015b. URL [https://arxiv.](https://arxiv.org/pdf/1506.03134.pdf)
250 [org/pdf/1506.03134.pdf](https://arxiv.org/pdf/1506.03134.pdf).
- 251 Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchi-
252 cal attention networks for document classification. In *Proceedings of the 2016 conference*
253 *of the North American chapter of the association for computational linguistics: human*
254 *language technologies*, pages 1480–1489, 2016. URL [https://www.cs.cmu.edu/~.](https://www.cs.cmu.edu/~hovy/papers/16HLT-hierarchical-attention-networks.pdf)
255 [/hovy/papers/16HLT-hierarchical-attention-networks.pdf](https://www.cs.cmu.edu/~hovy/papers/16HLT-hierarchical-attention-networks.pdf).
- 256 Muhan Zhang and Yixin Chen. Link prediction based on graph neu-
257 ral networks. In *Advances in Neural Information Processing Systems*,
258 pages 5165–5175, 2018. URL [https://papers.nips.cc/paper/](https://papers.nips.cc/paper/7763-link-prediction-based-on-graph-neural-networks.pdf)
259 [7763-link-prediction-based-on-graph-neural-networks.pdf](https://papers.nips.cc/paper/7763-link-prediction-based-on-graph-neural-networks.pdf).