STATIC ANALYSIS OF A C PROGRAM

USING FRAMA-C

Adishree Annabathula

Student, Department of Computer Science & Engineering

The University of Texas at Arlington

Arlington, Texas, USA

E-mail: adishree.annabathula@mavs.uta.edu

*Abstract* — To use an automated software engineering tool like Frama-C to analyze a program or system.

Keywords—static analysis; Frama-C; ACSL.

# Introduction

Static analysis is the [analysis of computer software](https://en.wikipedia.org/wiki/Program_analysis_(computer_science)) that is performed without actually executing the programs.[[1]](https://en.wikipedia.org/wiki/Static_program_analysis#cite_note-1) In most cases, the analysis is performed by an [automated tool](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis) on some version of the [source code](https://en.wikipedia.org/wiki/Source_code). It can also be used to find bugs or to ensure conformance to coding guidelines.

# Modelling framework used

## Some tools for the static analysis of C programs

* [Astrée](https://en.wikipedia.org/wiki/Astr%C3%A9e_(static_analysis)) – finds all potential [runtime errors](https://en.wikipedia.org/wiki/Runtime_errors) by [abstract interpretation](https://en.wikipedia.org/wiki/Abstract_interpretation), can prove the absence of runtime errors and can prove functional assertions; tailored towards safety-critical C code (e.g. avionics).
* [cpplint](https://en.wikipedia.org/wiki/Cpplint) – An open-source tool that checks for compliance with Google's style guide for C++ coding.
* [Clang](https://en.wikipedia.org/wiki/Clang) – An open-source compiler that includes a static analyzer ([Clang Static Analyzer](http://clang-analyzer.llvm.org/)).
* [Coccinelle](https://en.wikipedia.org/wiki/Coccinelle_(software)) – An open-source source code pattern matching and transformation.
* [Cppdepend](https://en.wikipedia.org/wiki/Cppdepend) – Simplifies managing a complex C/C++ code base by analyzing and visualizing code dependencies, by defining design rules, by doing impact analysis, and comparing different versions of the code.
* [ECLAIR](https://en.wikipedia.org/wiki/ECLAIR) – A platform for the automatic analysis, verification, testing and transformation of C and C++ programs.
* [Eclipse (software)](https://en.wikipedia.org/wiki/Eclipse_(software)) – An open-source IDE that includes a static code analyzer ([CODAN](http://wiki.eclipse.org/CDT/designs/StaticAnalysis)).
* [Frama-C](https://en.wikipedia.org/wiki/Frama-C) – An open-source static analysis framework for C.
* [Goanna](https://en.wikipedia.org/wiki/Red_Lizard_Software#Products) – A software analysis tool for C/C++.
* [Lint](https://en.wikipedia.org/wiki/Lint_(software)) – The original static code analyzer for C.
* [Polyspace](https://en.wikipedia.org/wiki/Polyspace) – Uses [abstract interpretation](https://en.wikipedia.org/wiki/Abstract_interpretation) to detect and prove the absence of [run time errors](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)), Dead Code in [source code](https://en.wikipedia.org/wiki/Source_code) as well as used to check all MISRA (2004, 2012) rules (directives, nondirectives).
* [Sparse](https://en.wikipedia.org/wiki/Sparse) – An open-source tool designed to find faults in the [Linux](https://en.wikipedia.org/wiki/Linux) kernel.
* [Splint](https://en.wikipedia.org/wiki/Splint_(programming_tool)) – An open-source evolved version of Lint, for C.

## Modelling framework used

Frama-C has been used in the current project to perform static analysis. It stands for Framework for Modular Analysis of C programs. It relies on CIL (C Intermediate Language) to generate an abstract syntax tree which is written in ACSL (ANSI C Specifications Language). It uses SMT solvers and theorem provers.

