UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Bachelor Thesis
in Mobile and Embedded Systems

Towards Understandable CPAchecker Counterexamples

Magdalena Murr

Supervisor:
Prof. Dr. Dirk Beyer

January 6, 2016

Abstract

CPAchecker is a tool for software verification. It is able to provide a counterexample-report. The report generator can be used to compile an interactive analysis report of a CPAchecker run, that means it shows the lines of code that lead to the error (errorpath) and related graphs, as well as source-code, log, statistics, and configuration properties. The existing counterexample-report has several problems: it has no detailed documentation, the design is a little outdated, and some features do not work well. This thesis is about the development of a renewed counterexample-report with new features that allow seeing all value-assignments of the analysed program with one click, or search through the errorpath and the value-assignments. This should give the user more possibilities to interact with the pictured data and by that enable him to better and quicker understand CPAchecker counterexamples.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Goal and Approach

The first goal of this thesis was to implement the given range of functionality with a more modern design, with a chosen architecture and framework, and well documented. It should be easy to create and to use the report – the fewer steps it takes to generate it, the better. Ideally the report should only be one file at the end so it could be copied around. One important step towards this goal was to generate most parts of the report within the normal CPAchecker run and only outsource the creation of the CFA- and ARG-graphs. These graphs are generated with the programme "graphviz" and this could slow down the process of the whole CPAchecker analysis significantly (as "graphviz" takes a long time to generate a big graph what often applies especially to the ARG-graph).
The other goal was to extend the counterexample-report with new features. My approach here was to perform a study amongst the users, to find out which features would enhance the report best (Annex B, First Evaluation (Survey), page 50), to implement those features and examine with a second survey (Annex C, Second Evaluation (Survey), page 54) if users could handle the new features well and if any problems occurred. In fact the second survey identified several problems that were solved in a last implementation phase.

1.2. Related Work

As related work the presentation of the error trace of the "Institute for System Programming of the Russian Academy of Sciences - Verification Center of the Operating System Linux" can be stated[1]. They list problems found in Linux. The comparability is good, because they often use CPAchecker as verifier, too. They show two

[1] `http://linuxtesting.org/results/bug?id=11`

windows side by side: the error trace (called "errorpath" in the counterexample-report) and the source code (possibly in different files). They give a lot of opportunities to manipulate the depiction of the error trace (like line-number on/off, function-bodies on/off, and many more). Below the two windows is space for a comment and some additional information about the bug like the verifier that was used, timestamp, and if it is fixed or not.
They do not provide graphs (CFA/ARG) which would make understanding source-code and error trace a lot easier, but the configurability of the depiction of the error trace could be an interesting feature – not only for the error trace, but also for the other parts of the report.

2. Existing Solution

2.1. Range of Functionality

In the following section we will cover all functionalities of the old counterexample-report (Fig. 2.1). Information about that can be found in the official documentation at the CPAchecker website[1]. We will take that information as basis, enhanced by personal observations and if necessary tips to mistakes in the documentation.

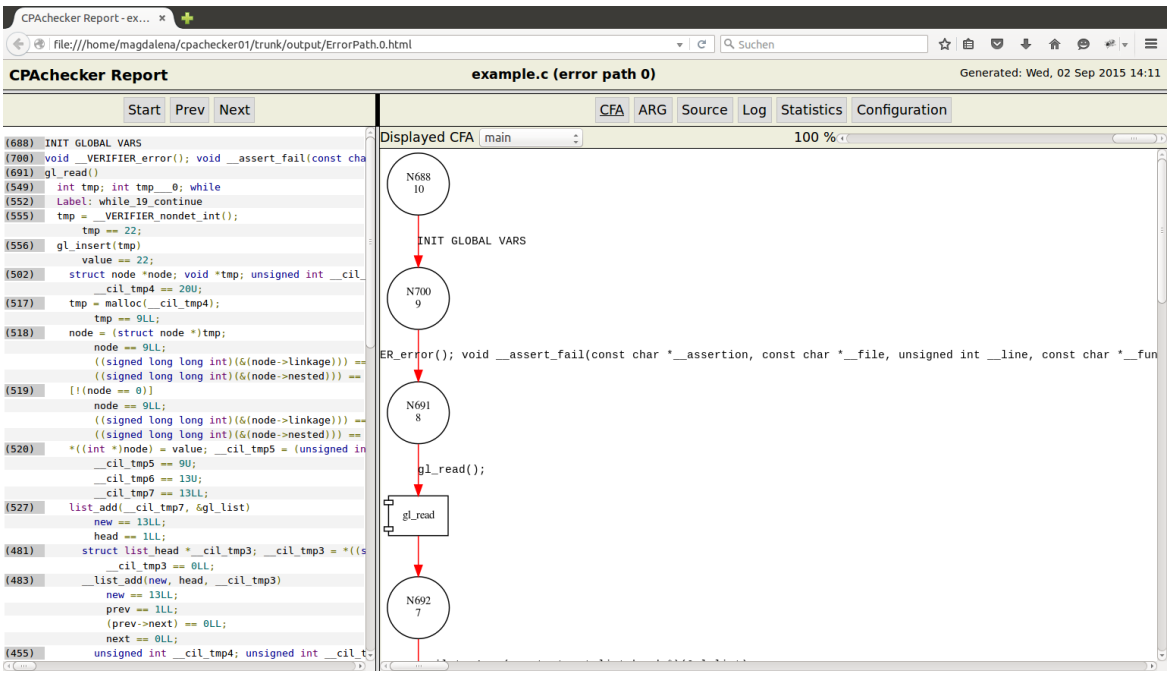


Figure 2.1.: The startscreen of the old counterexample-report.

The description of the functionalities will be divided in the different parts of the counterexample-report. On the left side we have the errorpath and on the right side we have several tabs, containing the CFA- and ARG-graphs, the source-code, log, statistics, and configuration properties.

