Work Project Report: Static Analysis Suite

CERN Summer Student Programme 2013

Student:	**Filip Bártek** <filip.bartek@cern.ch>
Main supervisor:	Danilo Piparo <danilo.piparo@cern.ch>
Second Supervisor:	Benedikt Hegner <benedikt.hegner@cern.ch>
Division:	PH-SFT
Project ID:	13875
Project name:	Static Code Analysis for Future High Energy Physics Data Analysis Frameworks

Introduction

I come from Czech Republic and I study theoretical computer science in Charles University in Prague, Czech Republic. This summer, I've come to CERN as a Summer Student to work in the PH-SFT group under the supervision of Danilo Piparo.
This document describes the project I was working on in CERN during the summer - the initial goal, the domain of interest (briefly), the work I performed and the tools I used, and the results.

Goal

The initial goal of the project was to develop a checker extension for the tool Clang Static Analyzer with focus at thread safety.
Excerpt from the project proposal:

> The goal of the project is to develop a battery of static code analysis rules for the "clang static analyzer" to assist the improvement of the code for thread-safety.

Domain of interest

Static analysis

Static analysis is the process of finding issues in programs without executing these programs. In automatic static analysis, the focus of my project, a program called *static analyzer* performs this task.
In the basic scenario, the static analyzer processes source code of target project and outputs a *bug report* containing information about parts of source code that seem to carry an issue.
The target issues range from coding conventions violations to dangerous bugs that can potentially make the system unreliable and are hard to spot by just reading the code.

Clang Static Analyzer

Clang Static Analyzer is one of the tools available that perform static analysis. It is integrated in *Clang* compiler, sharing the compiler's source code parsing and interpreting functionality. The tool is free and open source.
Clang Static Analyzer is described and available for download on the following website:

> http://clang-analyzer.llvm.org/

Checkers

The specific issue detection in Clang Static Analyzer is performed by *checkers*. A checker is a C++ class that uses Clang API to extract information from Clang compiler and emit reports in cases it detects issues in the analyzed source code.
In other words, Clang maintains the interface between the analyzed source code and issue-specific checkers.
Clang provides the checkers information about the analyzed source code especially in the following forms:

- *Abstract Syntax Tree* (AST) - a tree of statements generated by parsing the source code. This representation is shared with compilation functionality of Clang.
- *Control Flow Graph* (CFG) - a graph of symbolic program states generated by performing symbolic execution of the analyzed program. This representation is specific to the static analysis functionality of Clang.

Checkers act as visitors on these graphs - during static analysis, Clang traverses these graphs and calls the relevant methods of the registered and enabled checkers on each of the nodes.
A checker can be either linked directly into Clang or encapsulated in a shared object (possibly with other checkers), acting as a *Clang Static Analyzer plugin library*.
Clang Static Analyzer basic distribution contains a set of checkers. You can find a (non-exhaustive) list of the checkers distributed with Clang Static Analyzer on the following web page:

http://clang-analyzer.llvm.org/available_checks.html

scan-build

The recommended[1] way of using Clang Static Analyzer is the *scan-build* script (bundled with Clang), a sophisticated wrapper of static analysis function of Clang compiler.
scan-build allows checker selection, checker plugin loading and easy integration of the static analysis into existing build systems.

Project progress

Clang Static Analyzer: getting familiar

During the first weeks of my work on the project, I was getting familiar with Clang Static Analyzer. I performed static analysis using Clang Static Analyzer (with all stable relevant checkers) on FastJet[2], ROOT[3] and Geant4[4], three existing open source projects. The results can be found on the addresses in the following table:

[1] The scan-build script is implicitly presented as the main way to perform static analysis with Clang on Clang Static Analyzer homepage.
[2] http://fastjet.fr/
[3] http://root.cern.ch/
[4] http://geant4.cern.ch/

Project	Version	Bug report
FastJet	3.0.4	https://test-project-sas.web.cern.ch/test-project-sas/temp/fastjet-stable/report/2013-08-29-1/
ROOT	5.34.09	https://test-project-sas.web.cern.ch/test-project-sas/temp/root-53409-stable/report/2013-08-07-1/
Geant4	4.9.6.p02	https://test-project-sas.web.cern.ch/test-project-sas/temp/geant4-stable/report/2013-07-25-1/

A bug in Clang Static Analyzer

While analyzing FastJet, me and my co-supervisor Benedikt Hegner identified a bug in the scan-build script. I filed a bug record and proposed a solution in LLVM's[5] bug tracking system, which is customarily used for tracking bugs in Clang and Clang Static Analyzer. The bug report with relevant discussion can be found on the following web page:

http://llvm.org/bugs/show_bug.cgi?id=16809

Note that since Clang Static Analyzer is work-in-progress, occasional bugs are to be expected.

Static Analysis Suite: development

The next task in the project was the development of *Static Analysis Suite (SAS)* - a checker plugin for Clang Static Analyzer.

