Ad Hoc Syntax-Guided Program Reduction

Jia Le Tian
School of Computer Science
University of Waterloo
Canada
tom.tian@uwaterloo.ca

Mengxiao Zhang
School of Computer Science
University of Waterloo
Canada
m492zhan@uwaterloo.ca

Zhenyang Xu
School of Computer Science
University of Waterloo
Canada
zhenyang.xu@uwaterloo.ca

Yongqiang Tian
School of Computer Science
University of Waterloo
Canada
yongqiang.tian@uwaterloo.ca

Yiwen Dong
School of Computer Science
University of Waterloo
Canada
yiwen.dong@uwaterloo.ca

Chengnian Sun
School of Computer Science
University of Waterloo
Canada
cnsun@uwaterloo.ca

ABSTRACT

Program reduction is a widely adopted, indispensable technique for debugging language implementations such as compilers and interpreters. Given a program P and a bug triggered by P, a program reducer can produce a minimized program P^* that is derived from P and still triggers the same bug. Perses is one of the state-of-the-art program reducers. It leverages the syntax of P to guide the reduction process for efficiency and effectiveness. It is language-agnostic as its reduction algorithm is independent of any language-specific syntax. Conceptually to support a new language, Perses only needs the context-free grammar G of the language; in practice, it is not easy. One needs to first manually transform G into a special grammar form PNF with a tool provided by Perses, second manually change the code base of Perses to integrate the new language, and lastly build a binary of Perses.

This paper presents our latest work to improve the usability of Perses by extending Perses to perform ad hoc program reduction for any new language as long as the language has a context-free grammar G. With this extended version (referred to as $\text{Perses}^{\text{ad}}_{\text{hoc}}$), the difficulty of supporting new languages is significantly reduced: a user only needs to write a configuration file and execute one command to support a new language in Perses, compared to manually transforming the grammar format, modifying the code base, and re-building Perses.

Our case study demonstrates that with $\text{Perses}^{\text{ad}}_{\text{hoc}}$, the Perses-related infrastructure code required for supporting GLSL can be reduced from 190 lines of code to 20. Our extensive evaluations also show that $\text{Perses}^{\text{ad}}_{\text{hoc}}$ is as effective and efficient as Perses in reducing programs, and only takes 10 seconds to support a new language, which is negligible compared to the manual effort required in Perses. A video demonstration of the tool can be found at https://youtu.be/trYwOT0mXhU.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3613101

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

ACM Reference Format:
Jia Le Tian, Mengxiao Zhang, Zhenyang Xu, Yongqiang Tian, Yiwen Dong, and Chengnian Sun. 2023. Ad Hoc Syntax-Guided Program Reduction. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3611643.3613101

1 INTRODUCTION

Given a program P and a property ψ that P has, program reduction outputs a minimized program P^* derived from P by automatically removing the program elements in P that are irrelevant to ψ [7, 15, 17, 20, 21]. The property ψ is general and denotes any property of P that can be mechanically checked so that program reduction can be fully automated.

Program reduction is an effective technique for debugging language tools such as compilers, interpreters, linkers, and debuggers. In this case, ψ usually refers to that P triggers a bug in a language tool which takes P as an input. Program reduction has been widely used in practice. For example, both the communities of GCC and LLVM request bug reporters to reduce the bug-triggering program before reporting a compiler bug [3, 14]; the Mozilla community also recommends to use Lithium [18] to reduce web documents consisting of HTML, CSS and JavaScript that trigger bugs in Firefox [2].

Perses Perses [20] is one such program reducer. It is fast evolving, actively maintained, and open-sourced [19]. It leverages the formal syntax of a language to guide the reduction process for short reduction time and small results; meanwhile, Perses is language-agnostic because its reduction algorithm does not rely on any language-specific knowledge. Conceptually, to support a language, Perses only requires the context-free grammar G of the language. Currently, Perses has built-in support for a wide range of languages such as C, Rust, Scala, Python3 and Solidity.