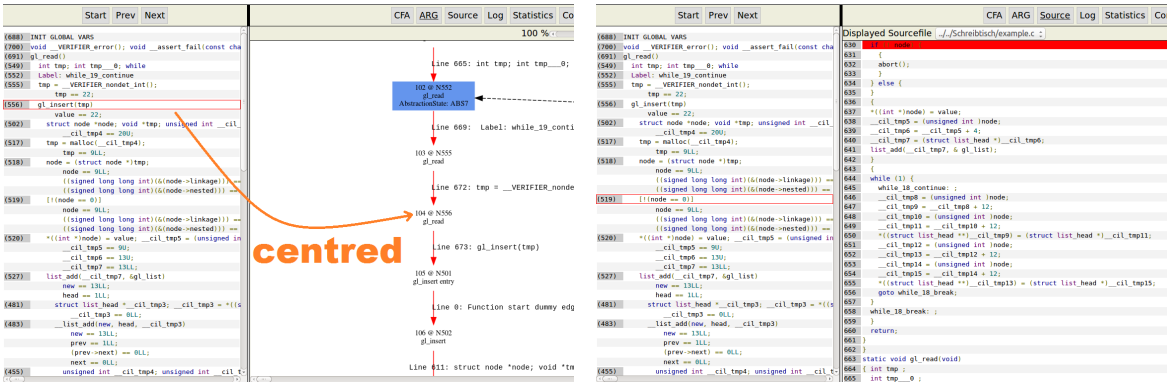
[1] https://github.com/sosy-lab/cpachecker/blob/trunk/doc/BuildReport.txt

- Left Panel: Errorpath (if a bug is found) (Fig. 2.2a)

  The indentation reflects the height of the call stack.
  Every line of the errorpath consists of a number at the left, and some code at the right. The number represents a node of the CFA (it is actually its unique node-number) and the code represents the label of the edge that leaves this node (Fig. 2.2b). By clicking the node/edge we can jump to the location in CFA/ARG/source (depending on the active tab) (Fig. 2.3a, Fig. 2.3b).
  With the buttons “Start”,“Prev”, and “Next” (or by clicking nodes/edges) we can walk along the errorpath. (Fig. 2.2c). The current position within the errorpath is marked with a red frame around the line. This mark goes forward in the errorpath and the active tab. In the source-tab the mark would be a red background of the line, in CFA/ARG there is no visual mark, the appropriate edge/node is just scrolled to the vertical centre of the view.
  Function start, function return, and unlabeled edges are not displayed. This feature causes a multiple stop of the mark (red frame) at these positions. That means the mark does not jump to the next/previous line when clicking “Next”/“Prev” for as much clicks as are not displayed. In CFA/ARG/source the mark goes forward with every click though (as these edges/nodes are displayed there).

(a) The errorpath-section (b) Nodes and edges (c) The navigation-buttons

Figure 2.2.: Elements of the errorpath



(a) Errorpath and ARG (b) Errorpath and source

Figure 2.3.: Errorpath: jump to associated location in active the tab

- Right Panel: CFA Tab

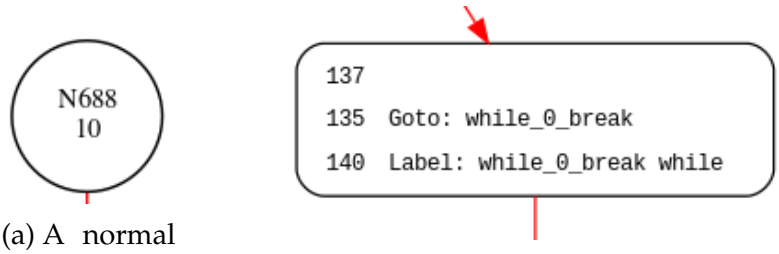
  The CFA has been cut into multiple images (one image per function). The displayed function can be changed by using the dropdown-menu at the top (Fig. 2.5b).

  Almost every node is labeled with two numbers. The first number "N***" is the node-number, each is unique within all CFAs. This is the number that is displayed in the errorpath (left panel) as well and to which is referred to in the ARG ("@N***"). The second number "***" gives orientation within one CFA, the flow begins with the highest and ends with "0" (Fig. 2.4a).

  Linear sequences of "normal" edges (statement-edges, declaration-edges, and blank edges) are displayed as a table in one big node. The left column contains the node number and the right column contains the label of the edge that is leaving this node (Fig. 2.4b). The node numbers in these multi-nodes are the "N***"-numbers even if they do not have an "N" in front of the number. In case there are too many nodes/edges to fit in one table-node, a node saying "Long linear chain of edges between nodes *** and ***" occurs.

  Because the CFA is spread out over multiple images, function call nodes and edges have been introduced. The node has a special form and indicates that the control flow at this point is continued in another CFA-graph.

  The route of the errorpath in the CFAs is highlighted in red (Fig. 2.5a). With an open CFA-tab we can jump from edge to edge in the red-highlighted errorpath by clicking on the relating lines in the errorpath in the left panel. It will vertically center the related edge and automatically change the shown CFA if necessary.

