Software-Maintenance: Assignment 2

Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa, Dipl.-Ing. Oliver A. Tazl,
Alessa Angerschmid, Laura Rössl, Clemens Walluschek
Institute for Software Technology, Graz University of Technology

March 27, 2020

1 Organization

You can achieve a total of 20 points for Assignment 2. The assignment consists of a theoretical (Section 2) and a programming part (Section 3), which count 50% each.

1.1 Tutorial

There will be a tutorial for the programming assignment. In this tutorial the framework will be explained. Additionally, we will provide some hints about your programming task (useful data structures, what to test, what to keep in mind etc.). Join the discord channel and ask your questions.

1.2 Question Hour

We will also provide a question hour on demand. This means, the question hour will only take place if there are questions in the newsgroup beforehand. We will announce this in the newsgroup, too. There will be a WebEx question hour, if necessary.

1.3 Framework

You can download a framework from the TeachCenter. It contains a LaTeX template for the theoretical part and a folder with source files for the programming assignment.

1.4 Submission

The submission consists of three parts: GIT repository, issue reports on `GitLab` and a digital copy of the theoretical part. For all three parts the deadline is **Monday, April** 20^{th} **2020, 10 am**. Late submissions will **not** be accepted and unreadable submissions will receive 0 points.

1.4.1 Programming Assignment via GitLab

For the duration of the assignments, you need a `GitLab`[1] account. Your repo is already created, so just reuse this one.
During the development of your program you must use `GitLab`'s issue tracking system to report software bugs[2]. Recording issues allows you and your team to maintain a high software quality and capture, report and manage bugs. For our purposes you must define a label named "BUG" (or "bug") and add issue reports whenever you find problems in the current version of your implementation. Adding this label is essential so that we can isolate the bug reports easily. While your group can decide whether they would like to assign bugs, define due dates etc. selecting the correct label is mandatory. Do not forget to close issues, once you have fixed them. You will be asked to show some of the issues you have created at the mandatory interviews. Appendix A contains a basic introduction into bug reporting and how to add them in `GitLab` .

[1] `https://git-students.ist.tugraz.at/`

[2] Of course you may also use the issue tracking system for additional purposes, e.g., backlog. In case you do think about appropriate labels.

Repository Structure

In order to allow automated testing, the following repository structure is obligatory for the folder `submission2/`:

- `group-XX-ass2.pdf` your solution of the theoretical part
- `readme.txt` description of your practical assignment
- `ProgramSlicer/` folder with the programming assignment
  - `build.xml` the ant-build file for your project
  - `src/` source code in the provided package-structure
  - `testdata/` data for the test cases
  - `lib/cfgGenerator.jar` jar based on assignment 1

Do **not** upload any other files or folders to the master branch of our repository!

Testing

Please note that failing the public test cases (public.java, public1.java, public2.java) will lead to zero points for this part of the assignment. Additionally, we use private test cases to measure the performance of your submission. The scoring of the implementation will be made on the basis of the passing of test cases which will cover different parts of the grammar.
For testing your application we use Java 8.
Use the `readme.txt` to describe which features you have implemented as well as missing features.

1.4.2 Digital copy

Hand in a digital copy of your solution of the theoretical part (Section 2) in the git repo as described above. For that, we provide the LaTeX-file `assignment2.tex` in the framework which contains a cover sheet that you must use for your submission as well as a template of how we would like to receive your solutions. Hand written documents will be accepted if the hand writing is in a readable form. However, unreadable submissions will receive 0 points! You can either write in German or English.

1.5 Interviews

Interviews will be held on **April** 23^{th} and 24^{th} **2020**, if possible. Watch the Teach Center for timeslots. All team members need to be able to explain your theoretical solutions as well as your source code. You should also be able to describe the bugs which you have inserted into the bug database. In addition, you should be prepared to solve small examples by hand.

1.6 Plagiarism

Discussion of the tasks between the lecture participants is welcome, but plagiarism will lead to a score of 0 points. Do not include any personal information (name, student ID, mail-addresses etc.) in your source code. Your files will be uploaded to an external server in order to cross-check against other submissions.

If you have questions regarding the assignment you should post them to the newsgroup (`tu-graz.lv.soma`).
Good luck!