However, in practice, though it is easy for the Perses developers to extend Perses to support a new language, it can be difficult for the user of Perses. The practical challenges are three-fold. First, the user needs to know how to run a tool that is embedded inside Perses to normalize the grammar G into a special grammar form PNF [20].

Second, the user needs to manually change certain parts of the code base of Perses to integrate the normalized grammar. Lastly, the user needs to know the build system of Perses (*i.e.*, Bazel [4]) and build a binary of Perses. The whole process can take the user considerable time if the user is not familiar with the code base of Perses.

Perses$^{ad}_{hoc}$ In this demonstration, we present an improved infrastructure of Perses referred to as Perses$^{ad}_{hoc}$ that significantly improves the usability of Perses by allowing a user to dynamically supply the grammar G of a desired language. Perses$^{ad}_{hoc}$ automatically normalizes G into the PNF form, generates necessary code and loads them when the user wishes to perform program reduction with the newly supported grammar. This work significantly improves the usability and accessibility of a critical tool in program reduction. The usefulness of Perses$^{ad}_{hoc}$ can be demonstrated with a real-world example [13] that leverages Perses to perform program reduction for GLSL [5] programs. With the help of Perses$^{ad}_{hoc}$, the amount of the Perses-related infrastructure code written by compiler researchers could have been reduced from 190 lines to only 20 while also removing the requirement of having a deep understanding of the codebase of Perses.

Contributions We make the following main contributions.

- We propose Perses$^{ad}_{hoc}$, an improved language-agnostic program reducer that bridges the gap between research prototyping and tool usability in practice.
- Our case study on GLSL demonstrates that Perses$^{ad}_{hoc}$ can indeed improve the usability of Perses by reducing the barrier of supporting a new language in Perses. Our extensive evaluations also show that Perses$^{ad}_{hoc}$ is as effective and efficient as Perses in reducing programs, and only takes 10 seconds to support a new language, which is negligible compared to the manual effort required in Perses.
- Perses$^{ad}_{hoc}$ is open-sourced at https://github.com/uw-pluverse/perses and the documentation is available at https://github.com/uw-pluverse/perses/blob/master/doc/install_new_languages.md.

2 BACKGROUND: PERSES

This section discusses the workflow of Perses and highlights the challenges for Perses users of supporting a new language in Perses.

Figure 1 shows the general workflow of Perses. To reduce P *w.r.t.* ψ, Perses needs the grammar of the language of P and the grammar needs to be statically integrated into Perses. Assume that P is written in GLSL [5] and Perses has not supported GLSL yet. To enable Perses to reduce GLSL programs, the user of Perses needs to

(1) have the grammar file `GLSL.g4`
(2) transform `GLSL.g4` to PNF [20] with a Perses-provided tool
(3) write necessary integration and configuration code in Kotlin, Java and Bazel BUILD files[1]
(4) and statically link the grammar and all the code into Perses, as shown in Figure 1.

The current challenge with Perses resides in the convenience of adding new grammars to the tool. Yet, there is currently no automated infrastructure to integrate it with Perses. To support a new language, users generally need to have a deep understanding of the code base of Perses.

[1]Perses is written in Kotlin and Java and built with Bazel.

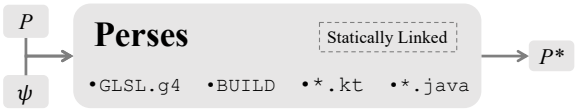

Figure 1: The workflow of Perses.

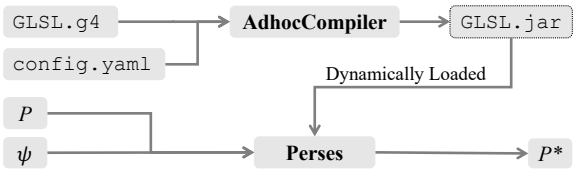

Figure 2: The workflow of Perses$^{ad}_{hoc}$.

3 TOOL IMPLEMENTATION

This section introduces the implementation of Perses$^{ad}_{hoc}$ and explains some specific use cases for this tool.