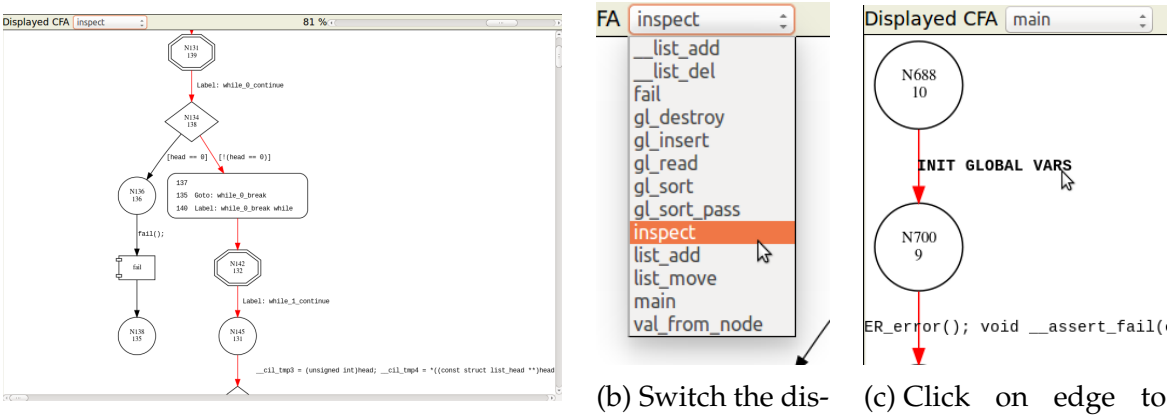


(a) A normal CFA-node

(b) A CFA-multistatement-node

Figure 2.4.: The label of CFA-nodes

Click an edge (or its label) to jump to the location in the source-code (Fig. 2.5c). The view jumps to the source-tab even if there is not an equivalent line in the source-code. The mark (red background) stays at the last marked position then. When jumping to a function head, the line of code before this function is marked.



(a) The errorpath is highlighted in red

(b) Switch the displayed CFA

(c) Click on edge to jump to source-tab

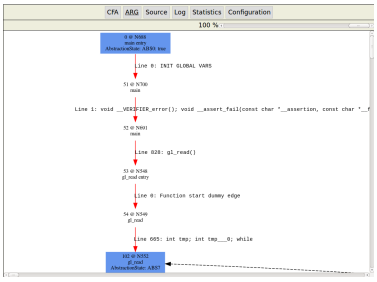
Figure 2.5.: The CFA-tab

- Right Panel: ARG Tab (Fig. 2.6)

  The route of the errorpath within the ARG is highlighted in red (Fig. 2.6a). By clicking an edge we can jump to the associated location in the source-code (Fig. 2.6d). By clicking a node we can jump to the associated location in the CFA (Fig. 2.6c). At this point we have an error in the documentation of CPAchecker (It says "Click an edge to jump to the location in the CFA."[2]).

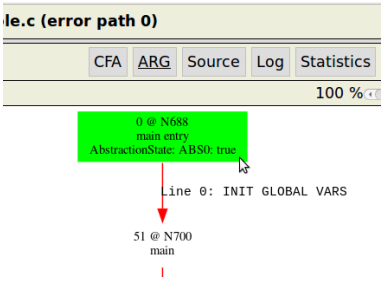
[2] `https://github.com/sosy-lab/cpachecker/blob/trunk/doc/BuildReport.txt`

The nodes in the ARG are labeled at least with two numbers and the name of the function that the program is in at this point. The first number "***" is unique within the ARG, the second "@N***" refers to the node-number of the belonging CFA-node (Fig. 2.6b).

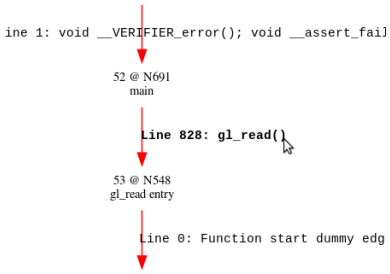


(a) The errorpath is highlighted in red

(b) The label of an ARG-node



(c) Click on node to jump to CFA-tab



(d) Click on edge to jump to source-tab

Figure 2.6.: The ARG-tab

- Right Panel: Source Tab

  It displays the source-code that was analysed with CPAchecker. If there are several source-files, we can select one in the dropdow-menu at the top.

- Right Panel: Log Tab

  It displays the output of CPAchecker.

- Right Panel: Configuration Tab

  It shows the configuration-settings that were made before running CPAchecker and building the report.

- Right Panel: Statistics Tab

  It shows some statistics that CPAchecker has generated.

2.2. Data from CPAchecker

One of the first steps in this project was to analyse the data that is used for the counterexample-report, where the files are generated in the flow of CPAchecker, where they are stored and what their content means exactly. The files that the counterexample-report needs are basically .json, .dot and .txt. We have .txt files that contain statistics, log and configuration properties. We have a .dot file for every graph (CFA/ARG) that later occurs in the counterexample-report. The .dot files are used to generate .svg files (with the programme "graphviz"). These .svg files can be used directly for showing the graphs in the counterexample-report. The .json files contain all necessary information about the errorpath and some additional information about the graphs, for example which nodes are combined in a multistatement-node. The analysis of these files (how can you extract which information) is not really relevant to follow this thesis, but the interested reader will find the work I did in the attachment (Annex A, Data from CPAchecker, page 48), as there is no detailed documentation for this topic available at the moment.

2.3. Known Problems

In this section we will cover problems and bugs that are known about the old counterexample-report. Again, there is information about that in the official documentation[3] and again we will take that as a basis and have it enhanced by my own observations.

- Browser-Problems

  The counterexample-report is in HTML format and therefore can be displayed in a browser. It works best in Firefox, because other browsers seem to have issues accessing local resources (CFA-/ARG-graphs) via Javascript. In case the file is hosted on a server, Google Chrome/Chromium should also work, except that CFA- and ARG-graph do not scroll to the correct element when navigating through the errorpath.

- Left Panel: Errorpath

  When we navigate through the errorpath with the buttons "Start", "Prev", and "Next", the view does not scroll with the marked element (red frame).

[3] `https://github.com/sosy-lab/cpachecker/blob/trunk/doc/BuildReport.txt`

For example if we have a long errorpath, the view is scrolled to the bottom, and then we click “Start”, the mark jumps to the first element, but the view does not scroll to the top, so we cannot see it.
Another problem when we navigate through the errorpath is, that on the right only the active tab gets updated. That means if we change the tab and want to scroll to the associated element or mark it (red background in source-tab), we have to click the line in the errorpath again.

