School of Innovation, Design and Engineering

Module 4: Static Program Analysis

(Revised: 2017-04-19)

This document provides the laboratory instructions for the part of Module 4 that deals with static program analysis. The objective of the lab is to give hands-on experience with the state-of-the-art tool **Astrée** for static program analysis. Astrée performs a value analysis, and can issue warnings for a number of value-related run-time errors based on the results of this analysis. Since Astrée's analysis is sound, the absence of warnings guarantees that the code is free from those defects.

ASTRÉE HANDS-ON EXERCISE

This exercise will take place at MDH during one of the campus days. You will use our lab equipment, Dell Inspiron laptops, where we have preinstalled Astrée. If you want to get started before the campus day, or if you are curious in general how Astrée works, then you can also obtain an evaluation license and install the tool on your own computer. see Section 7 below.

1. Starting Astrée

Astrée comes bundled with the "RuleChecker" tool: the combined tool is called "a³ for C". RuleChecker performs various syntactic checks for things like compliance with coding standards: we will not consider this tool any further in this exercise.

A³ for C has a server-client architecture, with the server doing the analyses and the client taking care of the user interaction, analysis configuration, etc. We have set up the client on each lab machine, and we have set up a single server **stl.idt.mdh.se.** With this setup, a³ for C is launched in the following way:

- Launch a³ for C. (Under Windows, use the "Astrée" launcher; under linux, run "`alauncher`" from the command line.)
- Select the "a³" tab, and click the "a³ for C" entry.
- A welcome window will appear, as well as a popup window "Connect to server". If the server is already configured then "Host" is set to "stl.idt.mdh.se", and "port" to "36000". If not, then enter these values.
- Then select your own, unique Username and Password. Do **not** go with the default "anonymous" user name: we have disabled this entry to prevent that all your projects will end up in the same place. Do not forget your login credentials, once you have created them!
- Then click "OK". The client will then connect to the server, and a new popup window with title "Open project" should appear. It should be empty at this stage: click "cancel" to make it go away.

2. Exploring the features of Astrée

Visiting Address: Drottninggatan 16 A
Postal Address: P.O. Box 325, SE-631 05 Eskilstuna, Sweden
Phone: +46 16 153 600

Visiting Address: Högskoleplan 2
Postal Address: P.O. Box 883, SE-721 23 Västerås, Sweden
Phone: +46 21 101 300

Web: www.mdh.se/idt
E-mail: {name.surname}@mdh.se

Let us now turn to the welcome window. Click "View examples" under "Projects" (to the left in the welcome screen). You will get a list of predefined example projects, for Astrée to the left and RuleChecker to the right. These projects are designed to highlight different features of the analysis tools. We will now use some of these example projects to show how Astrée operates, as well as some of its features.

Open the "Scenarios" sample project. The toolbar will extend, a menu will appear to the left, and some green buttons will appear at bottom. Now let's explore the menu to the left. The different entries are used to control the operation of Astrée.

At the top is the **project name**. By right-clicking it, you can change some project settings. In particular you can set the project to be **private**, which means that it will not be visible to other users. Later, when you create your own projects, you should make them private such that no other students can access them.

Configuration: here, the different settings controlling the analysis can be changed. The most interesting part is the “Analyzer”. If you click it, the main window will contain a number of options that can be set to control the analysis. Now double-click on the “Astrée” heading. The subheadings “Semantics”, “Precision”, and “Output” will appear, each with a number of options to set. The “Semantics” options gives fine details how the actual C implementation works (necessary, since the C standard gives room for different semantics). The “Precision” options are used to fine-tune the analysis to find the best trade-off between precision and analysis time: we will return to these options later. The “Output” options, finally, affect what to report from the analysis and in which form.

Results: this heading collects means to view and report on analysis results. Analysis results are also reported, in various ways, in other places: an analysis summary is shown in the lower left corner, and the buttons at the bottom of the window will display various analysis results when clicked.

Files: this is a file tree containing the C files in the project that have been preprocessed, and are ready to analyze. Clicking on a file will show it in the main window.

3. Analyzing a first predefined example

Double-click "scenarios.c" under “Files” (Preprocessed). you will now see the code to be analyzed. To the right you have the original C code in the file, and to the left the C code that results after running the C preprocessor on the original code. It is the latter code that is being analysed by Astrée. As you can see, this code also contains some Astrée directives that start with the string "`__ASTREE_`": `__ASTREE_assert` tells Astrée to check a certain fact, `__ASTREE_log_vars` will show the calculated constraints for the values of the listed program variables in the chosen program point, `__ASTREE_volatile_input` will mark a variable as volatile, and `__ASTREE_unroll` will cause Astrée to analyze a certain number of loop iterations separately from each other thus potentially increasing the precision of the analysis.

Press the "play" button in the toolbar to analyze the example program. This will cause all the code that is reachable from the selected entry point to be analysed. Look what happens. In the lower left, the "traffic light" will turn to red. This means that the analysis has detected some potential serious errors in the code.

