Compiler Engineering
CS6383 - Sep 2019
Mini Assignment 4:

Clang Static Analyzer and Code Compliance

INSTRUCTOR: Dr Ramakrishna Upadrasta

Report By:
Vijay Tadikamalla - CS17BTECH11040
Sai Harsha Kottapalli - CS17BTECH11036
Sagar Jain - CS17BTECH11034
Shraiysh Vaishay - CS17BTECH11050
Jatin Sharma - CS17BTECH11020

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Code Compliance

Code compliance (sometimes called **Code enforcement**) is compliance with a written, adopted or required a set of rules that have been set into a code format.
An authority usually enforces a set of rules, or a body of laws and compel those subject to their authority to behave in a certain way.

Significance of Code Compliance rules

We live in a built environment designed around rules to ensure our safety and well-being.
For example, in Computer Science Field, there is a set of comprehensive coding standards best practices that include all of the code development. It helps to ensure that all developers work in the same manner. Consistency has a positive impact on the quality of the program, and one should maintain it while coding.

Implications

Enhanced Efficiency
Developers spend a more substantial part of their time in solving those issues that could have been prevented. Implementing the coding standards would help other developers to detect problems early or even prevent them altogether. This will increase efficiency throughout the software process.

Minimal Complexity
Complex codes are more vulnerable to errors. Coding standards help in the development of software programs that are less complex and thereby reduce the errors.

Easy to maintain
If the coding standards are followed, the code becomes consistent and can be easily maintained.

Bug rectification
It becomes straightforward to locate and correct bugs in the software if the source code is written in a consistent manner.

Cost-efficient
A clear code gives the developers an opportunity to reuse the code whenever required. This can radically reduce the cost along with the efforts put in the development.

If the coding standards are not defined, developers will use any of their own methods, which might lead to certain negative effects such as **Security concerns.**
Software becomes vulnerable to attacks if it is inconsistent, contains bugs and errors in logic. Most of the aforementioned problems arise due to the faulty programming code that might have resulted from poor coding practices.

IEC 61508 / ISO 26262

IEC 61508 (Functional safety standard) is the international standard for electrical, electronic and programmable electronic safety-related systems. It provides requirements for the system to have the required SIL(Safety integrity level). These include how to apply, design, deploy and maintain safety-related systems.
There are 4 SIL's defined, classified based on risk assessment of the system function, namely, A, B, C and D in the increasing order of its danger level.
ISO 26262 is a derivative from IEC 61508 specifically for Automotive Electric/Electronic systems which is currently being widely used.
It details the regulations and recommendations for the development of a product from initial stage(concept/theory) to deployment to decommissioning while also helping in documenting testing process (i.e.) Management, development, production, operation, service and decommissioning. This is also referred to as automotive safety lifecycle.
With the help of ASIL (Automotive SIL's) to determine the necessary requirements of items to attain an acceptable safety level.
Testing being a critical component helps us verify if in different scenarios we get results within safety limits.
Tools under ISO 26262 must provide:

- Software Tool Qualification Plan
- Software Tool Documentation
- Software Tool Classification Analysis
- Software Tool Qualification Report

IEC 62304

It is an international standard for the development of medical device software (applicable to any integral part of the medical device software) widely used for benchmarking for products in the market. It is also referred to as medical device software — software lifecycle processes.
With the assumption of quality management systems and risk management systems, the standard provides 9 sections.

- Scope.
- Normative References.
- Terms and definitions.
- General requirements.
- Software development process.
- Software maintenance process.
- Software risk management process.
- Software configuration management process.
- Software problem resolution process.

The standard categorizes the product into 3 classes :

- A: No injury possible
- B: Minor Injury possible
- C: Serious Injury/Death possible

Compliance can be accelerated by using a static code analyzer such as Helix QAC to enforce coding standards.

MISRA

MISRA is currently famous for its C/C++ standard(especially MISRA C) with the aim to achieve code safety, security and portability for embedded systems. It is used in different fields such as aerospace, telecom, medical devices, etc.
Before the launch of ISO 26262, MISRA was the standard used for software of automobile systems for safety purposes.

The set of guidelines provided by MISRA can be classified into three types:

- Mandatory - Must be satisfied
- Required - Should be satisfied unless there exists a deviation

- Advisory - Only recommended but not necessary by default.

Categorization of rules:

- Avoiding possible compiler differences, for example, the size of a C integer may vary but an INT16 is always 16 bits. (C99 standardized on int16_t.)
- Avoiding using functions and constructs that are prone to failure, for example, malloc may fail.
- Produce maintainable and debuggable code, for example, naming conventions and commenting.
- Best practice rules.
- Complexity limits.

This standard is famous because C is a very essential part in the embedded realm and that it is hard to debug(especially run-time). Following MISRA standard has helped developers not go through most of these bugs and hence reduce the debug time drastically.