- Right Panel: Graphs (CFA/ARG)

  The CFA- and ARG-tab use iframes to load the .svg images and the red errorpath is highlighted dynamically with Javascript. This happens 0.5 seconds after the .svg was loaded. In case the report would be requested from a server, this would possibly not apply anymore and the errorpath would not be highlighted.
  The slider for zooming a graph out (Fig. 2.7) cannot be moved by click&drag like the presentation indicates. We have to click on the arrows to the left and right, or click at a point in the slidebar left/right from the slider to scroll the graph.

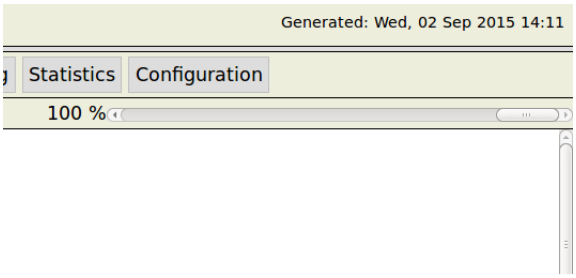


Figure 2.7.: The Zoom-Slider for CFA and ARG

- Right Panel: CFA

  Jumping to the source-tab by clicking edges does not always work correct. Some edges do not jump to a location in the source-code (even if the tab gets changed), the mark (red background) just stays at its last position and edges that leave a multistatement-node (Fig. 2.4b) jump to a wrong location in the source-code. This problem probably goes back to the data the counterexample-report gets from CPAchecker (Annex A, Data from CPAchecker, page 48).

- Right Panel: ARG

  The ARG often has an enormous size. If that is the case, it could take awhile until it is loaded when switching to the ARG-tab. It is also possible, that we do not see anything in the view at first and that we have to scroll to the right to find the beginning of the ARG (if nodes/edges below reach out far to the left).
  Sometimes, a few edges that are part of the errorpath stay black (they should be highlighted in red), but reloading the page seems to fix that.
  Not all edges of the ARG are clickable (this error seems to only affect the black edges that do not belong to the errorpath).
  The programme "graphviz" that is used for generating the .svg files does not seem to like to create nodes with a very long label (for exapmple the node in main(), that contains the global declarations). Instead it only writes the node number.

3. Design

3.1. Architecture

The first decision I made was choosing an architecture. As MVC is a very common architecture, I started my search for a suitable one from that point. MVC would be appropriate as we have data in the background (model) that should somehow be presented (view) and users can change the presentation of the data and should be able to interact with it (controller) up to a certain point (navigate through it, jump from one representation of the data to another, examine it).
Another interesting architecture is MVVM (Model-View-ViewModel). It is a variant of the classic MVC. It should give the possibility to programme logic and user interface completely indepentent from each other. There is no controller that is responsible for every action that is taken. We have a view, that shows the data from the model. The viewmodel sort of "translates" from one layer to the other and contains the logic, like functions, that have to be executed (just like the controller in the MVC), but the viewmodel does not control everything. The view is bind to the model (via the viewmodel) and gets updated when the model changes and vice versa.
The conclusion at this point is, that both architectures fit our needs. They both allow to implement logic and user interface independently. In the next subsection we will have a look at frameworks that follow MVC or MVVM and with the decision for a framework we will decide on an architecture.

3.2. Framework

The next step was to choose a framework, so that the new implementation would be well organised and structured.
I did some research on the most spread frameworks and did a short comparison between Ember (MVC), Backbone (MVC), AngularJS (MVC+MVVM), and Knock-

out (MVVM). For that purpose we have some main characteristics listed (Tab. 3.1), based on statements from Stefan Wintermeyer (Ember[1]) and Golo Roden (Knockout[2], Backbone[3], AngularJS[4]) – both experienced in the field of web development. The characteristics in the table were mentioned in those articles. In italics we see why they are an advantage or disadvantage for this project.

Based on the results you can observe in the table (3.1), I had a closer look on AngularJS. I was not able to find any strong disadvantages. On the contrary it has some concepts that seemed very promising for our purpose and so I decided to use AngularJS for this project. What follows is a closer look at this framework.

AngularJS and Bootstrap Like we can see in the table, AngularJS is maintained by Google and this hopefully indicates detailed documentation (a look at the official website[5] confirms that). When we have a look at the trend of Google-searches[6] we can see, that since a few years AngularJS leaves the other frameworks far behind (assuming that Google would not manipulate its statistics on their particular interests). This further indicates a well established developer community with lots of helpful forum entries.

AngularJS is a Javascript framework that can be simply added to a HTML-file via the <script> tag. A usual AngularJS application consists of an HTML-file (view) and one or more controllers (summarised in a module) that implement the logic and manages the data (model).

AngularJS extends the HTML-syntax, so we can use new elements like "directives" and "expressions" to realise its main concept: the two-way-data-binding. Two-way-data-binding means, that view and model are connected bidirectional – changes in the view are immediately reflected in the model and the other way round.