3.1 Tool Implementation

Figure 2 shows the overview of Perses$^{ad}_{hoc}$. To allow users to dynamically supply the grammar of the desired language, Perses$^{ad}_{hoc}$ is implemented with a grammar compiler, named `AdhocCompiler`, which takes as input a grammar and a language configuration and outputs a Java library in the JAR file format. The language configuration contains meta information of the language, such as the language file extensions and the reduction formats [20] this language supports. The JAR file produced by `AdhocCompiler` contains the necessary compiled Java code for supporting the new language. Finally, code modifications were made in existing Perses to allow either program reduction with a pre-supported language or a dynamically loaded Perses$^{ad}_{hoc}$ JAR file.

Grammar Library Generation `AdhocCompiler` first invokes the internal PNF normalizer to transform the input grammar to PNF. Then, with the configuration file and predefined stub templates, `AdhocCompiler` generates Java source code that can be integrated into Perses to support the new language. After generating the PNF grammar and all the necessary code, `AdhocCompiler` invokes the Java Compiler APIs to compile the code and packages all files into a JAR file to be loaded for the following program reduction process.

Reduction with the Grammar Library Perses$^{ad}_{hoc}$ extends Perses to take the grammar library generated by `AdhocCompiler`, as an extra input. Then Perses dynamically loads this library into the Java Virtual Machine process, and uses the library code to parse P and guide program reduction. In this way, what users need to do is first generating the grammar library with the grammar compiler, and then passing the program and test script along with the grammar library to Perses.

3.2 Application Scenario

We use a real-world example to illustrate the application scenario of Perses$^{ad}_{hoc}$, *i.e.*, to help the developers perform program reduction for languages currently not yet natively supported by Perses.

Table 1: The files changed to manually support GLSL without Perses$^{ad}_{hoc}$.

File	#Line	Purpose
PnfGlslParserFacade.java	89	Incorporate GLSL into Perses
SingleParserFacadeFactory.kt	3	
Bazel Build Files	67	Configure compilation
LanguageGLSL.kt	33	Configure GLSL

Program Conditioning [13] is a recent technique to detect bugs in OpenGL Shading Language (GLSL) compilers. In this work, Perses is used as the default option to reduce test cases, since "Perses performs best with respect to both time taken for reduction and size of fully-reduced test cases", according to the authors.

Without Perses$^{ad}_{hoc}$ Since Perses did not natively support GLSL then, the authors of Program Conditioning [13] manually implemented such a support in Perses. In addition to the GLSL grammar file provided by ANTLR [1], they added or modified four files involving 200 lines of code, as shown in Table 1. To make proper changes to Perses without inducing bugs, the authors have to understand the codebase of Perses, including the logic flow, design patterns, and build configurations, *etc.*. Furthermore, the implementation requires an in-depth understanding of the custom build macros specific to Perses. These macros are not standard in Bazel and do not have any documentation; thus, the developers needed to read through the macro definitions within the project. The Git history shows that the duration from the first commit involving incorporating GLSL into Perses to the last one is around 6 hours.

With Perses$^{ad}_{hoc}$ Perses$^{ad}_{hoc}$ can significantly reduce the labor work of developers to use Perses on GLSL programs, as well as other languages. With Perses$^{ad}_{hoc}$, users only need to download the GLSL grammar, write a language configure shown in Figure 3, and generate `GLSL.jar` file using `AdhocCompiler` with the grammar and the language configure as inputs. Next, Perses can be used to reduce GLSL programs by taking the generated grammar library as an extra input. Figure 4 shows the commands to generate the grammar library and run Perses to reduce GLSL programs. The entire process excluding reduction may only take 10 minutes.

```
---
name: "glsl" # Name of Grammar
extensions: # Acceptable File Extensions
  - "glsl"
  - "comp"
  - "frag"
  - "vert"
origCodeFormatControl: "ORIG_FORMAT" # Reduced Program Format
defaultCodeFormatControl: "COMPACT_ORIG_FORMAT"
allowedCodeFormatControl:
  - "COMPACT_ORIG_FORMAT"
  - "ORIG_FORMAT"
```

