School of Innovation, Design and Engineering

Module 4: Static Program Analysis

(Revised: 2016-05-23)

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.

ASTREE 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. Alternatively you can download the client and install according to the instructions that we have sent out by mail. You can then use the client to connect to the Astree server that we have set up, see below.

1. Starting Astrée

Astrée 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, Astrée is launched in the following way:

- Launch the Astrée Server Controller.
- Select “Astrée for C” to the left.
- In the popup window “Connect to server”, enter the host name (stl.idt.mdh.se), and use the default port (36000) and Username (anonymous).

The client will then connect to the server. A window displaying available projects will pop up: close that window. You will now have some menus, and a toolbar, to the upper left, and a welcome screen to the right. To the lower left you will have a (still empty) analysis summary, and to the lower right a (likewise empty) area with different tabs showing different aspects of the analysis results.

2. Exploring the features of Astrée

Click on "view examples" under "Projects" (to the left in the welcome screen). This will open a menu with a number of example Astrée projects, designed to highlight different features of the analyser. 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 an "Analysis entry" screen (the "Analysis entry") will appear to the right. This screen tells that the analysis will use the "main" function as starting point, plus some user-provided directives (under "Wrappers and stubs") telling the analysis that some program variables are volatile (can be changed by activities outside the program, like a concurrently executing thread), and that their values always will fall within certain ranges.

Now let's explore the menu to the left. The different entries are used to control the operation of Astrée.

Local settings: for this session, we will skip it.

Analysis configuration: here, the different settings controlling the analysis can be changed. Open the "settings" item, and browse through the different groups:

- The "Semantics" group contains settings that affect the assumed semantics of the C program, like how variables are initialized.
- The "Precision" group controls settings affecting the trade-off between analysis time (and memory consumption) and precision. Of particular interest is the "Domains" setting: by default all abstract domains are turned on, which yields high precision but potentially long analysis times. For larger codes it might be beneficial to first turn off some of the more complex domains, and gradually re-introduce them only if needed to remove suspected false positives.
- The "Output" settings control what to report, and the level of detail.
- The “Rule checker” section is used to enable the checking of different coding rules (like “MISRA C” for safety-critical automotive applications).

Files: this is a file tree containing the C files in the project that have been preprocessed, and are ready to analyze.

3. Analyzing a first predefined example

Click "scenarios.c" under “Files”. 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` 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”, and the red texts indicating “definite runtime errors”. These refer to alarms or errors found at lines 73, 80, 81, 84, 85, and 127 in the preprocessed C code, and the printout also gives information about the contexts for which the errors will occur. If you 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 "Unroll" example. Analyse it. Now 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?

Now open the "dhrystone" project, and 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. Again, scroll through the text displayed under the Output tab and inspect the erring statements in the correct files. Also have a look at the "Summary/F" 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 a simple project

Now start a new project by selecting "New" from the "Project" menu. This will start the "New Project" wizard. Add the source file insertsort.c (not preprocessed) that is available from the course web page: it can also be found locally under `C:\Course\code-examples`. Then go through the steps to set up the project: set “analysis start” to “`main`”, and then just click "Next" (and “Finish”) to allow the default configuration for the analysis. When the wizard is done, preprocess the file (press the "Preprocess" button to the lower right, then press "Import files", then “OK” in the popup window).

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.

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 Settings -> Precision 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 6 and 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.