2 Theoretical Part

2.1 Flow Propagation Algorithm *[5 points]*

You may create graphs with a tool of your choice. You have to use the provided template for your hand in, otherwise your points will be reduced.
read(a) ... The variable a is initialised from the input stream.
write(a) ... The variable a is passed to the output stream.

```
1  begin;
2    read(a);
3    b = 5;
4    c = a + 2;
5    d = a - 1;
6    while (c > a) do
7     begin
8       c = c - 1;
9       while (b < c) do
10       begin
11         d = c - a;
12       end;
13      od;
14      if (a == b) then
15       begin
16         b = (a - c) * 2;
17         d = 1;
18       end;
19     else
20      begin
21        b = 2;
22      end;
23      fi;
24      a = b + 1;
25    end;
26   od;
27   res = a + d;
28   write(res);
29 end;
```

a. Calculate the backward slice for **(11,** $\{\mathbf{d}\}$**)**. [*2.5 points*]

b. Calculate the forward slice for **statement 3**. [*2.5 points*]

2.2 Delta Debugging *[5 points]*

In the following example you have to minimize inputs (c_f), using different versions of Delta Debugging. For each task use the algorithm from the script (displayed above each task).

Delta Debugging (see script) is based on the "divide and conquer" principle. It works with the differences (deltas). These deltas are substrings of the input sequence (c_f), with the following characteristics:

- $\Delta_1 \cap \Delta_2 = \emptyset$
- $\Delta_1 \cup \Delta_2 = c_f$
- $|\Delta_1| \approx |\Delta_2|$

We strengthen this definition by the following restriction:
$|\Delta_i| = |\Delta_{i+1}| + \epsilon$ with $\epsilon \leq 1$

For your submission you have to use the LaTeX-template we provided, otherwise you will get point

reductions.
Do not forget to identify each rule that could be used!

1. **The minimizing delta debugging algorithm (version 3 in the script (page 68))**

$$ddmin(c_f) = ddmin2(c_f, 2)$$

$$ddmin2(c_f, n) = \begin{cases} ddmin2(\Delta_i, 2) & \text{if } test(\Delta_i) = FAIL(\text{reduce to subset}) \\ ddmin2(\nabla_i, max(n-1, 2)) & \text{else if } test(\nabla_i) = FAIL(\text{reduce to complement}) \\ ddmin2(c_f, min(|c_f|, 2n)) & \text{else if } n < |c_f|(\text{increase granularity}) \\ c_f & \text{otherwise(done)} \end{cases}$$

   (a) We have "`0123456789ABCDEFGHIJ`" as an input string. The inputs `1`, `C` and `H` cannot be applied without each other. Input `D` reveals a bug. [*2 points*]

2. **The isolation differences algorithm**

$$dd(c_s, c_f) = dd2(c_s, c_f, 2)$$

$$dd2(c'_s, c'_f, n) = \begin{cases} dd2(c'_s, c'_s \cup \Delta_i, 2) & \text{if } \exists\{i|test(c'_s \cup \Delta_i) = FAIL\} \\ dd2(c'_f \backslash \Delta_i, c'_f, 2) & \text{else if } \exists\{i|test(c'_f \backslash \Delta_i) = PASS\} \\ dd2(c'_s \cup \Delta_i, c'_f, max(n-1, 2)) & \text{else if } \exists\{i|test(c'_s \cup \Delta_i) = PASS\} \\ dd2(c'_s, c'_f \backslash \Delta_i, max(n-1, 2)) & \text{else if } \exists\{i|test(c'_f \backslash \Delta_i) = FAIL\} \\ dd2(c'_s, c'_f, min(2n, |\Delta|)) & \text{else if } n < |\Delta| \\ (c'_s, c'_f) & \text{otherwise} \end{cases}$$

   (a) Input : "`0123456789ABCDEFGHIJ`". In order to obtain a result unlike UNRES the program requires all three `0`, `8` and `E`. The program crashes if the input string contains `D`. [*3 points*]

3 Programming Assignment

3.1 Objectives

The aim of this programming assignment is to take the theories presented in the lecture and produce a practical application. Therefore you should implement Backward and Forward Slicing in **Java** *[10 points]*. Additionally, you must add at least two issue reports (see Section 1.4.1 and Appendix A) and be able to show them during the interviews. You will not receive points for that, however, we will deduce the points of your programming part by a maximum of 1.0 point if you do not fulfill this requirement! The programming task counts 50 % of the second assignment.

3.2 General Overview

The framework is based on the framework of Assignment 1. Therefore it reads a Java program from a file and already produces a CFG (see Section 3.4). Printing the calculated results into files is also handled by the framework in the class `IOHandler.java`, you are **NOT** allowed to change these output functions as they will be used for the automated testing.