Figure 3: The YAML configuration of GLSL in Perses$^{ad}_{hoc}$

In summary, Perses$^{ad}_{hoc}$ allows the developers to treat the internal implementation of Perses as a blackbox while still having the flexibility to use Perses on a broad range of languages. Without

```
java -jar adhocCompiler.jar  \ # Generate Jar File for GLSL
    --parser-grammar "GLSL.g4" \ # Parser File
    --start-rule "translation_unit" \ # Start Rule
    --token-names-of-identifiers "IDENTIFIER" \ # Identifier
    --package-name "org.perses.grammar.GLSL" \ # Package Name
    --language-kind-yaml-file "config.yaml" \ # YAML File
    --output "GLSL.jar" # Output File

java -jar perses.jar \ # Run Perses with the GLSL.jar
    --input-file <the input file to be reduced> \
    --test-script <the script specifying property> \
    --language-ext-jars "GLSL.jar" # Jar File
```

Figure 4: Commands to use Perses$^{ad}_{hoc}$ on GLSL.

Perses$^{ad}_{hoc}$, users have to manually revise Perses after understanding the codebase of Perses, which is labor-intensive and bug-prone. We believe Perses$^{ad}_{hoc}$ can effectively reduce the workload of users of Perses and boost their productivity.

4 EXPERIMENT

We conducted several experiments to demonstrate the effectiveness and efficiency of Perses$^{ad}_{hoc}$.

4.1 Effectiveness

We first compare the effectiveness of Perses$^{ad}_{hoc}$ and Perses. For Perses$^{ad}_{hoc}$, the C language support is supplied by a grammar library generated by `AdhocCompiler`, while Perses has native C language support. We collected five C programs used for evaluating Perses, which is collected from the bug repository of the GCC and Clang compilers, as the benchmarks. We measure the number of tokens in the reduced program outputted by Perses$^{ad}_{hoc}$ and Perses.

Table 2: Effectiveness of Perses$^{ad}_{hoc}$ and Perses

Subject	Property (ψ)	Original Tokens	Perses	Perses$^{ad}_{hoc}$
clang-23353	Crash	30,196	98	98
clang-22382	Crash	21,068	144	144
gcc-65383	Miscompile	43,942	153	153
gcc-66186	Miscompile	47,481	328	328
gcc-71626	Crash	6,133	51	51
Mean		29,764	155	155

Table 2 shows the results. The "Original Tokens" column shows the number of tokens in the original program and the other two columns show the number of tokens in the final minimized programs outputted by Perses and Perses$^{ad}_{hoc}$ respectively. The results show that Perses$^{ad}_{hoc}$ and Perses have the identical number of tokens in each reduced program, demonstrating that Perses$^{ad}_{hoc}$ with the ad hoc language support performs at the same level as Perses with native language support in terms of effectiveness.

4.2 Efficiency

We also measure the efficiency of Perses$^{ad}_{hoc}$ and Perses using two metrics shown as follows:

Time The time that the reduction process takes

Speed The reduction speed, *i.e.*, $\frac{\text{Number of deleted tokens}}{\text{Time}}$

Table 3 shows the efficiency of $\text{Perses}^{\text{ad}}_{\text{hoc}}$ and Perses. Across these metrics, there is no significant difference between $\text{Perses}^{\text{ad}}_{\text{hoc}}$ and Perses with native support. In terms of reduction time T, $\text{Perses}^{\text{ad}}_{\text{hoc}}$ takes 1,222 seconds on average, which is marginally longer (2.6%) than that of Perses, *i.e.* 1,191 seconds. These results demonstrate that compared to the native support by Perses, the ad hoc support used by $\text{Perses}^{\text{ad}}_{\text{hoc}}$ almost has no side effect on efficiency.

