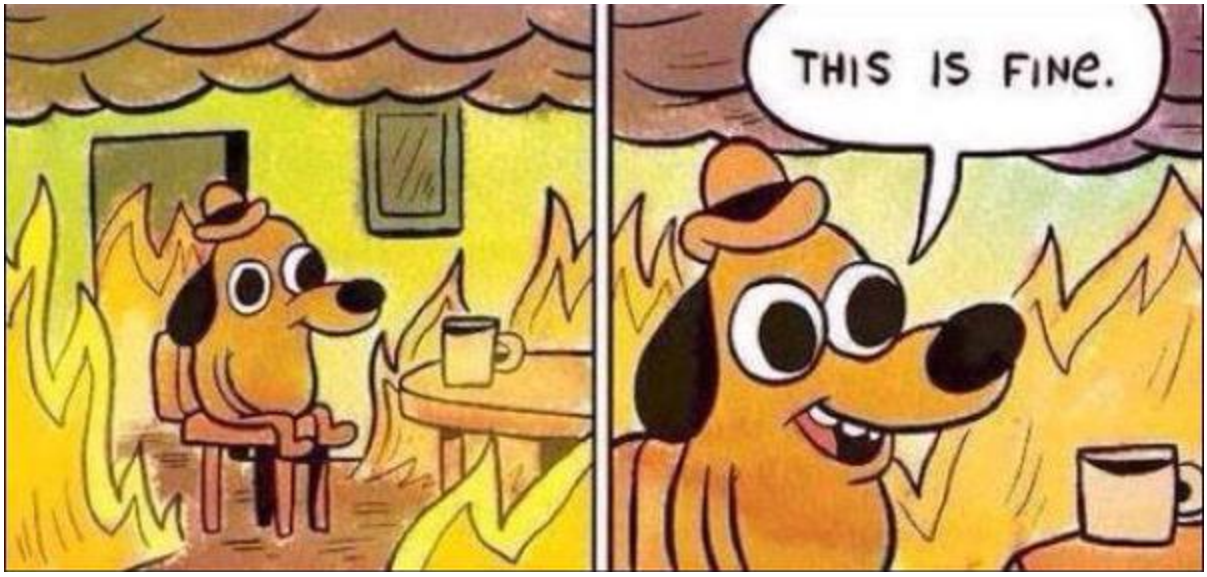
CS380C - Spring 2015
Matt Allen, Josh Eversmann Michael Gregory
Final Project Technical Report

Off-Broadway: A Partial Implementation of the Broadway Framework



Credit: Gunshow Comic

- Project Goals
    - Programs often contain assumptions about valid data and operations that aren't visible to the compiler. Without those assumptions, compilers miss valuable opportunities for both optimization and error checking.
    - There are existing annotation frameworks that help solve this problem, but none of them are implemented in llvm, and there is still work to be done in creating better annotation languages.
- Scope of Solution
    - We intend to implement a subset of the Broadway annotation specification in llvm, exposing additional optimizations for programmers using llvm.
        - We primarily focused on implementing proper parsing of the grammar, and a subset of features that support arbitrary use of property lattices for dataflow analysis.
        - We have partial implementations of the report and analyze directives, as well as an Alias Analysis with access to the on_entry and on_exit information, which it does not know how to use.
    - Instead of implementing Broadway's client driven pointer analysis, we used llvm's built in Alias Analysis framework, which did not prove powerful enough for our needs
- Overall Infrastructure
    - Python parser generator library Grako, available on GitHub, runs on the annotation file after any C preprocessor and generates a json AST for the pass to use
    - llvm::FunctionPass for interacting with the bytecode of the program we are analyzing
    - C++ rapidjson framework for deserializing and traversing json data in the AST
    - We have exposed an alias method which is part of the llvm alias analysis, an implementation of pointer analysis could be put there and easily given access to Broadway's extra function metadata.
        - If it were stable/buildable could use poolalloc, which is where llvm moved all interprocedural analyses, such as Steensgaard
- Development Process
    - Clang plugin
        - Initially, we considered adding a plug-in to the Clang preprocessor to handle a superset of the C language, namely Broadway. Unfortunately, Clang does not support plugins that handle syntax other than C, and we realized we would need to fork Clang to add Broadway functionality to it.
        - In addition, clang's AST provides many correctness guarantees which we would be unable to meet with the Broadway metadata
    - C++ Parsing
        - Next, we considered writing a C++ module that parsed the Broadway annotations, but because of the structure of llvm passes, that dependency would not be any easier for a third party to manage than

arbitrary code, and C++ does not have as much library support for parsing grammars as other languages.

