Tracking the Flow of Ideas through the Programming Languages Literature

Michael Greenberg[1], Kathleen Fisher[2], and David Walker[1]

1 Princeton University
2 Tufts University

Abstract

How has PLDI changed over the last 20 years? Does NSF funding influence the themes that appear in OOPSLA? In what topics does POPL lead and ICFP follow? In what topics does ICFP lead and POPL follow? How have the ideas in O'Hearn's classic paper on separation logic spread across the programming languages world? Does my program committee cover the range of submissions we'd expect for this conference? Who was doing research like me back in the 80s? If we had better tools for analyzing the programming languages literature, these are some of the questions we might be able to answer. In this submission, we explain the role that *topic modeling*, a branch of machine learning, might play in helping the programming languages community better understand our literature. We explain what topic modeling is, how we have applied it to the PL literature, some preliminary results we have obtained, and a proposal for building web tools that will help PL researchers better understand our history.

1 Introduction

Over the past half-century, the programming languages community has developed a foundational new science, the science of software, that now governs our world. One of the legacies of this great work is a vast literature — a literature so large no one person or institution can know much more than the tiniest fraction of a fraction of what is out there. Unfortunately, our normal interaction with this literature is tragically limited. At our best, we read a small selection of papers in our area, look at their related work sections and follow the citations backwards a level or two. Occasionally, a friend will alert us to a neat paper that she has read. That is about it.

If we had better tools for analyzing the programming languages literature, there are so many more interesting questions we could ask and answer:

- How has PLDI changed over the last 20 years?
- How did O'Hearn's classic paper on separation logic influence POPL over the years?
- Does NSF funding influence the topics appearing in top conferences?
- Where did the ideas in my paper *really* originate?

In addition to answering questions simply for the sake of curiosity, tools better able to analyze the programming languages literature might be of help to the chairs of programming languages conferences and the editors of programming languages journals. Program chairs would like to know that their PCs cover the appropriate topics effectively. Editors would like reviewers with sufficient expertise, but, perhaps, different perspectives.

Many of these questions are deep semantic questions pertaining to the true flow of ideas across the programming languages literature and even beyond. We cannot answer these questions with perfect accuracy or certainty, but we may be able to approximate them. In order to investigate this possibility, we have begun to explore a specific group of analysis

SNAPL.
Editor: Shriram Krishnamurthi; pp. 1–9
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

techniques developed in the machine learning community, known as *topic models* [3]. Generally speaking, topic models are probabilistic descriptions of the words that appear in a document or a collection of documents. Such models provide insights in to the themes that pervade a corpus and the relative weights of those themes. One can use these models to compare individual papers or groups of papers, and one can track the ebb and flow of themes over time.

Such models have recently been used to analyze articles from the journal *Science*, the *Yale Law Review*, and other archives of documents [3]. They have also been used to try to understand the links betweeen academic literature and funding [4]. We propose to apply these models and techniques to the programming languages literature and to ask questions of interest to the programming languages community. Moreover, we plan to build a tool that will allow other researchers to browse the programming languages literature by topic, find related papers, and visualize the evoluation of the topics studied in PL over time.

As suggested in the SNAPL call-for-papers, we present our plan for doing this in the form of a proposal. At the moment of submission, we have gathered enough data and done enough initial investigation of topic models to convince ourselves that topic models have the potential to provide, at the very least, an *interesting* lens on the programming languages literature if not a practical one. We believe the curious amongst programming language researchers will be intrigued.

So far, we have a corpus of 4,355 abstracts from four of the top programming languages conferences: ICFP, OOPSLA, PLDI and POPL. We are able to generate models for the programming languages community as a whole and for the conferences individually and compare them. We are also able to analyze individual documents and compare papers based on their models. However, we have only run a small number of tests so far; there is much more work to be done. What we have done so far is available on github.[1] We will continue to update our repository as we make progress. We are confident that by the final paper submission deadline, we will have more results to explain. By the SNAPL conference itself in May, we believe we will be able to present an unusual, new perspective on the programming languages literature to the audience. Michael Greenberg may be the most junior amongst the authors, but he gives superb talks and will be presenting the results. We welcome feedback from the program committee, especially suggestions concerning specific questions we should ask and creative ways we might apply the models we are building.

