A GSoC 2015 Proposal
Enhancing the Alias Analysis Passes in LLVM

Mingxing Zhang, Tsinghua University

**A** lias analysis (AA) is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the alias information they receive. Thus I think it is necessary and meaningful to increase the precision of the current AA passes in LLVM by making them interprocedural; field-sensitive; and context-sensitive.

**Background**

In general, alias analysis is a technique that used to determine whether or not separate memory references point to the same area of memory. Many applications of program analysis, such as program optimization [1], automatic parallelization [4], and finding bugs [3], need the alias information. And almost all of these program analyses are more effective when given more precise alias information. Moreover, the scalability of such program analyses may also be impacted by AA’s precision [5].

**Motivation**

As a fundamental building block of code optimization, there already exist many alias analysis passes in LLVM, such as basicaa and steens-aa. But these standard LLVM AA passes either take a large amount of time (Anderson Analysis at cubic time and large memory requirements) or are cheap but somewhat imprecise (Steensgard Analysis). In order to address this problem, cfl-aa, which is done at Google, implements a demand-driven, CFL-based methods based on the algorithm derived from recent researches [8, 9]. It provides more accurate result than Steensgard’s algorithm, while significantly faster than Anderson’s. However, the current implementation of cfl-aa is intraprocedural; field-insensitive; and context-insensitive. As explained in the following subsections, these properties imply a huge space for increasing precision.

**Intraprocedural V.S. Interprocedural**

In compiler theory, an intraprocedural analysis means that the analysis is done within the scope of one procedure, while an interprocedural analysis is done across the entire program. Albeit relatively more time-consuming, interprocedural alias analysis is needed in many scenarios. For example, detecting memory leaks rely on interprocedure alias information heavily, since an object with proper destructors will never leak if its life cycle is within a function. Section 5.2 of paper [6] also gives an example of using interprocedure AA to find security vulnerabilities. It tries to assure that a security key will not flow into a String object.

**Field-insensitive V.S. Field-sensitive**

Yong et al [7] gives an in-depth evaluation of the precision difference between field-sensitive and field-insensitive AA, and find that “distinguishing individual fields of structs is important”. In their evaluation, the average pointer sets produced by field-insensitive AA are at least twice as large as the sets produced field-sensitive AA (10X larger in the worst case).
In Section 6.3 (Figure 6) of paper [6], the author compares the effectiveness of type refinement between using context-sensitive AA and context-insensitive AA, and shows a considerable increase. Although the difference is not that significant when compared with the field sensitive, as demonstrated by Guyer et al [2], a small amount of imprecision in isolated parts of the program can significantly impact the effectiveness of the client analysis in specific cases, such as security analysis and parallelization.

**Plan**

There are several things that I plan to do for enhancing the alias analysis passes in LLVM during this summer of code project.

1. Make the alias analysis field-sensitive by representing fields of a struct with separated nodes. In order to handle type casting, I intend to use the “collapse at casting” approach described in paper [7].

2. Handling special global variables, such as errno.

3. Extend cfl-aa to interprocedural analysis. Both context-insensitive and context-sensitive approach will be explored. The implementation of context-sensitive analysis may be based on the cloning technique proposed by Whaley et al [6].

The expectant schedule is given in Table 1.

<table>
<thead>
<tr>
<th>Work</th>
<th>Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Investigating recent researches on alias analysis</td>
<td>1</td>
</tr>
<tr>
<td>Profiling the code</td>
<td>1</td>
</tr>
<tr>
<td>Field-sensitive analysis</td>
<td>3</td>
</tr>
<tr>
<td>Handling special global variables</td>
<td>1</td>
</tr>
<tr>
<td>Context-insensitive interprocedural analysis</td>
<td>2</td>
</tr>
<tr>
<td>Context-sensitive interprocedural analysis</td>
<td>2</td>
</tr>
<tr>
<td>Evaluating the precision</td>
<td>1</td>
</tr>
<tr>
<td>Scrub code, write documents</td>
<td>1</td>
</tr>
</tbody>
</table>

The expectant schedule is given in Table 1.