3.3 Your Tasks

3.3.1 Implement Backward Slicing

`generateBackwardSlice` in *src/at/tugraz/ist/soma/FPA/FPAlgorithms.java*

Here you already gained the slicing criteria `slicingCriteria` and the CFG `cfg` as implemented in Assignment 1 with some small adjustments (see Section 3.4), now it is your task to calculate the backward slice as explained in the tutorial slides and the lecture script. Therefore you should fill the datastructure `bwdSlicingTable` by inserting all the nodes and calculating the corresponding sets (def, ref, gen, kill, in, out, inSlice). At the end the calculated table is handed over to the datapool.

3.3.2 Calculate the Forward Slice Criteria

`calculateForwardSlicingCriteria` in *src/at/tugraz/ist/soma/FPA/FPAlgorithms.java*

With the backward slicing tables calculated before it is now your task to select for each table the corresponding slice. Given these slices you have to select the criteria for the forward slicing. The forward slicing criteria is defined as the following:

- The Line Number that occurs in most of the backward slices.
- If two lines occur the same amount of times, choose the lower line number.

At last the chosen line number is now handed over to the datapool.

3.3.3 Implement Forward Slicing

`generateForwardSlice` in *src/at/tugraz/ist/soma/FPA/FPAlgorithms.java*

In this subtask you have to implement forward slicing as discussed in the tutorial and in the lecture. We already have the slicing criteria `fwdSlicingStmt` and the CFG `cfg` given. Now you have to fill in the datastructure `fwdSlicingTable` with all the corresponding sets (ref, def, gen, kill, in, out, inSlice). After that this table is again handed over to the datapool.

3.4 CFG Extension

The CFG is based on the one you already implemented in Assignment 1. Due to different implementations our `CFGNode` from the CFG contains the following two hopefully helpful methods:

- `ArrayList<CFGNode> getPredecessors()`
  Returns all the predecessors of that node

- `ArrayList<CFGNode> getSuccessors()`
  Returns all the successors of that node

For example:

```
1  ...
2  int a = 1; // predecessors = {}, successors = {3}
3  while (a < 5) { // predecessors = {2}, successors = {4,6}
4    a = a + 1; // predecessors = {3}, successors = {3}
5  }
6  int b = a + 1; // predecessors = {3}, successors = {}
7  ...
```

3.5 Compilation and Execution

You have to build an executable .jar file `ProgramSlicer_<GroupNr>.jar` from your source code. (Do not forget to change the group number in the build file!) Building the executable can be done using the command: `ant jar`

Your program must be able to be executed from the command line/shell with the command:
`java -jar ProgramSlicer_<GroupNr>[.jar] <input-file> <slicing criterias>`

resp. for further more detailed testing with the command:
`java -jar ProgramSlicer_<GroupNr>[.jar] <input-file> <slicing criterias> deeptest`

If you add the 'deeptest' keyword your implementation will output the calculation tables for the backward and forward slice as well, so that we can check if your tables are correct too. So please make sure that you output on the console only if the keyword 'debug' is given (you can check this via `DataPoolExtended.getInstance().getDebug()`).

The slicing criteria is a string consisting of one or more pairs (i, V), where i is the line number in the input program and V is the set of variables of interest. These slicing criterias will be used for the backward slicing from which you will calculate the forward slicing criteria.

An example program call for group no. 1 could look the following:

- `java -jar ProgramSlicer_1.jar testdata/input/test1.java "(12,{z})"`
- `java -jar ProgramSlicer_1.jar testdata/input/test1.java "(12,{z});(13,{x})"`

3.6 Testing

If you want to test your program you can add the 'debug' keyword and then the framework will output the CFG in a dot file (like in Assignment 1) and all the calculated Forward and Backward Slicing tables into an html file. This is done in the class `FPAVisualizer`, feel free to add any information you want, as this outputs will not be tested. For example:
`java -jar ProgramSlicer_1.jar testdata/input/test1.java "(12,{z})" debug`

Please use the `IOHandler` class and the `FPANode` class' `getString` methods for outputting your final results (this is already done by the framework if you haven't changed it). These classes and methods are used for our automated testcases, so make sure that your calculated output meets the output specifications.

3.7 Input Specification

- You can assume that all input programs meet a valid Java syntax and that the programs consist of only one file. This file contains a main, but no other methods.
- You can assume the given CFG and all its nodes as correct.