In the remaining sections of this submission, we will describe the specific kinds of topic models that we are currently using—the Latent Dirichlet Allocation models—and the way we have curated our data (Section 2). In Section 3, we describe some of our preliminary results. In Section 4, we describe ideas for future experiements. We conclude in Section 6.

2 Topic models

To explain the ideas behind the simplest topic model, the Latent Dirichlet Allocation (LDA), we adapt the explanation given by David Blei [3] to a programming-languages setting. In this model, each document is considered to be a mixture of some number of topics. For example, consider the paper "Gradual Typing for First-Class Classses" [6] shown in Figure 1, which describes a type system for gradually converting dynamically-typed object-oriented programs into statically typed programs in a companion language. We manually highlighted various words occuring in the abstract: yellow for words related to components, green for words

[1] See `https://github.com/mgree/sigplan/tree/master/lda`.

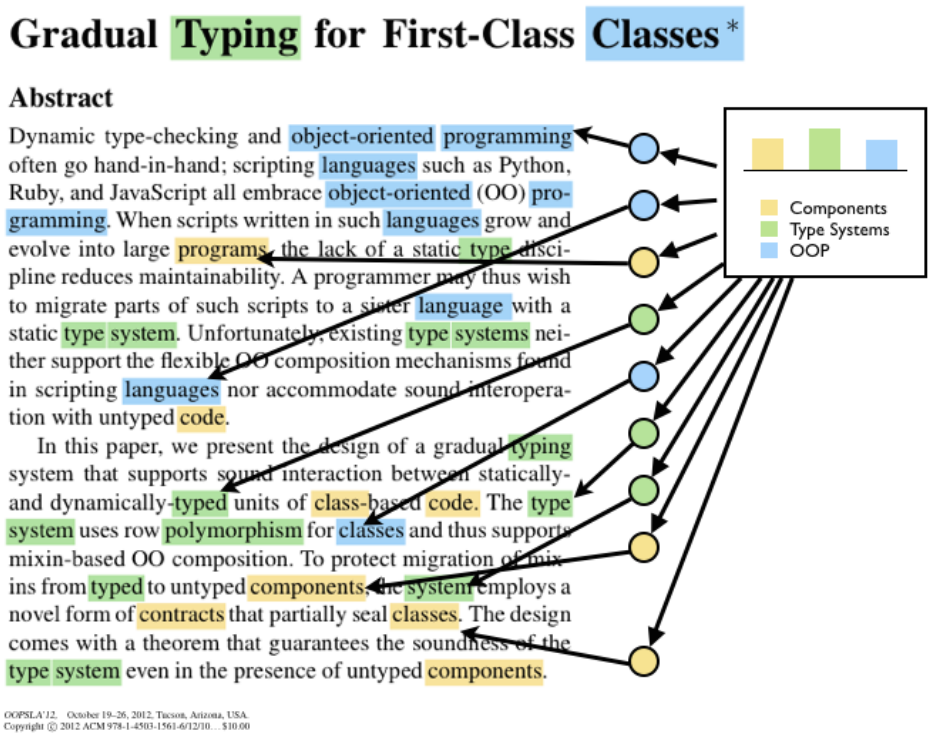

Figure 1 LDA topic modeling is based on the generative process illustrated here. It works over a fixed collection of topics assumed to be shared by all documents in the corpus (top right). First, we select a probabilty distribution over the topics (histogram). Then, to produce each word in the document, we first choose a topic from the distribution (the colored circles) and then we choose the actual word from the word distribution associated with the topic (the highlighted words).

related type systems, and blue for words related to Object-Oriented Programming (OOP). If we highlighted the entire paper in this way, it would be apparent that the document mixes concepts from these three topics: components, type systems, and OOP.

LDA models this intuition that a document is a mixture of different topics. In the LDA framework, each "topic" is represented as a probability distribution over a fixed collection of words. The probability associated with each word indicates how strongly the word belongs to the topic. For example, words like 'type' and 'polymorphism' have high probabilities in the "Type System" topic, while 'component' and 'contract' have high probabilities in the "Components" topic. In contrast, words like 'cpu' and 'register' have low probabilities in both topics. Similarly, a document is modelled as a distribution over such topics.