There is no official MISRA certification process even though the guidelines are documented. Although, there are third-party tools like Axivion Bauhaus Suite, CodeSonar etc to verify MISRA compliance.

CERT

CERT produced the CERT C Coding Standard with the aim to achieve safety, reliability, and security of software systems. It cross-references to other standards like MISRA, CEW, etc. It has produced standards for C++, Android, Java and Perl.
The guidelines are available here consisting of rules and recommendations. They have also described with many examples per guidelines along with the potential impact.

DO 178B/C

DO 178B/C (Software Considerations in Airborne Systems and Equipment Certification) provides guidelines for safety in software for aerospace systems. DO 178C is the revised version of DO 178B. It consists of guidelines and recommendations/guidance. Based on the potential risk/safety assessment of the software, it can be categorized into five conditions, also referenced to as Software Level/ Design Assurance Level:

- A - Catastrophic – Failure can potentially cause a crash.

- B - Hazardous – Failure can cause harm to passengers as well as increased workload to the crew thereby decreasing efficiency.
- C - Major – Failure can cause discomfort(and/or minor injuries) to passengers as well as increased workload to the crew thereby decreasing efficiency.
- D - Minor – Failure can cause a change in navigated path and other minor inconveniences.
- E - No Effect – Failure has no impact on safety, aircraft operation, or crew workload.

This certification process helps in generating documents for:

- Software Planning
- Software Development
- Software Verification
- Software Configuration Management
- Software Quality Assurance

Static Analysis tools such as CodeSonar are available which serves as a software verification tool.
This standard only serves as a necessary but not a sufficient certification.

Compass Code Compliance Checker - A Summary

Compass Code compliance checker is a tool built for analyzing software and can act on both source code and binaries. It is based on the ROSE compiler infrastructure. It provides a way to enforce predefined or arbitrary user specified properties on software.

Compass has an excellent design which makes it very user friendly and helps users be more productive when using it. It is built with extensibility in mind so that other source analyzers can be built on top of it with relative ease. The extent of its user friendliness is evident in the fact that it also provides a GUI interface for interacting with the infrastructure.

Also, very importantly, Compass is open source.

Purpose and Core Ideas

- Demonstrate the use of ROSE to build lots of simple pattern detectors for C, C++ and FORTRAN.
- Provide a concrete tool to support interactions with lab customers.
- Provide a home for the security analysis specific detectors being built within external research projects.
- Provide an external tool for general analysis of software.
- Provide a tool to support improvements to the ROSE source code base.
- Define an infrastructure for an evolving and easily tailored program analysis tool.
- Provide a simple motivation for expanded use of ROSE by external users.
- To serve as a basis for other source analysis tools

Design

Compass tries to reduce the trust on checker writers and hence includes a Compass verifier which will try to double check the checker who wrote a Compass checker by trying to prove that the checker behaves as expected.

So this is a two level check which reduces the chances of mistakes by a huge factor. Firstly the checker which will check the software and secondly the Compass verifier which will check and verify the Compass checker and ensure that it behaves as expected. This work can now be divided between a checker writer (worker) and a higher authority (admin) and follow a hierarchical fashion.

Compass is designed in a way that allows users who do not necessarily have compiler backgrounds to utilize the ROSE infrastructure to build their own analysis tools. Compass is foremost an extensible open source infrastructure for the development of large collections of rules.

Implementation and Architecture

Compass exposes a plugin interface which makes it easier to deal with.
It utilizes the ROSE compiler infrastructure to analyze and parse the given software and then traverses the AST it receives to implement several checks it contains.

It contains within itself a violation handler which is invoked whenever a violation is encountered for clean execution.

All checkers, such as the ConstCast checker, utilize the abstract classes to traverse a program with all its nodes and to output violations found in that code according to the local algorithm.

CompassMain is the main executable that initially calls ROSE to parse a program. Then buildCheckers is called to load all checkers that are specified within a configuration file.

The configuration file allows users to turn on and turn off specific checkers for their run-time analyses. However, the configuration file only permits checkers to be loaded that were part of Compass at compile-time.

The main interface file compass.h contains the class Checker and the abstract class OutputObject. Checker is the interface to ROSE, giving the metadata for a checker and a function to apply the checker to a given AST. OutputObject aids to output defects found by a specific checker. More functionality to handle e.g. file input and parameters provided to Compass, is provided within the Compass namespace.

Compass Checkers

Default Case

We ensure that each switch in a program has a default option. This is helpful as unexpected cases of fallthrough can cause ‘difficult-to-detect’ bugs.

Implementation

During static analysis, check all statements inside switch statements and search for default statement. If not found, we report it.

Default Constructor

A default constructor must always be created for a class. This is for clarity of the code.