Now browse through the tabs for the error reporting, to the lower right. They provide different views. In particular, the "Output" tab gives the full, raw output from the analysis. Scroll through it until you find the lines starting with “ALARM”. Some of them contain the text “M2012.8-*.*-required”: these alarms are issued by RuleChecker and indicate that some MISRA 2012 coding rules were violated (MISRA is a coding standard for C code in safety-critical applications). Here, we are more interested in the other alarms: they are all issued by Astrée. Most of the alarms point out possible problems, which may be false positives, but the alarms that are followed by lines in light red starting with “ERROR” are problems that for sure are present. If you click on the error message (or scroll the left code window), then you will find the erring statements highlighted in red. Try to understand the alarms: what has caused them? Are they real, or false positives? Can you spot the bugs in the code causing the alarms?

4. Analysing further predefined examples

Let's check out another example. Click "Welcome" under "Example 1: scenarios" to the upper left to get back the sample projects menu, and select the "Dhrystone" example. Analyse it. This is an example of a project that consists of several source files, and it is intended to show the capability of Astrée to make a precise analysis also in situations where there are complex calling relations between different functions. Scroll through the text displayed under the Output tab and inspect the erring statements in the correct files. Also have a look at the "Not reached" tab: now, you will see that for some files the coverage is not 100%. This means that some code is unreachable, possibly due to a fatal runtime error that will cause an interrupt. In the preprocessed code window, the unreachable code will be grey.

If you wish, you may also analyze the remaining example projects.

5. Setting up and analyzing simple projects

Now start a new project by selecting "New" from the "Project" menu. This will start the "New Project" wizard. Set the project type to “Astrée”. Add the source file unroll.c (not preprocessed) that is available from the course web page.Then go through the steps to set up the project: just click "Next" (and “Finish”) to allow the default configuration for the analysis. When the wizard is done, you should set the starting point for the analysis to be “main” (Analyzer → Analysis entry → double-click “value” and enter “main”). Then preprocess the file (select Preprocessor, press the "Preprocess" button to the lower right, then press "Import files", then “OK” in the popup window). Also don't forget to make the project private through right-clicking the project name in the menu.

Analyze the example. If the project is set up correctly then the green traffic light will be lit, indicating zero detected runtime errors (for this program: no over/underflows, or divisions by zero). Contrary to dynamic analysis and testing, since Astrée's analysis is designed to be safe, *this can be trusted*. Thus, there is *no need to further test this program for these errors*.

The main part of this program is a loop. Astrée succeeded in proving the absence of runtime errors since it, by default, analyzes a large number of loop iterations separately. To disable this setting for this loop, insert the directive `__ASTREE_unroll((0))` immediately before the loop in the left code window. Then run the analysis again. (If a window pops up asking to save files, just click “save”.) What is the result, and why to you think that it differs from the first result?

The next example to analyze is insertsort.c (not preprocessed), which is available from the course web page. The program applies the well-known insertsort sorting algorithm to sort an array with eleven numbers. It contains a planted bug. Can you spot it? Spend a few moments inspecting the code to see if you can find it. (If you have problems to spot the bug, then ask the instructor.) Analyse the code, and study the report. Not surprisingly you will have a cascade of errors. Some are real, some might be false positives. (It might be helpful to check the tab “Findings”, which provides a more structured view of the issues found.)

Correct the code, and analyse it anew. (You can edit directly in the window for the preprocessed code. Click OK in the popup window when running the analysis.)

As you see, there are still quite some errors being reported. They are actually false positives, and are due to that the analysis over-approximates the possible values of the loop variable j. This can be remedied by setting the "loop unrolling" limits to high values enabling both the inner and outer loops to be semantically unrolled during the analysis. Select Analyzer → Astrée → Precision → Loops and set "Maximum number of outer (inner) loop unrollings" (two settings) to 10. Analyse anew both the erroneous and the corrected versions of the program. Notice the results! Did you succeed to prove the absence of runtime errors for the corrected version?

6. Analyzing own code (optional)

If you have some suitable C code of your own, then load it onto the machine, set up a new project in Astrée, import the code into the project, and analyse it! Start with the standard settings for the analysis. If some bugs are reported, then increase the precision of the analysis to see if it is a false positive that will go away.

7. Continuing on your own

If you want to try out Astrée further on your own, then you can request a time-limited free evaluation license from AbsInt GmbH at http://www.absint.com/astree/contact.htm.

8. Assignment

The assignment for this lab is to write a report that gives an account for your work and your findings in Sections 3 - 7 above. Describe how you analysed the code, which real errors in the code that you found, how you tuned the analysis in order to get rid of false positives, and whether or not you ended up with a piece of code with zero alarms (and thus provably free from the run-time errors that Astrée can detect). If you analysed some own code, then it is very interesting to know whether you managed to uncover any previously unknown bug or weakness in the code. Send your report (pdf format) to bjorn.lisper@mdh.se.