In the paper in Figure 1, the probability distribution over topics would assign weights to the topics "Components," "Type Systems," and "OOP." The chart at the right of the figure shows this distribution, with "Type Systems" carrying slightly greater weight than either "Components" or "OOP." The words in the paper are drawn from the probability distributions associated with these topics. To put it another way, if we wanted to generate this document, or one similar to it, we would pick each word for the document independently according to the following *generative process*: (1) pick a topic (according to the topic distribution) and then (2) pick a word from that topic (according to its distribution over words). Other papers in the SIGPLAN corpus would exhibit a different mix of topics. A key point is that all the documents in an LDA framework share the same collection of topics,

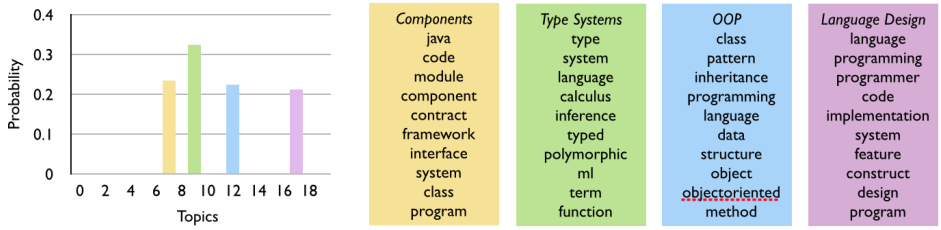

Figure 2 LDA inference for 20 topics run on a corpus of 4355 titles and abstracts from SIGPLAN conferences and applied to the example paper "Gradual Typing." The probability distribution at the left shows the inferred topic mixture for the paper. The boxes at the right show the 10 most probable words for the four topics present in the paper.

just mixed in different proportions (including nearly zero).

Of course the purpose of topic modeling is not to generate random documents but rather to automatically discover the topics present in a corpus. We *observe* the documents, but the topic structure is *hidden*, including the set of topics, the topic distribution for each document, and the per-document, per-word topic assignment. The challenge of this area of research is to reconstruct the hidden topic structure from an observed corpus, essentially running the generative process backward, attempting to answer the question: What fully-defined topic model most likely explains the given corpus?

We ran David Blei's LDA-C[2] over a corpus of 4355 titles and abstracts from ICFP, OOPSLA, PLDI, and POPL papers that are available in the ACM Digital Library, having set the number of topics at 20.[3] This process produced a topic model M, judged to be the most likely 20-topic model to explain the SIGPLAN corpus. We then applied M to the "Gradual Typing" paper to impute the topic distribution that best explains the content of that paper. The results appear in Figure 2. This histogram on the left shows that although M could have assigned any probability distribution to the twenty topics, it selected only four. The right side of the figure shows the most likely words associated with those four topics. Looking at these words, we can see that three of the topics correspond to the ones we higlighted in Figure 1: Components, Type Systems, and OOP. LDA identified a fourth topic, whose words suggest it might be called "Language Design." Note that LDA does not produce the titles for the topics; we manually came up with names for them based on the most likely words. Automatically inferring such titles is an open research problem.

2.1 Our methods

As mentioned above, we used LDA-C to build models for four SIGPLAN conferences. A typical run of LDA-C with 20 topics takes about 2 minutes on a four-core 2.8Ghz Intel i7 with 16GB of RAM and a solid-state hard drive; simultaneous runs with 200, 300, and 400 topics take just shy of two hours. Our code is available at `https://github.com/mgree/sigplan/tree/master/lda`. Here we give the context to understand this code: Where did we get the data, how did we prepare it, and how did we interpret the output?

The ACM Digital Library (DL) stores SIGPLAN research papers.[4] We scraped titles,

[2] `http://www.cs.princeton.edu/~blei/lda-c/`
[3] One of the limitations of LDA is that it will not help decide how many topics appear in a corpus. The user must decide in advance and seed the inference mechanism with the chosen number.
[4] `http://dl.acm.org/`; SIGPLAN conferences are posted at `http://dl.acm.org/sig.cfm?id=SP946`

author lists, and abstracts from every available year of ICFP (1996-2014), OOPSLA (1986-89,1991-2014), PLDI (1988-2014), and POPL (1973,1975-2015). The scraping is imperfect—it doesn't find every paper listed in the DL (see Section 2.1.1). From this scraped data, we considered the concatenation of a paper's title and its abstract as a document in our corpus; we tracked metadata for each document, recording its author list along with when and where it was published.