- Python Parsing
  - Given python's strong library support for parsing and AST construction, we settled on using the Grako library to parse Broadway files before the execution of the llvm passes; a version of the annotation's AST serialized as json can be easily provided to the llvm pass.
  - We used the rapidjson C++ library here to deserialize the annotation framework, but started running short on time and cut features of the language from the C++ representation.
- Broadway Pass
  - We used Matt's assignment 3 data flow analysis framework to perform much of the analysis work, but in order to handle lattices specified by the Broadway annotations' property directive, we designed and built a new BroadwayLattice data structure to perform lattice operations on arbitrary level lattices to be specified at runtime.
- Alias Pass
  - Once we had the AST for the Broadway annotations, we needed to access and modify the points-to sets of the variables in the executed code. In llvm, all pointer aliasing is handled in passes that inherit from AliasAnalysis class. Also, Aliases in llvm build on the results of previous alias passes. Thus, we thought that we would run a default llvm alias analysis to build basic points-to sets, and then modify those sets with more specific information derived from the broadway annotations, since we have stronger guarantees about the behavior of pointers inside of function calls. During this process, we discovered that llvm's AliasSetTracker assumes when an instruction is a CallInst nothing can be assumed and it places everything in a may alias set, which meant inserting our extra knowledge concerning calls was difficult. We got around this restriction by removing all call instructions from the version of the code analysed by AliasSetTracker, and manually merging sets accordingly to our information.
- Broadway Pass into a Monolith
  - After figuring out how to properly subclass and use the AliasAnalysis and AliasSetTracker classes, we realized that we needed information present only in the BroadwayPass in order to properly modify and use the alias sets created by our custom alias analysis. We ended up merging the two passes so that information and operations previously available were available in both.
- Ignoring Alias Analysis
  - Unfortunately, we then discovered that the alias sets created by an AliasAnalysis pass do not provide the points to information that we need in order to properly integrate Broadway restrictions. After inspecting the

llvm source and documentation further, we discovered that the analysis that would provide information we needed was also moved to poolalloc with the other interprocedural analyses in poolalloc.

- Evaluation
    - We didn't reach feature parity with Broadway
    - We were able to parse and serialize all of the elements of the original Broadway Annotation Language
    - Our deserialization system gave us access to most of the information present in the language within our llvm pass, and we were unable to put all of it to use
    - taint.pal was a simplification of the taint analysis scheme put forth in the Security Analysis paper, with the pointer information stripped out. Our dataflow system is able to recognize errors in simple programs using this system, but is unable to report in a helpful way
    - memory.pal was the file from the original Broadway work that we were most able to make use of. Because it doesn't use any of the advanced pointer decomposition features, a simple points-to analysis would allow our system to produce meaningful results based on its data flow system.
    - The stdlib security annotations made use of more of the features of the annotation language, and we were unable to produce meaningful results with them.
    - The PLApack annotations used almost every feature of the annotation language, and would have been unfeasible to produce results from it; in addition, we didn't have PLApack itself to link against, or any code that used it for us to analyze.
- Challenges and Future Work
    - No single paper contained an accurate description of the Broadway Grammar, different things seem to have been added and renamed at different times; many things aren't fully discussed and some are never even mentioned. We assumed that the annotation files we were provided were the canonical iteration of the grammar, but we observed several issues with its construction that could be improved if someone sat down to rewrite a comprehensive annotation grammar.
    - Because we chose to operate on llvm's bytecode rather than on a C AST, replace-with would have been difficult to implement as specified, as it was designed around string concatenation and mapping a C function to IR is difficult in this context. We designed some partial solutions to this problem which we were unable to implement.
    - llvm's Alias Analysis is not the type of pointer analysis that Broadway was built to interact with, and since they've removed the points-to analyses from their source base and apparently stopped maintaining them, we were unable to strongly integrate with any available pointer analysis.

CS380C - Spring 2015
Matt Allen, Josh Eversmann Michael Gregory
Final Project Technical Report

- Background Material
    - llvm Source Code/Doxygen
    - S. Guyer and C. Lin, “An Annotation Language for Optimizing Software Libraries”, Second Conference on Domain Specific Languages. October, 1999. pp. 39-53.
    - S. Guyer and C. Lin, “Error Checking with Client-Driven Pointer Analysis,” Science of Computer Programming Journal, vol 58, 2005, pp. 83–114.
    - S. Guyer and C. Lin, “Broadway: A Compiler for Exploiting the Domain-Specific Semantics of Software Libraries,” Proceedings of the IEEE, vol 93, issue 2, 2005, pp. 342-357.
    - S. Guyer, “Incorporating Domain-Specific Information into the Compilation Process,” The University of Texas at Austin, May 2003.
    - W. Chang, B. Streiff, and C. Lin, “Efficient and Extensible Security Enforcement Using Dynamic Data Flow Analysis, ”Computer and Communications Security, 2008, pp. 39-50.

- Project personnel. If you work in a group, which is encouraged, explain the division of labor.
    - Michael: Pointers and Alias analysis in LLVM.
    - Matt: Parser in python, C++ de-serialization framework.
    - Josh: Test cases and simplified pal files, alias analysis, proof of concepts for llvm interactions, BroadwayLattice
    - We used the data flow framework developed in project 3.