Table 3: Reduction Efficiency of $\text{Perses}^{\text{ad}}_{\text{hoc}}$ and Perses

Subject	Perses		$\text{Perses}^{\text{ad}}_{\text{hoc}}$	
	$Time_{(sec)}$	$Speed_{(\#/sec)}$	$Time_{(s)}$	$Speed_{(\#/sec)}$
clang-23353	917	32.82	926	32.50
clang-22382	756	27.67	789	26.51
gcc-65383	1,527	28.68	1,725	25.38
gcc-66186	2,701	17.46	2,611	18.05
gcc-71626	56	108.61	57	106.7
Mean	1,191	43.05	1,222	41.828

Time to Generate Grammar Libraries We measured the time spent by `AdhocCompiler` in generating grammar libraries. Note that for each language, such generation only needs to be performed once. Table 4 shows the time spent in the compilation of six languages. On average, it takes around 10 seconds to compile a language, while the longest one, Java (8), takes around 16 seconds. This result demonstrates that the compilation time has little impact on the overall efficiency of $\text{Perses}^{\text{ad}}_{\text{hoc}}$.

Table 4: Generation Efficiency of Grammar Libraries.

	JSON	C (11)	Scala	Rust	C++ (14)	Java (8)
Non-Blank Lines	55	810	1,160	1,830	872	1,475
Time (s)	2.483	8.679	8.684	14.356	14.648	15.672

5 RELATED WORK

Various program reduction techniques have been proposed by researchers. However, most of them often fall short in terms of either flexibility, *i.e.*, their applicability to different languages , or performance, *i.e.*, the reduction time and the size of reduced programs.

Delta Debugging (DD) [21] is a general minimization algorithm that first divides the program into chunks of equal size and then keeps attempting to remove a chunk or the complement of the chunk from the program. The chunk size is initialized with half the size of the program and is halved whenever nothing can be removed with the current chunk size. This strategy forms the basis of many reduction tools, including Lithium [18], a tool used to reduce test cases for Mozilla Firefox. It employs a similar approach but goes a step further. When the chunk size is reduced to 1, Lithium continually traverses each chunk, persistently trying to remove it until no further reduction can be achieved. This continual refinement at chunk size 1 is a key aspect distinguishing Lithium from classic Delta Debugging. However, both Delta Debugging and Lithium treat lines, characters, or tokens as atomic units for deletion, without considering the hierarchical structure of the program, which can lead to the generation of syntactically invalid programs.

Hierarchical Delta Debugging (HDD) is an advanced reducer that takes into account the hierarchical structure of a program. In this way, the number of syntactically invalid programs generated during the reduction process is reduced, and thus the reduction performance can be significantly improved. Picireny [8], an implementation of HDD, accepts any input program and its corresponding grammar. It uses ANTLR4 to generate a lexer and a parser, which transform the source program into a parse tree for HDD. However, a significant limitation is that Picireny applies DD directly on each level of the parse tree which can still generate invalid programs during the reduction process. In contrast, $\text{Perses}^{\text{ad}}_{\text{hoc}}$ addresses this limitation by deleting nodes from the parse tree with the guidance of the syntax. This approach completely prevents the generation of syntactically invalid programs during reduction, making $\text{Perses}^{\text{ad}}_{\text{hoc}}$ a more effective and efficient solution for program reduction tasks.

There are also many language-specific program reducers [6, 9–12, 16, 17]. Notably, C-reduce [17] is designed to reduce C and C++ programs. Since C-reduce leverages C/C++-specific transformation in the reduction, it has limited support for languages other than C/C++. By contrast, $\text{Perses}^{\text{ad}}_{\text{hoc}}$ is a language-agnostic tool for program reduction. Its design enables it to easily support new languages, enhancing its flexibility in application.