We parsed the scraped abstracts into a format LDA-C can read: a document file that represents each document as a "bag of words." Parsing forces us to decide which words are interesting and how to count those words. For example, the word 'we' appears many times in academic writing, but it isn't a meaningful topic word in the way that 'type' or 'flow' or 'register' are. So that we may focus on meaningful words, we drop all words on a *stopword* list from the document. We started with a standard stopword list for English[5], amending it with two lists of words. First, we added words which appeared in more than 25% of the documents in our corpus. We also added words that felt irrelevant, such as 'easily', 'sophisticated', 'interesting', and 'via'.

Once we've winnowed out the stopwords, we need to turn each text into a bag of words: a list pairing words with the number of times they occur. Are 'type' and 'types' different words? What about 'typing' and 'typical'? We counted words as the same when they reduced to the same uninflected word via NLTK's[6] WordNet [8] lemmatizer. In this case, 'type' and 'types' both reduce to the uninflected word 'type', but 'typing' and 'typical' are left alone.

2.1.1 Limitations and impediments

Problems with the data begin at the ACM. The ACM DL is missing data. The abstracts for ICFP, OOPSLA, PLDI, or POPL are missing for 1991. POPL is missing abstracts from 1996 through 1999. The ACM DL listings don't obviously differentiate hour-long invited keynote talks and the shorter talks in the sessions. We chose to include abstract-less keynotes as "documents"... but is that the right decision? Scraping the DL for abstracts is not completely straightforward. Some papers are clearly in the HTML but our scraper doesn't detect them. Ideally, we would have access to the database underlying the DL, with programmatic access to metadata and the *full text* PDFs of each paper. The full text contains vastly more data, much of it more technical and specialized, which could easily lead to superior results. We will investigate gaining access to more full text PDFs prior to SNAPL itself.

For the graphs shown here, we ran LDA-C with 20 topics. Figure 3 offers two histograms. On the left, we see that every document has non-minimal (0.03189) weight in *some* topic; on the right, we see that nearly one in five papers (869 out of 4,355) doesn't have a weight higher than 20 in any topic. Running with more topics reduces the number of indeterminate, topicless documents. However, more topics can also lead to over-fitting and difficulty in distinguishing the underlying concepts. For comparison, Blei's analysis of the journal Science used 100 topics; Analysis of the Yale Law Review used 20 topics. We have experimented with up to 400 topics, but elected to present 20 for simplicity here.

[5] `https://pypi.python.org/pypi/stop-words/2014.5.26`
[6] `http://www.nltk.org/`

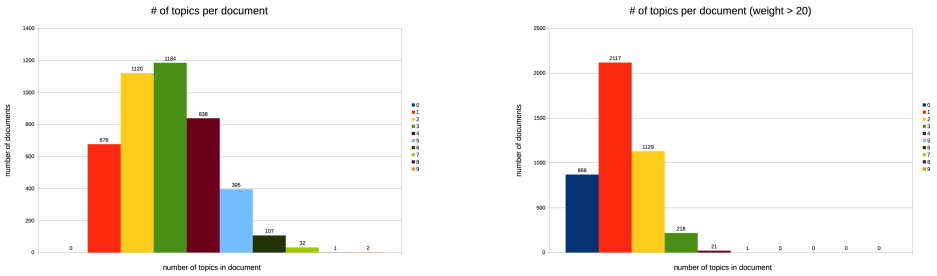

Figure 3 Number of topics per paper (left: normalized per topic, right: normalized per topic, minimum topic weight of 20

3 Preliminary results: Comparing Conferences

We have analyzed the programming languages literature in several different ways using our topic models. We have used them to search for similar papers, to analyze topic correlation via 2- and 3-dimensional scatter plots, and to study the evolution of topics over time. All of these experiments are quite preliminary and we have had varying levels of success. In this section, we present one of our more interesting results so far, an analysis and visualization of the evolution of topics over time in major programming languages conferences. You can browse a TMVE[7] visualization of our analysis at `http://www.cs.princeton.edu/~mg19/sigplan_lda20/`.

In this experiment, we separated papers in to collections according to conference (ICFP, OOPSLA, PLDI and POPL) and year (1973-2015). In each conference, in each year, we can apply our topic model to determine the topics and weight at which they appear. Figure 4 presents the results for our 20-topic model. The top chart represents research presented at POPL between 1973 and 2015. Each column represents a year; the length of a colored line represents the weight of the corresponding topic in the given year. The three smaller charts present data for ICFP, OOPSLA and PLDI respectively. When viewing these charts online, we recommend zooming in to inspect the details.