CMS Static Analyzer import

CMS Static Analyzer[6] is a checker plugin for Clang Static Analyzer developed in CMS experiment[7] for their internal purposes. I used CMS Static Analyzer as the starting point for SAS.[8]

I created a dedicated SAS remote Git repository and imported the CMS Static Analyzer into it so that I could track my work on the project.

Then I stripped the project of dependencies on CMS, starting by changing the name to Static Analysis Suite and ending by updating the documentation.

New checkers

My main projected task was development of new checkers. Due to time constraints, I only managed to implement two new checkers.

Varname

sas.example.Varname is an example checker. This means that typically Varname checker doesn't perform any useful analysis, but it has well documented and easy to understand source code and functionality.

Varname checker reports every instance of a variable name that doesn't start with an uppercase letter.

I successfully used it in my final presentation as a simple example of a custom checker.

---

[5] *LLVM* is the underlying compiler infrastructure of Clang.

[6] https://github.com/cms-sw/cmssw/tree/CMSSW_7_0_X/Utilities/StaticAnalyzers

[7] http://cms.web.cern.ch/

[8] Permission was obtained by my supervisor Danilo Piparo from the developers of CMS Static Analyzer.

GlobalAccInCtor

sas.security.GlobalAccInCtor is a checker that detects access to global variables from constructors. This can be useful because such access may be unsafe.[9]
GlobalAccInCtor checker reports every access from a constructor to a variable that:

- is declared as globally stored (i.e. global or static member) and
- can't be proved to be initialized by an expression evaluated as constant (at compile time).

I used this checker to find one occurrence of constructor access to a global variable initialized by a function call in FastJet.[10]

Checker muting

I implemented a basic checker muting functionality - a system that can be used to suppress false positives fired by SAS checkers.
The muting is local (specific to a source code line) and selective (specific to a checker).
To mute a checker (disable its check) on a specific source code line, the user can insert a specifically formatted comment[11] on the directly preceding line.
To make a checker support muting, the checker's author must prevent the checker from reporting an issue in case the function *IsDisabled* provided in *CheckerDisabler.h* returns *true* on the analyzed statement or declaration.
Detailed instructions can be found in SAS documentation.

Documentation

I created a comprehensible documentation for the SAS. The main part of the documentation is contained in the *README*[12] file. The file contains instructions for:

- building SAS,
- performing static analysis with SAS,
- muting checker warnings and
- adding new checkers to SAS.

I distributed specific parts of the documentation in the relevant source code files and scripts. These files are referenced from the README file in appropriate parts.

Technologies used

JIRA

For task management of the project, me and my supervisors used the task management system JIRA. In the project's JIRA page, we kept a list of tasks related to SAS.
I found especially the following features helpful:

- "Tasks" to keep track of the work items,
- "Comments" and "Descriptions" to document my work on the project and
- "Statuses" to mark tasks as "Open", "In progress" or "Resolved".

I found JIRA especially useful when I needed to recall details of some past actions performed

[9] http://julipedia.meroh.net/2005/10/c-constructors-and-global-data.html
[10] https://test-project-sas.web.cern.ch/test-project-sas/temp/fastjet-globalaccinctor/2013-09-02-1/report-31fa29.html#EndPath
[11] Example: `// sas[disable_checker : "sas.example.Varname"]`
[12] https://git.cern.ch/web/?p=sas.git;a=blob;f=README (internal to the PH-SFT group)

in the project. In this sense, the JIRA project pages can be viewed as structured and continually actualized documentation of the project.
You can find all the tasks related to SAS on the following web page:

https://sft.its.cern.ch/jira/browse/CFHEP-130

Git

For source code management, I used the source management system Git. I used a remote Git repository provided by CERN. Git has proved very useful to me because it allows working locally and supports easy committing, branching and merging.
You can find the web page of the SAS Git repository on the following web page:

https://git.cern.ch/web/?p=sas.git (internal to the PH-SFT group)

Other

Other notable technologies I've used in my project:

- LLVM
- Clang
- Clang Static Analyzer
- CMake

I used a computer provided by CERN with the Scientific Linux CERN operating system distribution. For computation-heavy tasks, I used the remote machines in the lxplus cluster.

Final presentation

By the end of my stay here, I did a presentation for the PH-SFT group about the SAS project. The slideshow I created for this presentation is available for download at the following web page:

https://indico.cern.ch/conferenceDisplay.py?confId=267747

Conclusion

During my stay in CERN, I implemented Static Analysis Suite, a checker plugin for Clang Static Analyzer. This plugin contains several checkers and allows for extension with new checkers. It is documented for both analyzing target projects and adding new checkers. I released the SAS internally in PH-SFT group.[13]
During the work on this project, I deepened my knowledge of C++ language, gained confidence in using Git for source code management and learned to use JIRA for task management. I got introduced to Clang Static Analyzer and learned to use it for analyzing arbitrary C and C++ projects and to extend it with additional checkers to fit new needs.
I also appreciate my experience with writing user documentation for the final product and preparing a presentation to communicate my results to my colleagues.

[13] Note that the tool is meant become public in the future.