#7 A Pragmatic Approach to Syntax Repair

Email notification
Select to receive email on updates to reviews and comments.

▼ PC conflicts
None

Accepted

Submission (345kB) 22 Jul 2023 7:18:37am EDT · 45b54beb

▼ Abstract

Programming languages share a social and formal heritage. These families were historically divided, but share deep roots, and we argue their destined matrimony heralds important consequences for language design and generative language modeling. In our work, we develop a sociotechnical frame- work for understanding the dynamics of programming and argue it captures many of the social and formal properties of language acquisition and evolution.

▶ Author

B. Considine [details]

	OveMer	RevExp
Review #7A	4	3
Review #7B	4	2

You are an **author** of this submission.

Edit submission

Reviews in plain text

Review #7A

Overall merit
4. Accept

Reviewer expertise
3. Knowledgeable

Paper summary

The paper proposes a modification of the Variant [8] parsing algorithm for automatic syntax error repair. It describes Tinyparse, a tool based on these ideas, and then proposes to evaluate the tool by measuring throughput by measuring (1) he rate at which repairs are accepted by the parser, and precision, by measuring (2) the rate at which human repairs are matched (rather than t).

As test input for (1) it synthetically corrupt parseable single-line statements in Java and Kotlin source code by sampling edits uniformly at random from the Levenshtein ball and rejecting any edits accepted by the true parser. As input for (2) is uses a dataset of human syntax errors and repairs from Python snippets on StackOverflow [9]. It hypothesizes that Tinyparse will have better throughput and precision than Sweq2Parse, and Break-it-Fix.

Comments for authors

It seems to me that the approach proposed by the paper is similar to how current standard syntactic error recovery tools based on LALR- or LL-parsers work by searching for alternative inputs which are acceptable to the parser, and which are not that distnat from the current erroneous input. If so, then I wonder how Tinyparse would behave if compared with the syntax error recovery tools.

I enjoyed reading the paper, both for the contents and for the confident style (eg pragmatic approach to program repair takes a charitable view of the author: not as a clumsy fool to be scolded into conformity, but an earnest programmer trying to convey their intent).

Nevertheless, I am not clear whether Tinyparse has been developed, and the experiments described in the paper have been carried out, or are still future work? If the former, then I would love to hear the outome of these experiments,

Is the use of the matrix algorithm by [8] for syntax error repair known in the literature? If so, then where is it described? and if not, then perhaps Breandan should write a separate paper on that.

The paper introduces several intriguing ideas, which it then does not follow up. I believe that this may be because Breandan is still trying to distill where he next wants to go. However, these many intriguing ideas confused me, and perhaps the paper should be streamlined. I will be three examples below:

Example_1

> Programming languages share a social and formal heritage. but I do not see this theme revisited in the rest if the paper.

Example_1 in the introduction

> Language games are a pragmatic framework for model- ing the social dynamics of code. In one such game, two players, the driver and a navigator, collaborate across a shared workspace to construct a domain-specific language that solves a chosen problem. Initially, both parties have a limited understanding of their counterpart and to reach a common

> understanding, must exchange messages conveying, e.g., their intent, abilities and knowledge. This game comes to an end when the chosen problem is successfully solved, the clock eventually runs out, or the problem is deemed unsolvable and one player forfeits due to exhaustion.

But I cannot see the connection between section 5, with such computer games.

Example_3 On page 2, the paper introduces and discusses several criteria for such repair tools: soundness, completeness, efficiency, responsiveness, robustness, scalability, flexibility, extensibility. However later on it only talks about precision and throughput.

Example_4 It is interesting that the abstract and my summary of the paper tell a different story. It may be that I have misunderstood the paper, or it may be that Breandean changed his mis mind while writing the paper.

At times, when you speak of "constraints" and "specifications" whether you were planning to work on syntactic errors, or semantic erorrs, or even on not adgering to program's specification,

> as well-formed programs have many semantic constraints, is that not a feature of the language rather than the program? And who enforces these constraints?

> end-users should be able to extend the framework with their own custom grammars and side-constraints. What do you mean by "sideconstraints"? If it is semantic rules, or type checking doe it not go way beyond the remit of parsing?

> we take the intersection of the respective grammars' languages, is there more than one language we are interested in when reparing the program?

Can you give a URL for Tidyparse?

What does "Precision@{1, 5, 10, All}" mean?

Review #7B

Overall merit

4. Accept

Reviewer expertise

2. Some familiarity

Paper summary

The paper motivates research on the introduced pragmatic alignment framework (as opposed to alignment known in large language models that work through reinforcement/imitation learning with human feedback to finetune transformer models). Pragmatic alignment represents a language game that wants to model the social dynamics of code. The particular approach that grounds the framework in an example is syntax repair. The paper motivates design decisions necessary to make

syntax repair practically viable and describes the methodology to evaluate the approach against reasonable baselines (Seq2Parse, BIFI).

Comments for authors

The paper is very well written and the idea of a pragmatic alignment framework to model the social dynamics of code is intriguing.

The motivation stayed quite abstract in its description, so even though it was nice to read, it was hard to imagine specifics. So I liked that the rest of the paper wanted to exemplify the framework with a concrete approach in mind (syntax repair). However, I feel like the attempted mapping from pragmatic alignment described in the motivation to the description of TidyParse in the problem description and approach has a bit of a disconnect. The problem description tries to establish this connection but is too vague to actually understand the components of an agent-based language game that was promised earlier in the paper. As part of the presentation, maybe you can provide a more explicit connection between the abstract framework and the concrete application (i.e., Tidyparse).

If I try to fill in the gaps myself or just forgo the framing of this as a language game, I find the approach and evaluation description convincing and interesting. I think that also as part of the symposium, we can provide feedback on some of the framing in the approach, and the evaluation methodology (particularly on a potential user study that is theoretically planned).