6 CONCLUSION

Program reduction serves as a powerful tool for developers to pinpoint sources of specific program behaviors. To that end, Perses is a state-of-the-art program reduction tool that is language-agnostic, fast, and provides small reduced programs. However, the laborious process of integrating a new language for Perses limited its utility. To address the limitations, we introduce $\text{Perses}^{\text{ad}}_{\text{hoc}}$, an improved language-agnostic program reducer that is easy to extend to previously unsupported grammar without the need to modify and extend Perses itself. With $\text{Perses}^{\text{ad}}_{\text{hoc}}$, users only need to provide the grammar and a simple configuration file to make Perses support a new language. Our evaluation showed that $\text{Perses}^{\text{ad}}_{\text{hoc}}$ matches the performance of Perses on languages with native support in Perses. $\text{Perses}^{\text{ad}}_{\text{hoc}}$ took 1,222 seconds on average to reduce five real-world programs, only marginally longer (2.6%) than Perses, *i.e.* 1,191 seconds with almost no side effect on the final size of the reduced programs. With $\text{Perses}^{\text{ad}}_{\text{hoc}}$, it takes only an average of 10 seconds to install a new language. Through $\text{Perses}^{\text{ad}}_{\text{hoc}}$, we have greatly improved the versatility of Perses and broadened its potential across different languages and applications.

ACKNOWLEDGMENTS

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grant, a project under Waterloo-Huawei Joint Innovation Lab, and CFI-JELF Project #40736.

REFERENCES

[1] ANTLR. 2017. *The ANTLR Parser Generator.* Retrieved 2022-09-20 from https://www.antlr.org/

[2] Firefox. 2017. *Using Lithium to Reduce Bugs in Firefox.* https://github.com/MozillaSecurity/lithium/blob/master/src/lithium/docs/using-for-firefox.md, accessed: 2022-11-05.

[3] GCC. 2017. *A Guide to Testcase Reduction.* https://gcc.gnu.org//A_guide_to_testcase_reduction, accessed: 2021-01-05.

[4] Google. 2015. *About Bazel.* Retrieved May 10, 2023 from https://bazel.build/about

[5] Khronos Group. 2019. *The OpenGL Shading Language Version 4.60.7.* Retrieved May 10, 2023 from https://registry.khronos.org/OpenGL/specs/gl/GLSLangSpec.4.60.pdf

[6] Kihong Heo, Woosuk Lee, Pardis Pashakhanloo, and Mayur Naik. 2018. Effective Program Debloating via Reinforcement Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 380–394. https://doi.org/10.1145/3243734.3243838

[7] Satia Herfert, Jibesh Patra, and Michael Pradel. 2017. Automatically Reducing Tree-Structured Test Inputs. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering* (Urbana-Champaign, IL, USA) *(ASE 2017)*. IEEE Press, 861–871.

[8] Renáta Hodován, Akos Kiss, Daniel Vince, and Zhiqiang Zang. [n. d.]. *Picireny.* https://github.com/renatahodovan/picireny

[9] JS Delta. 2017. *JS Delta.* https://github.com/wala/jsdelta, accessed: 2017-08-05.

[10] Christian Gram Kalhauge and Jens Palsberg. 2019. Binary reduction of dependency graphs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 556–566.

[11] Christian Gram Kalhauge and Jens Palsberg. 2021. Logical bytecode reduction. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 1003–1016.

[12] Gereon Kremer, Aina Niemetz, and Mathias Preiner. 2021. ddSMT 2.0: Better Delta Debugging for the SMT-LIBv2 Language and Friends. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12760)*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer, 231–242. https://doi.org/10.1007/978-3-030-81688-9_11

[13] Bastien Lecoeur, Hasan Mohsin, and Alastair F. Donaldson. 2023. Program Reconditioning: Avoiding Undefined Behaviour When Finding and Reducing Compiler Bugs. *Proceedings of the ACM Programming Languages* 7 (2023).

[14] LLVM. 2017. *How to submit an LLVM bug report.* https://llvm.org/docs/HowToSubmitABug.html, accessed: 2021-01-01.

[15] Ghassan Misherghi and Zhendong Su. 2006. HDD: Hierarchical Delta Debugging. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. Association for Computing Machinery, New York, NY, USA, 142–151. https://doi.org/10.1145/1134285.1134307

[16] Aina Niemetz and Armin Biere. 2013. ddSMT: a delta debugger for the SMT-LIB v2 format. In *Proceedings of the 11th International Workshop on Satisfiability Modulo Theories, SMT*. 8–9.