- You can assume that like in Assignment 1 there will be only one statement per line.
- You can assume that there are no methods calls, arrays, lists or other non-primitive datatypes used.

A Bug Reporting

A.1 General Notes

There are three key points to bear in mind when creating a bug report for a team of developers [1]:

- **Show me (how to show myself)**: How exactly can the bug be reproduced (be as precise as possible)?
- **Tell me exactly what you saw**: What has actually happened?
- **Tell me why you think what you saw is wrong**. What would you expect to happen instead?

Before reporting any issue make sure that the bug is not already known and that you are using the latest version/commit. In case there are multiple problems, report each issue separately. Some general advice to follow when writing your report [1,2]:

- include a short and concise title
- be specific
- give rather more information than less (however stick to the relevant parts)
- use precise wording
- if suitable provide good attachments (screen captures/log files/etc.)
- reread what you wrote before submitting it
- follow up on your bugs

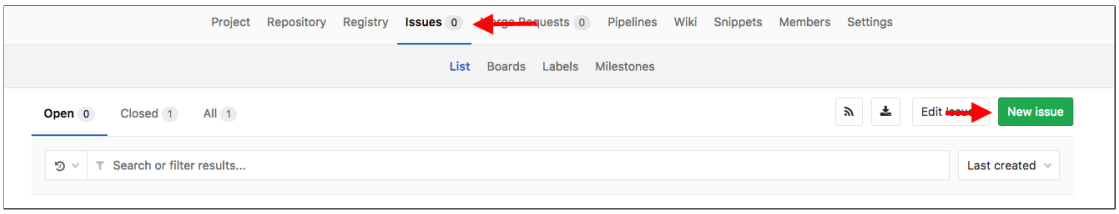
A.2 How to Create a Bug Report in `GitLab`

1. Create a new issue (Figures 1a).
2. Describe the bug (Figures 2a and 2b).
3. Select/Create a label for the bug (Figures 3a and 3b). You may also assign the bug to a developer, add a weight and due date etc.
4. Work on fixing the open issues and close them once resolved (Figures 4a and 4b).

References

[1] Simon Tatham. *How to Report Bugs Effectively*. 1999.
`https://www.chiark.greenend.org.uk/~sgtatham/bugs.html`
(visited on 24/08/2017)

[2] Joel Montvelisky. *Principles of Good Bug Reporting*. 2008.
`http://qablog.practitest.com/principles-of-good-bug-reporting/`
(visited on 28/08/2017)



(a) Add a new issue

Figure 1: Creating a new issue in `GitLab`

When loading an FMEA, the "Theory" text field label is not entirely visible

Write **Preview**

When loading a new FMEA as an Excel File using the GUI Menu Item *Load Model*, the text field label *Theory* is not entirely visible.

Repeatable: Always

Steps to repeat:

1. Execute Motes with the option *-gui*
2. Click on the Menu Item *File*
3. Click on *Load Model*
4. Use the file opener and select an appropriate Excel file (e.g. Converter_Test.xlsx)
5. Observe that the Text field and GUI does not resize correctly

Expected Results: The text field label *Theory* is completely visible. Expected Result: *Theory* entirely visible

Actual Results: The text field label *Theory* is cut off at the beginning Actual Result: *Theory* cut off

☐ This issue is confidential and should only be visible to team members with at least Reporter access.

(a) Good example

Title: Not everyting is visible!!!!!

Add description templates to help your contributors communicate effectively!

Description: Write **Preview**

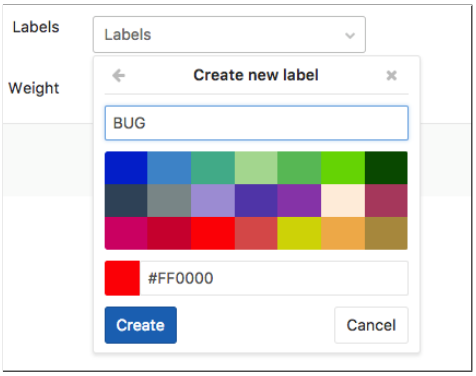
I load a model and not all labels are visible. PLEASE FIX ASAP!!!

BTW: Sometimes the program crashes.