Implementation

Again, during static analysis, traverse AST, visiting member functions of class definitions. If, for any class, a default constructor is not found, report a violation.

Discard assignment

The assignment operator should be used in assignment statements only - as an independent expression statement. For example, it should not be used as a part of a branch condition where it might be confused with the equality operator.

Implementation

Find assignments from a program during static analysis and check for its parent. If the parent is not an expression statement, then it must not be controlling expression statement for iterative or control flow constructs.

Float For Loop Counter

As floating-point arithmetic is not exact but has rounding errors, one must refrain from using them in loop counters.

Implementation

While traversing the AST, find all for loop initialization statements and check for the type of variable declaration. If it is float or double, then we report a violation.

Floating Point Exact Comparison

As floating-point numbers are never exact (and platform-dependent), we should avoid comparison of a floating variable with a floating value.

Implementation

We traverse the AST and try to find a test. If the operands are of floating type (float or double) and the operator is ‘==’ or ‘!=’ then a violation is reported.

Set Pointers To Null

After freeing a pointer, it is advised to set it to NULL. This prevents double-free and access-freed-memory vulnerabilities.

Implementation

Find all the calls to the standard free memory function and check if the value has been set to NULL in the next statement. This can be done in a single AST traversal.

CLANG CHECKERS

ArrayBoundChecker

ArrayBoundChecker is a path-sensitive check which looks for an out-of-bound array element access. Essentially it checks whether all accesses to array elements are to legal elements.

Implementation Details

The function checkLocation is used over Symbolic values, we extract the regions from this value and index of the element accessed, we also get the size of the array using getSizeInElements, after this, there are simple checks on whether the access is legal or not.

BoolAssignmentChecker

BoolAssignmentChecker is a simple check which checks for assignment of non-Boolean values to Boolean variables. That is, a variable of type bool should be assigned only boolean values.

Implementation Details

The main function for this check is BoolAssignmentChecker::checkBind, it first checks if the store is into a boolean if not returns straightaway. After this we get the R-Value and check for the two conditions that the value must not be less than 0 or greater than 1.

PointerSubChecker

PointerSubChecker is a check which makes sure that if there are any pointer subtractions in the source code, they must not belong to different memory chunks. Since subtraction in the same chunk with knowledge of the data type can allow us to make some inference but for different chunks the allocation is the responsibility of the OS and is neither deterministic nor meaningful.

Implementation Details

PointerSubChecker::checkPreStmt is the main function for this check, in this check we first check if the binary operator for the expression is subtraction following that we get

the memory regions for the two pointers, we check if they belong to the same base memory region, if not warning is issued.

ReturnUndefChecker

ReturnUndefCheckeris a path-sensitive check which looks for undefined or garbage values being returned to the caller. Essentially the motive of this check is to look for instances when the value returned by a function is garbage or undefined value and if found try to avoid them.

Implementation Details

ReturnUndefChecker::checkPreStmt is the driver function in this check, we get the retval from the arguments to the function if the function is not a void type and we have an undef or null return we issue warning. We also have a function checkReference which checks the pointer values which might be returned and if they are null it emits the bug.

ConversionChecker

ConversionChecker is a path sensitive check which checks that there is no loss of sign/precision in assignments, comparisons and multiplications. The following cases are covered:

1) A negative value is implicitly converted to an unsigned value in an assignment, comparison or multiplication.
2) Assignment or initialization when the source value is greater than the max value of the type that is being assigned to.
3) Assignment or Initialization when the source integer is above the range where the target floating-point type can represent all integers.

The third one of the above points is specific to floating-point numbers since above a certain value they lose the capability to represent all the numbers in a range.

Implementation Details

ConversionChecker::checkPreStmt is the driver function which checks for two values initially, (i) the type of statement/expression, (ii) the type of operation, once we have this information, based on the values we use either isLossOfSign or isLossOfPrecision and if either return true we issue a warning.

References

- https://www.telgian.com/what-is-code-compliance/
- https://en.wikipedia.org/wiki/Code_enforcement
- https://www.multidots.com/importance-of-code-quality-and-coding-standard-in-software-development/
- https://www.ni.com/en-in/innovations/white-papers/11/what-is-the-iso-26262-functional-safety-standard-.html
- https://en.wikipedia.org/wiki/IEC_61508
- https://en.wikipedia.org/wiki/IEC_62304
- https://en.wikipedia.org/wiki/MISRA_C
- https://en.wikipedia.org/wiki/CERT_C_Coding_Standard
- https://en.wikipedia.org/wiki/DO-178C
- https://en.wikipedia.org/wiki/DO-178B
- https://www.grammatech.com/software-assurance/certifications-compliance/do-178b
- https://drive.google.com/file/d/1mRWXC88e_2bEDwNoAZ1m-b1CWPJBevyv/view