[17] John Regehr, Yang Chen, Pascal Cuoq, Eric Eide, Chucky Ellison, and Xuejun Yang. 2012. Test-Case Reduction for C Compiler Bugs *(PLDI '12)*. Association for Computing Machinery, New York, NY, USA, 12 pages. https://doi.org/10.1145/2254064.2254104

[18] Mozilla Security. 2008. Lithium: Line-Based Testcase Reducer. https://github.com/MozillaSecurity/lithium, accessed: 2021-01-01.

[19] Chengnian Sun, Yuanbo Li, Qirun Zhang, Tianxiao Gu, and Zhendong Su. 2018. *Perses: Syntax-Directed Program Reduction.* Retrieved May 10, 2023 from https://github.com/uw-pluverse/perses

[20] Chengnian Sun, Yuanbo Li, Qirun Zhang, Tianxiao Gu, and Zhendong Su. 2018. Perses: Syntax-Guided Program Reduction. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 361–371. https://doi.org/10.1145/3180155.3180236

[21] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. *IEEE Trans. Softw. Eng.* 28, 2 (Feb. 2002), 183–200. https://doi.org/10.1109/32.988498

7 APPENDIX

7.1 Walkthrough

In order to use the Perses$^{\text{ad}}_{\text{hoc}}$ flow, the language configuration and grammar file are needed. The configuration must be in `YAML` and follows the same format found in Figure 3. The grammar needs to be specified in `Antlr`. The grammar file can be a single, combined grammar or seperate lexer and parser grammars.

We then build and call the Perses$^{\text{ad}}_{\text{hoc}}$ installer using the command in Figure 5. This produces a Jar file that is needed in phase two of the Perses$^{\text{ad}}_{\text{hoc}}$ flow. Note that the variable `OUTPUT_JAR` defines the name and location of the Jar upon completion.

```
# Parameters
readonly PARSER_GRAMMAR_FILE="Parser.g4"
readonly OPTIONAL_LEXER_GRAMMAR_FILE="Lexer.g4" # This is optional
readonly START_RULE_NAME="compilationUnit" # The start rule of the grammar
readonly LANGUAGE_KIND_YAML_FILE="language_kind.yaml"
readonly IDENTIFIER_TOKEN_NAMES="IDENTIFIER" # The name of the token type Identifier
readonly PACKAGE_NAME="org.perses.grammar.adhoc.mygrammar" # Name your own package
readonly OUTPUT_JAR="my_grammar.jar"

# Building Perses Adhoc Installer
bazel build //src/org/perses/grammar/adhoc:perses_adhoc_installer_deploy.jar

# Installing Language and Generating Jar
java -jar bazel-bin/src/org/perses/grammar/adhoc/perses_adhoc_installer_deploy.jar \
    --parser-grammar "${PARSER_GRAMMAR_FILE}" \
    --lexer-grammar "${OPTIONAL_LEXER_GRAMMAR_FILE}" \
    --start-rule "${START_RULE_NAME}" \
    --token-names-of-identifiers "${IDENTIFIER_TOKENS}" \
    --package-name "${PACKAGE_NAME}" \
    --language-kind-yaml-file "${LANGUAGE_KIND_FILE}" \
    --output "${OUTPUT_JAR}"
```

Figure 5: Generate Jar File for Language

After installing the grammar, build the main Perses Jar. Then invoke the command in Figure 6 to run Perses to reduce programs written in the new language.

```
# Building Perses
bazel build //src/org/perses:perses_deploy.jar
# Running Perses with New Language Support
java -jar bazel-bin/src/org/perses/perses_deploy.jar \
    --input-file <the input file to be reduced> \
    --test-script <the script specifying the property> \
    --language-ext-jars "${OUTPUT_JAR}"
```

Figure 6: Run Perses with Installed Language

A specific example of generating the Jar File for a language and running Perses with the Jar File can be found in Figure 4. A ReadMe documentation for using Perses$^{\text{ad}}_{\text{hoc}}$ can be found at https://github.com/uw-pluverse/perses/blob/master/doc/install_new_languages.md.

Received 2023-05-11; accepted 2023-07-20