## ACSL

The ANSI/ISO C Specification Language (ACSL) is a behavioral specification language for C programs. Specifications are given as annotations in comments written directly in C source files, so that the source files remain compilable. Those comments must start by ‘/\*@’ and end with ‘\*/’. Identifiers may start with the backslash character ‘\’. The keyword ‘requires’ starts a precondition and the keyword ‘ensures’ starts a postcondition. Annotations may be either assertions or loop annotations. Assertions are allowed before any C statement or at end of blocks and start with the keyword ‘assert’. Loop annotations can be either invariants, assigns clauses or variants, and allowed before any loop statements. Each of these clauses must be terminated with a semicolon ‘;’. [2]

# source code overview

An Armstrong number is an n-digit number that is equal to the sum of the nth powers of its digits. The program chosen to perform static analysis on generates a sequence of Armstrong numbers consisting of 3 digits. It lets the user to input the lower and upper limits.

## Code in C language

#include<stdio.h>

void main()

{

long int n1, n2; /\* intervals \*/

long int i, temp, num, rem;

printf("Enter two intervals: ");

scanf("%ld %ld", &n1, &n2);

printf("Armstrong numbers between %ld and %ld are: \n", n1, n2);

for(i=n1; i<=n2; ++i)

{

temp=i;

num=0;

while(temp!=0)

{

rem=(temp%10);

num+=rem\*rem\*rem;

temp/=10;

}

if(i==num)

printf("%ld\t ",i);

}

printf("\n");

}[3]

## Adding specifications to the code

#include <stdio.h>

void main()

{

long int n1, n2; /\* intervals \*/

long int i, temp, num, rem;

printf("%d", 6/0);

printf("Enter two intervals: ");

scanf("%ld %ld", &n1, &n2);

printf("Armstrong numbers between %ld and %ld are: \n", n1, n2);

/\*@ ensures n1 > 0;

ensures n2 > 0;

\*/

/\*@ loop invariant n1 <= i <= n2;

\*/

for(i=n1; i<=n2; ++i)

{

/\*@ assigns temp, num;

\*/

temp=i;

num=0;

while(temp!=0)

{

/\*@ assigns rem, num, temp;

\*/

rem=(temp%10);

num+=rem\*rem\*rem;

temp/=10;

}

if(i==num)

printf("%ld\t ",i);

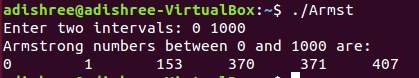
}

printf("\n");

}



## Output



# analysis of program

Frama-C is organized with a plug-in architecture. Different plug-ins implement different analysis methods. Frama-C is extensible. It contains several ready-to-use plug-ins for the static analysis of C code, but more importantly, any new plug-in may use the results or functionalities provided by the existing plug-ins.[4]

Some of the plug-ins are:

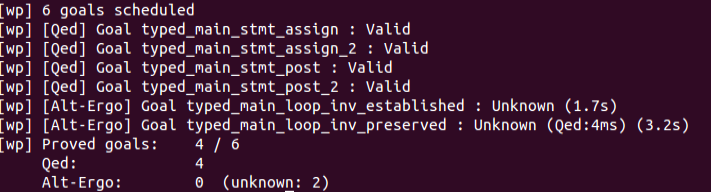
* [Value analysis](http://frama-c.com/value.html), computes variation domains for variables. It is quite automatic, although the user may guide the analysis in places. It handles a wide spectrum of C constructs. This plug-in uses abstract interpretation techniques.
* [Jessie](http://frama-c.com/jessie.html) and [wp](http://frama-c.com/wp.html), two deductive verification plug-ins based on weakest precondition computation techniques. They allow to prove that C functions satisfy their specification as expressed in [ACSL](http://frama-c.com/acsl.html). These proofs are modular: the specifications of the called functions are used to establish the proof without looking at their code.

## Plug-in wp

The plug-in ‘wp’ (which stands for “weakest precondition”) uses Hoare Logic to compute the weakest precondition of a function (the weakest condition under which the function is correct). This weakest precondition is computed as a formula in some logic. Finally, the ‘wp’ plug-in asks a Theorem Prover if the weakest precondition is implied by the actual precondition of the function.