[1] http://www.heise.de/developer/artikel/Ember-js-1-1-im-Einsatz-2053975.html

[2] http://www.heise.de/developer/artikel/Model-View-ViewModel-mit-Knockout-js-1928690.html

[3] http://www.heise.de/developer/artikel/Model-View-Controller-mit-Backbone-js-1938069.html

[4] http://www.heise.de/developer/artikel/Webanwendungen-mit-AngularJS-1955101.html

[5] https://angularjs.org/

[6] https://www.google.com/trends/explore?hl=en-US#q=ember.js%2C%20angularjs%2C%20backbone.js%2C%20knockout.js&date=today%2012-m&cmpt=q&tz=Etc%2FGMT%2B8

Framework	Pros	Cons
Ember	Very strict conventions *You have a fixed structure you can hold on to*	Very strict conventions *You can only do what the developers of Ember planned you to do*
		High initial hurdle *This work was limited to a relatively small time-frame*
		Many versions *Older sources for help cannot be used* *Compatibility of code expires quickly*
Backbone	Good maintainability	Much infrastructure for small projects *This project is very small*
		You cannot manage different views *We have several views*
		No concept for actualization of view *At least the mark has to change*
AngularJS	Getting in is very easy *This work was limited to a relatively small time-frame*	
	Code is automatically clearly structures *This was one of the main reasons a new report was necessary*	
	Maintained by Google *We usually can expect good documentation*	
Knockout	Steep learning curve *This project was limited to a relatively small time-frame*	No possibility to change or reload views *We have many different views and want to keep the possibility to load the graphs later*

Table 3.1.: Framework-comparison

"Directives" allow us to attach a special behaviour to an HTML-element. For example the "ng-click" directive adds a mouse-click-event-listener to the element it is attached to. With "ng-click=,doSomething()'" we could call a function "doSomething()" in the related controller. We do not have to add a specific listener in our

javascript code.
“Expressions” can directly display data in the view. With curly braces we could display name and whenever the variable “name” (defined in the related controller) changes, the view updates automatically and immediately.
To include AngularJS in an application we just have to attach “ng-app=,module‘” to the outermost html element that should include AngularJS (for example the body-element). “module” is the module that contains all controllers. Different controllers (with logic and data) from the module can then be attached to several parts of the application that they should control and supply with data. The two-way-data-binding seems a good concept for our purpose (data can change in realtime, one action triggers an reaction in another window at once).

Through AngularJS I came across Bootstrap. Bootstrap is a Javascript framework as well and it provides a a large CSS stylesheet. It is used more for designing purposes and as it could be used in combination with AngularJS and the cloud-interface of CPAchecker uses it as well, I decided to add it to this project.

4. Implementation

4.1. Range of Functionality (New Features and Solved Problems)

In the following section we will cover the functionalities of the new counterexample-report (Fig. 4.1). I will only point out what has changed in comparison to the old counterexample-report – I will show new features and indicate solved problems.

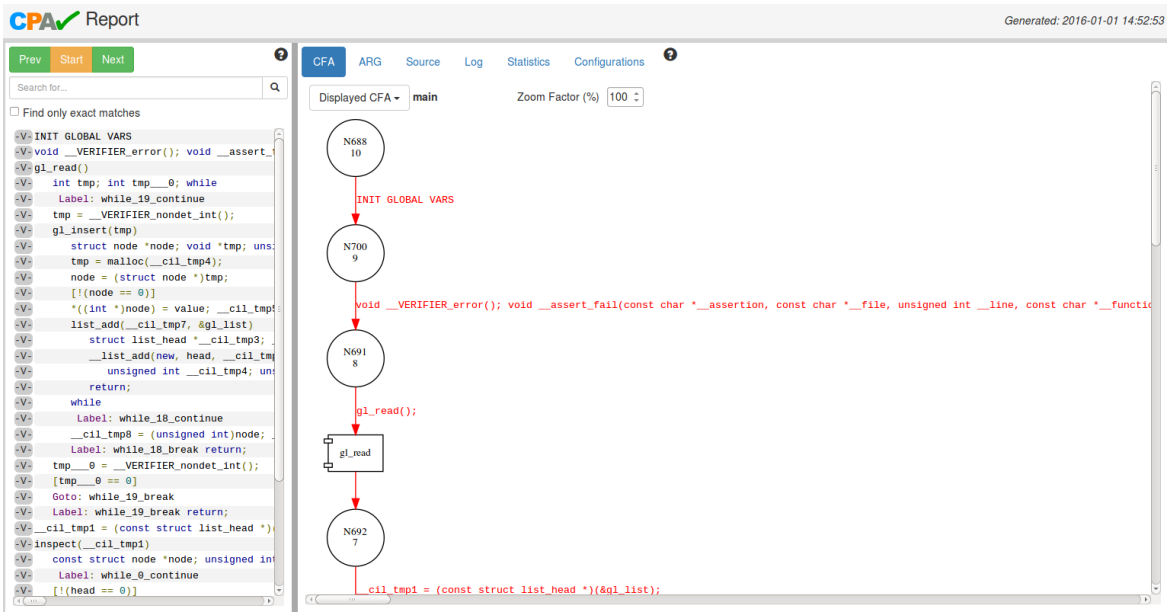


Figure 4.1.: The start screen of the new counterexample-report.

Like in the chapter “2, Existing Solution” (page 9), the description of the functionalities will be divided in the different parts of the counterexample-report. For purposes of consistency I decided to maintain the structure of the old counterexample-report with the errorpath in the left panel and the different tabs in the right panel. So we have the errorpath and the search functionality on the left side. On the right side we have the tabs, containing the CFA- and ARG-graphs, and the source-code.

Additionally, we have two help buttons, one on each side of the counterexample-report. The presentation of log, statistics, and configuration properties has not changed and therefore will not be treated here.

- Left Panel: Errorpath (if a bug is found)

  One line of the errorpath consists of a button "-V-" and some code that represents the label of an edge (CFA/ARG) that is a “step” in the errorpath.
  By clicking the “-V-” button we can see all relevant variables at that point of the programme and their values (Fig.4.2).