Analysis of these charts confirms many of our intuitions about the relative character of these conferences. For instance, only a few percent of POPL papers concern themselves with classic compiler optimizations (the dark blue at the bottom of each chart) or low-level compiler optimization (the dark green 4th from the bottom). At PLDI, on the other hand, compiler optimization has a robust presence. Interestingly though, over the past few years, research on optimization appears to be waning significantly at PLDI, especially in contrast to its hayday in the mid-90s. Three of the topics that seem to be replacing compiler optimization at PLDI are the analysis of concurrent programs (the bright red), test generation (medium blue) and verification (the dull red). These topics had a limited presence at PLDI in the 90s but have become significant components of the program in the 00s. Such graphs suggest our approach has promise, but there is much more to be done.

4 Proposal

Before SNAPL meets in May, we plan to build tools for analyzing the programming languages literature and apply them to a variety of data sets. One end goal is to build a website offering

[7] https://code.google.com/p/tmve/

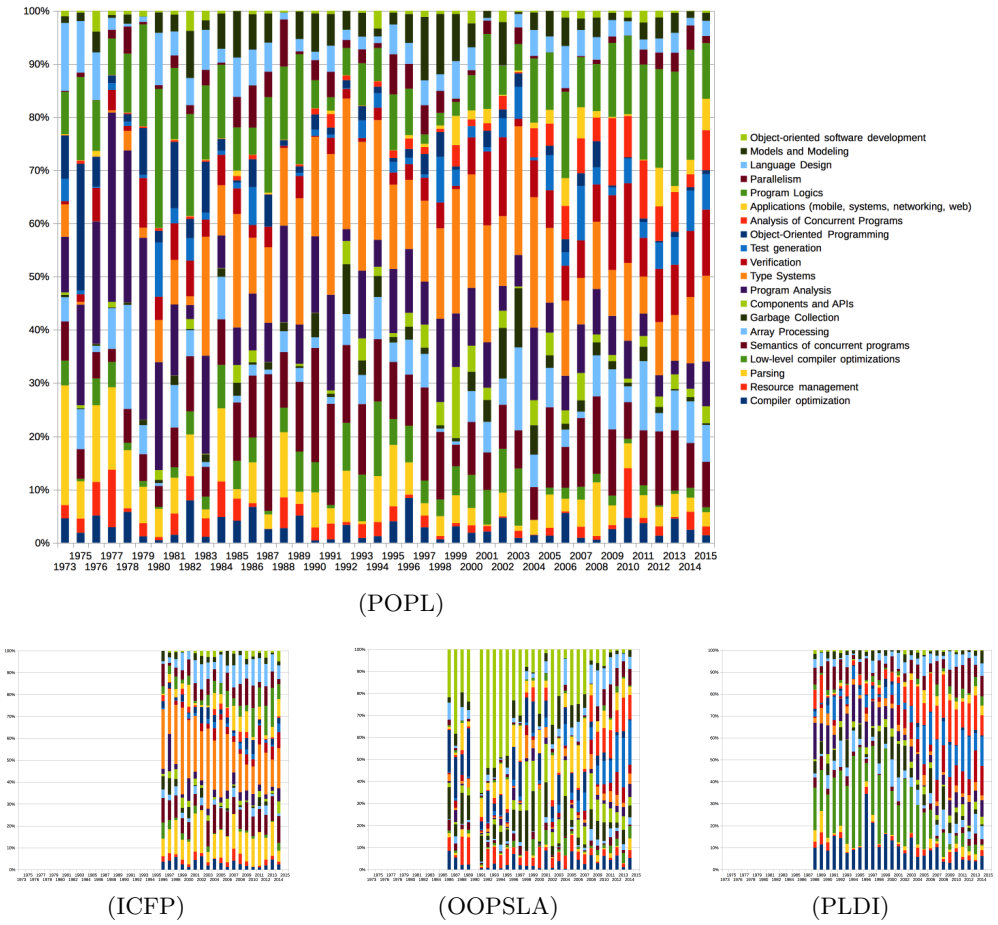

Figure 4 Topics over time of four programming languages conferences.

programming languages researchers access to the tools and/or analysis results we find are most interesting or effective. The backend source will be freely available, allowing other communities to reuse our work. TMVE [7] is a topic model visualization engine that offers much of the infrastructure we need for this work. The following paragraphs flesh out some of the possibilities.