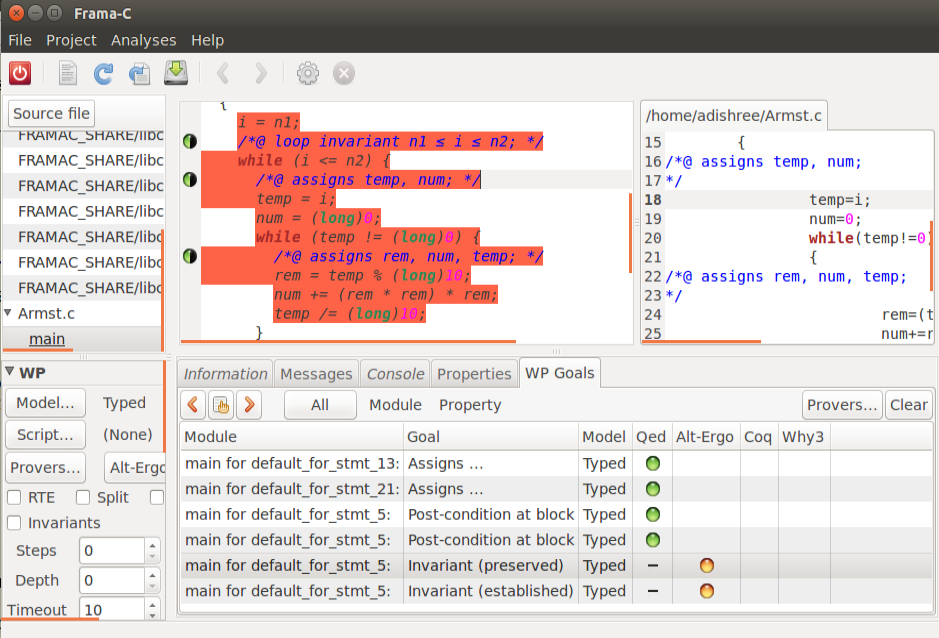
The command used to run the wp plug-in is:

frama-c –wp Armst.c



Frama-c can also be run in the gui mode.

frama-c-gui –wp Armst.c



## Plug-in rte

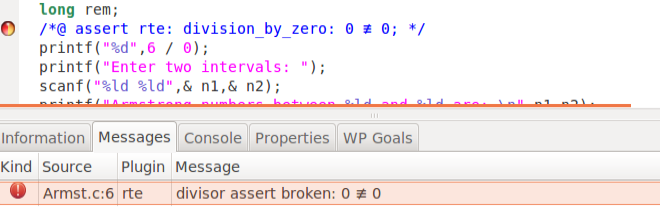
The aim of the rte plug-in is to automatically generate annotations for:[5]

* Common runtime errors, such as division by zero, signed integer overflow or invalid memory access
* Unsigned integer overflows
* Precondition checking (requires and assumes clauses) at function’s call sites
* Postcondition checking (ensures clauses) and assigns, for functions having an ACSL specification

Just to show how the plug-in works, I added a statement in the C program to generate a run time error.

The command used to run the rte plug-in is:

frama-c –rte Armst.c



# conclusion

The input language influences the depth of the static analysis that can be undertaken easily. Languages that are essentially dynamic, like C, are more difficult to analyze than languages, like Ada, that include strong typing and range constraints. Hence, the nature of the input language needs to be taken into account in the specification of the static analysis to be undertaken.[6]

##### Acknowledgment

I would like to express my gratitude to my professor, Taylor Johnson, for his constant support and encouragement, and without whose help and counsel, the project would have been immeasurably more difficult.

##### References

1. Wichmann, B. A.; Canning, A. A.; Clutterbuck, D. L.; Winsbarrow, L. A.; Ward, N. J.; Marsh, D. W. R. (Mar 1995). ["Industrial Perspective on Static Analysis."](https://web.archive.org/web/20110927010304/http:/www.ida.liu.se/~TDDC90/papers/industrial95.pdf). Software Engineering Journal: 69–75. Archived from [the original](http://www.ida.liu.se/~TDDC90/papers/industrial95.pdf) on 2011-09-27.
2. http://frama-c.com/download/acsl.pdf
3. <http://www.programiz.com/c-programming/examples/armstrong-number-interval>
4. Frama-C: A Quick Start Guide Robert K¨onighofer, 06.12.2013.
5. http://frama-c.com/download/rte-manual-Neon-20140301.pdf
6. Industrial perspective on static analysis by BA Wichmann, AA. Canning, D.L. Clutterbuck, LA Winsborrow, N.J. Ward and D.W.R. Marsh