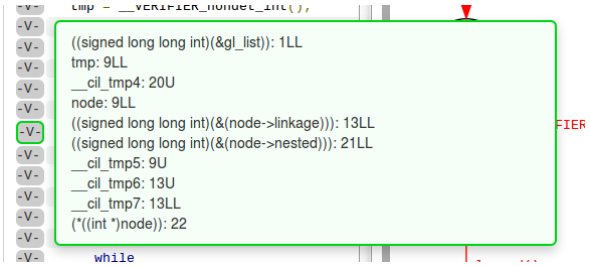
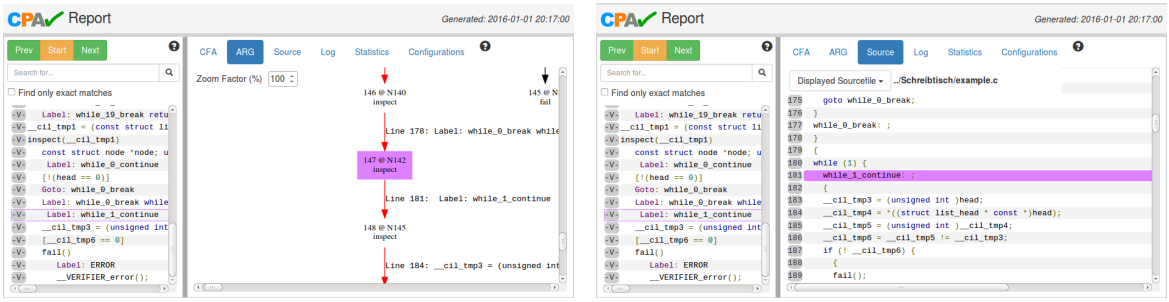


Figure 4.2.: Value-Assignments

With the buttons “Start”, “Prev”, and “Next” or by clicking the edge-label we can navigate through the errorpath (like in the old counterexample-report). The difference is that now the current position in the errorpath is marked with a purple frame around the line. The associated location in the active tab (CFA/ARG/source) is scrolled into view and the same purple colour is used to mark this location in the active tab and the inactive tabs as well (Fig. 4.3). It is only the purple mark that gets updated in the inactive tabs (if you want the view to scroll to the right location after you change the tab you have to click on the errorpath-edge again), but I consider this a feature - an inactive tab that changes its focus while the user is “away” might be confusing.
Clicking the button “Start” automatically scrolls to the top of the errorpath, so that the marked line will be in the view.

Another feature has to do with the hiding of function start, function return, and unlabeled edges. I maintained hiding these elements, but the “Next” and “Prev”-Button does not “rest” at these positions anymore. Like I explained in “Existing Solution”(page 9), navigating with these buttons stopped at those positions for as many clicks as edges were left out. I see this behaviour as pos-



(a) Errorpath and ARG

(b) Errorpath and source

Figure 4.3.: The current location gets marked in the errorpath and the active tab.

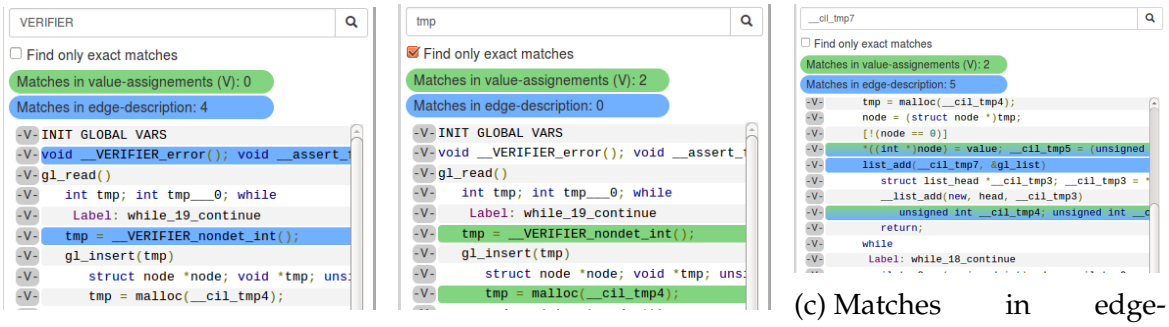
sibly confusing and that is why I did not adopt it in the new counterexample-report.

- Left Panel: Search Functionality

  The search functionality's range is limited to the left panel (edge-label of errorpath, value-assignments). Matches within the edge-labels of the errorpath are marked with a blue-highlighted line (Fig. 4.4a) and matches within the value-assignments are marked with a green-highlighted line (Fig. 4.4b). A line where a match within edge-label and value-assignments occurred will be highlighted with a colour gradient of blue and green (Fig. 4.4c).
  There are some special features when searching within value-assignments: First, we can search for the name and also the value of a variable. Second, it only searches for occurences where the variable was initialized or where the variable changed its value (because in every line after initialization or change, the variable will occur with the same value as before).
  The search functionality can be modified to only show "exact matches". For example when we search for the variable "tmp", the variable "tmp7" will occur in the matches as well, except we have set "exact matches".

- Right Panel: CFA

  The graphs are directly included into the HTML as <svg>-elements, that means they are all loaded from the beginning. The red marking of the errorpath is implemented as a property of the edge-elements, so it is highlighted from the beginning as well (the old counterexample-report had problems with that).
  Clicking edges leaving the introduced multistatement nodes (Fig. 2.4b) now



(a) Matches in edge-descriptions (b) Matches in value-assignments (c) Matches in edge-descriptions and value-assignments

Figure 4.4.

jumps to the correct location in the source-code (the old Counterexample-Report had problems with that).
Clicking a function-node (they were implemented because the CFA is spread out over multiple images) jumps to the CFA of that function (Fig. 4.5).