Related work search. Topic models offer a similarity measure, allowing us to compare papers and researchers. When two documents (or sets of documents) are assigned topic vectors that are close in a geometric sense, we can infer that those two documents are similar. Such similarity measures can help authors search for related work, and program chairs or journal editors to search for appropriate reviewers. We also envision authors using our semantic search facility just for the sake of curiousity, to see what pops up.

Understanding the conferences. There are a number of ways we can paint a more detailed picture of our programming languages conferences. First, we can simply try running LDA with more topics. Second, we can investigate richer topic models. For instance, dynamic LDA [2] models the shift of topic content over time, and has been used for *lead/lag* analysis: When new topics appear, do they appear in one conference and spread to others? One very interesting study looked at the relationship between the topics funded and the reearch that appeared at conferences [4]. We could obtain the summaries of funded NSF

proposals (and possibly other institutions) to determine if we can identify relationships between funded topics and conference content.

Understanding the researchers. We can also use topic models to learn topics for researchers, not just documents. There are many ways to do this—from collating all of a researcher's oeuvre to using time-aware techniques, like dynamic LDA [2]. Given a topic model for each researcher, it is possible to construct many tools:

– to assist with assembling a PC or ERC, we can find groups of authors who haven't recently served, with aggregated topic weights sufficiently covering a given set of topics;
– to chose which papers to review, PC or ERC members can find papers with topics similar to their own;
– to identify papers with insufficient reviewer expertise, a PC chair could measure how well the reviewers' topics cover the topics of the paper under review.

We may also be able to use models of PL researchers to answer questions about how researchers change over time. Do they stay in one topic, or work in many topics? Are there researchers who lead their conferences? Can we apply social network models to co-authorship graphs to understand how people share interest in topics? Can this help in assembling workshops or Dagstuhl-like seminars?

5 Related work

There is much work in the machine learning community aimed at understanding large collections of documents. David Blei's web page [1] on topic modeling contains pointers to several survey papers, talks and software packages. Our interests are in using this software to study the flow of ideas across the programming languages literature, something which has not be done before using similar methods.

There are various other ways one might analyze the programming languages literature. For instance, in a recent article for CACM [5], Singh *et. al.* studied research in hardware architecture between 1997 and 2011 on the basis of occurrences of keywords. Measuring keyword occurrences has the benefit of simplicity, but it does not identify collections of words—themes, as Blei would call them—that find themselves colocated in documents. These themes may help us characterize papers and researchers succinctly and may suggest interesting measures to compare them. Moreover, the LDA algorithm identifies these themes for us without anyone having to specify them.

6 Conclusion

Topic models offer the potential to give us a new lens on the programming languages literature. We already have some interesting results analyzing the content of our major conferences through this lens. We hope to be able to build some useful tools for analyzing the literature, but at the very least, we believe should have some interesting results to present at SNAPL.

References

1 David Blei. Topic modelling. `http://www.cs.princeton.edu/~blei/topicmodeling.html`, 2015.
2 David Blei and John Lafferty. Dynamic topic models. In *International Conference on Machine Learning*, 2006.
3 David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

4 Xiaolin Shi, Ramesh Nallapati, Jure Leskovec, Dan McFarland, and Dan Jurafsky. Who leads whom: Topical lead-lag analysis across corpora. In *NIPS Workshop*, 2010.

5 Virender Singh, Alicia Perdigones, and Fernando R. Mazarrón José Luis Garcia, Ignacio Cañas-Guerroro. Analyzing worldwide research in hardware architecture, 1997-2011. *CACM*, 58(1):76–85, 2015.

6 Asumu Takikawa, T. Stephen Strickland, Christos Dimoulas, Sam Tobin-Hochstadt, and Matthias Felleisen. Gradual typing for first-class classes. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pages 793–810, New York, NY, USA, 2012. ACM.

7 TMVE: Topic model visualization engine. `https://code.google.com/p/tmve/`, 2015.

8 Princeton University. Wordnet. `http://wordnet.princeton.edu`, 2010.