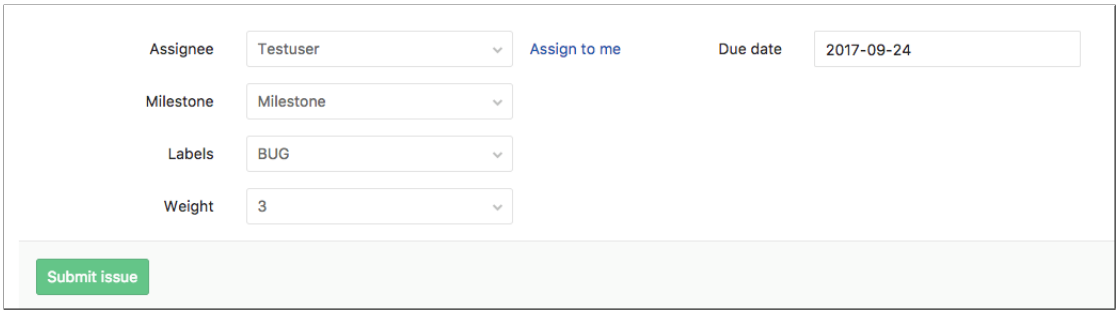
☐ This issue is confidential and should only be visible to team members with at least Reporter access.

(b) Bad example

Figure 2: Describe the bug

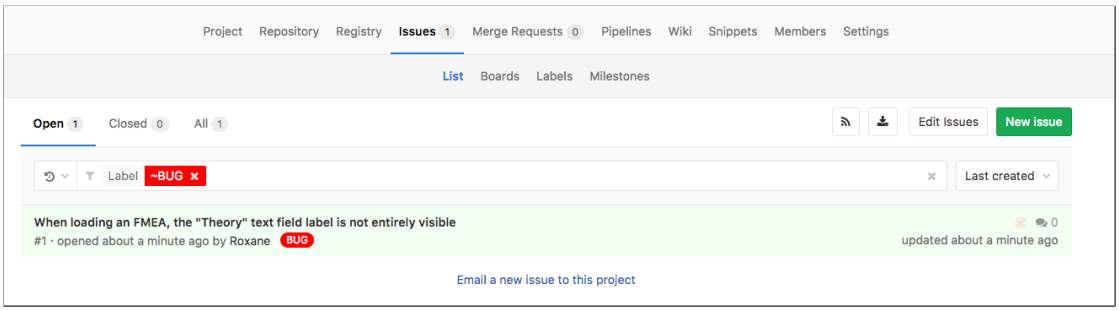


(a) Define a new label name and color code

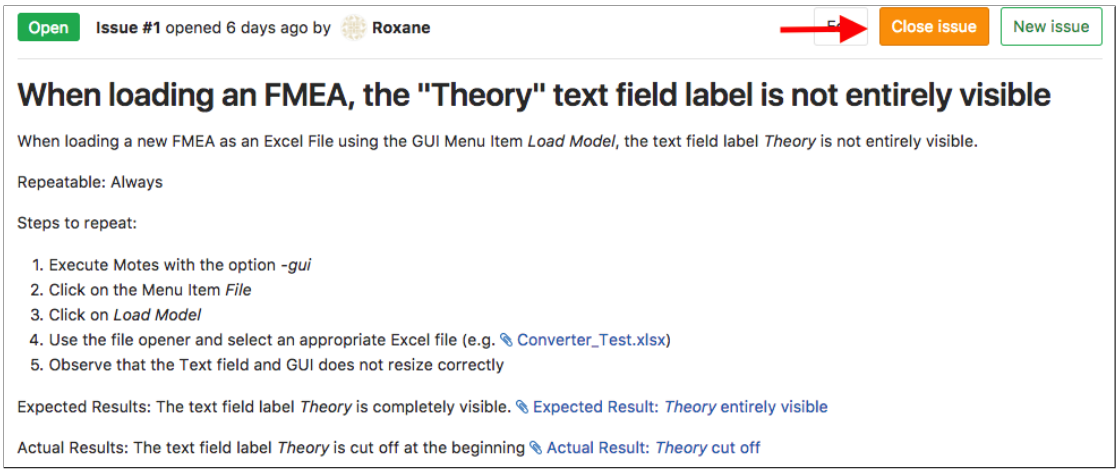


(b) Provide additional specifics, e.g., assign the bug to a developer, define a weight (i.e., impact of the bug), or set a due date

Figure 3: Add specifics to the bug



(a) Open issues



(b) Close fixed issues

Figure 4: Use the issue tracking system