Nodes/Edges that are marked (by clicking on edge-label in errorpath or

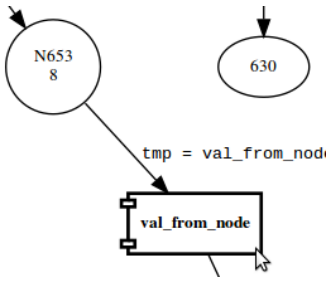


Figure 4.5.: A function-node

equivalent node in ARG) are scrolled to the centre of the view and highlighted in purple.
A doubleclick at an edge jumps to the associated location in the source-tab.
The graph can be zoomed in and out by adding a value or pushing arrows (the zoom-slider of the old counterexample-report did not work smoothly)(Fig. 4.6).

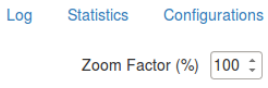


Figure 4.6.: The zoom-functionality

- Right Panel: ARG

  The graphs are included as <svg>-elements (see CFA).
  Marked nodes (by clicking the edge-label in errorpath) are scrolled to the centre and highlighted in purple.
  By doubleclicking a node we can jump to the associated location in the CFA-tab and by doubleclicking an edge we can jump to the associated location in the source-tab.

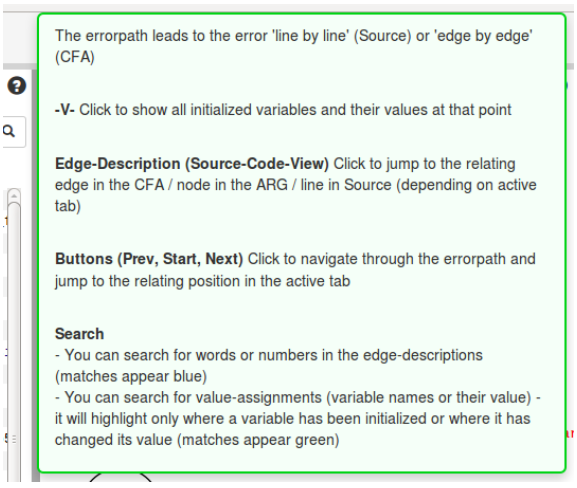
- Right Panel: Source

  The current location (from the errorpath) is marked in purple and scrolled to the vertical centre of the view.

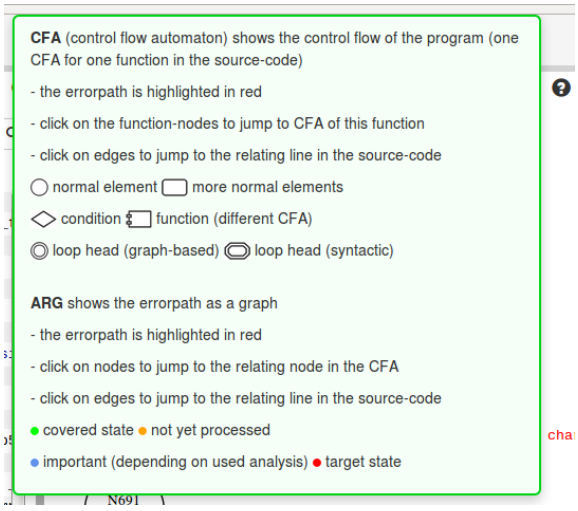
- Help-Texts

  There are two help-buttons (Fig. 4.7), one for the left panel (errorpath, value-assignments, and search-functionality) (Fig. 4.8a) and one for the right panel (CFA and ARG) (Fig. 4.8b).

Figure 4.7.: The help-button



(a) The help-text for the left panel



(b) The help-text for the right panel

Figure 4.8.

4.2. How the Report implements Architecture and Framework

The counterexample-report can be roughly divided into the view (HTML) and the logic and data behind (Javascript). This distribution only gets violated, because static files like statistics or configuration properties and the <svg>-elements of the CFA- and ARG-graphs are directly inserted into the HTML template.
The view of the report can be easily divided into several parts and each of these parts has its own controller (Javascript). The controllers provide data for their part of the view and functions to deal with different events. The controllers can communicate with each other via "broadcast"-messages. Each controller has its own "scope", that means its own memory and they all share the memory "rootScope".
After the page is loaded, a function (outside the AngularJS logic) is called that equips the CFA- and ARG-elements with special ids (so they can easily be accessed later) and that highlights the errorpath in the CFA (red colour).

4.2.1. View with Data-Binding

In Fig. 4.9 we have an overview of the hierarchy of those main parts of the HTML that have a controller attached. The hierarchy is only valid for the view, not the controllers. That means every inner view element can be reached by the controller of the outer element, but the outer controllers do not have access to the variables or methods of the controllers of the inner elements. Every Controller has its own independent scope and the controllers only share the root-scope. The data that is stored in those scopes is not inserted into the view by the controllers manually. Through directives and expressions (explained in "3.2 AngularJS and Bootstrap", page 18) the data is bind to the view and vice versa.
"<body>" is the HTML-element that contains all other elements of the view and it is attached to "ReportController". This controller provides the current date, the CPAchecker-logo and the content for the help-texts. This data is stored in the scope of the controller and where it should be inserted in the view it is accessed by date for example.
"Left Panel (Errorpath)" covers the errorpath, its navigation-buttons ("Start", "Prev", "Next"), the search functionality, and the value-assignments. It is attached to the "ErrorpathController". The data for the value-assignments and the edge-labels of the errorpath is bind here.

The “Search-Functionality” has its own controller “SearchController”. The number of matches is bind to the view here (it gets automatically updated every time the number of matches changes due to a new search).
The “Value-Assignments” are attached to the “ValueAssignmentsController”. All value-assignments for each line of the errorpath are bind to the view after being preprocessed in the errorpath-controller.
The “CFA-Tab” is attached to the “CFAController”. The graph itself is inserted directly in the HTML template, but the names of the single CFAs in the dropdown-menu are bind to the view. The “Zoom”-functionality of the CFA-tab has its own controller “ZoomController”. The current value of the zoom is bind to the controller with the directive “ng-model=,zoomFactorCFA‘”. This is a directive that binds an input (zoom scale) in the view to a variable (“zoomFactorCFA”) in the controller.
The “ARG-Tab” is attached to the “ARGController”. Again, the graph itself is inserted directly in the HTML. The “Zoom”-functionality is attached to the same “ZoomController” as the one in the CFA-tab and the current value is bind to the controller with “ng-model=,zoomFactorARG‘”. The “Source-Tab” is attached to the “SourceController”. The source-file is also directly inserted into the HTML template, but like in the CFA-tab, the names of the available source-files are bind to the dropdown-menu.

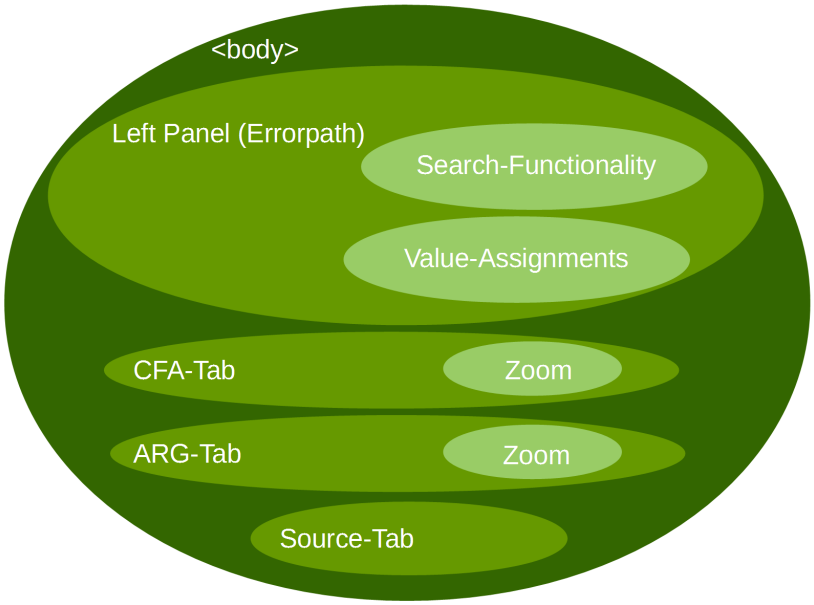


Figure 4.9.: The hierarchy of view elements that are attached to a controller

4.2.2. Logic in Controllers

In Fig. 4.10 we can see all controllers with their scope's variables (first segment), their listeners to broadcast-events ($\rightarrow$ "$on(event)", second segment), and their scope's methods (third segment). For internal processes or calculations the controllers use additional variables that are not bind to their scope and therefore not accessible for the view. The broadcast-events that are sent by the controllers for communication purposes are sent over the root-scope. That means every other controller could receive them, but only those controllers that are interested in the event implement a listener to it. Now we will have a closer look at the controllers and the functionality they provide to the application.

The report-controller (bind to the HTML-<body>-element) manages general tasks. It is responsible for changing tabs (with the method "setTab()") responding to a click-event from the view on one of the tabs or responding to a broadcast-event from the CFA- or the ARG-controller. It listens to the event "ChangeTab" that is sent by those controllers and that provides also the tab that should be switched to. The listener to "FirstTimeErrorpathElementIsSelected" – sent by the errorpath-controller – is called the first time an element in the errorpath is selected. It invokes the function "setMarginForGraphs()" that allows CFA- and ARG-elements to be scrolled to the centre (when marked). The other methods ("setWidth()", "setMouseUp", "setMouseDown") allow to change the size of the left panel (errorpath) and right panel (CFA-tab, ARG-tab, etc.) by clicking&draggin the line between them.

The errorpath-controller (left panel) preprocesses the errorpath-data that is stored in the root-scope (because different controllers need to access it). This is the data we get from the "ErrorPath.json"-file. The variable contains an array of objects, one object for each line of the errorpath with all information about it that we need (look "A, Data from CPAchecker", page 48). It further handles click-events on the navigation-buttons "Start", "Prev", and "Next" and click-events on lines of the errorpath by broadcasting the line that has been clicked. The method "setLine()" marks the selected line (purple frame) in the errorpath and in case it was the first time an element has been selected, broadcasts the "FirstTimeErrorpathElementIsSelected"-event.

The search-controller handles search queries by browsing through the errorpath-data (stored in root-scope).

The value-assignments-controller changes the presentation of the "-V-" button if

its value-assignments window is shown (the presentation of the value-assignments themselves is handled by data-binding and does not need a controller).
The CFA-controller handles the click-events on CFA-edges by broadcasting the "clickedCFAEdge" message with information about which source-line is associated to the clicked edge (because a click on an edge jumps to the source-tab) and by broadcasting “ChangeTab”. With “setCFAFunction” and “cfaFunctionIsSet”, CFA-controller manages which CFA is shown in the view. CFA-controller has several listeners implemented: “clickedErrorpathButton” and “clickedErrorpathElement” invoke the method “markCFAEdge()” (providing the number of the edge that should be marked). The listener “clickedARGEdge” invokes “markCFANode” (providing the number of the node that should be marked). Both methods (“markCFAEdge”, “markCFANode”) then call “scrollToCFAElement” to scroll the marked element to the centre of the view.
The ARG-controller handles click-events on ARG-nodes and -edges by broadcasting the events “clickedARGNode” or “clickedARGEdge” as well as the event “ChangeTab” (providing the number of the CFA-tab in case of a clicked node or the number of the source-tab in case of a clicked edge). ARG-controller listens to the events “clickedErrorpathButton”, and “clickedErrorpathElement”. In case one of those events is received, it calls “markARGNode” (providing the number of the node that should be marked) and this method then calls “scrollToCFAElement” that scroll to the marked Element.
The source-controller handles the source-file that is shown in the source-tab with the methods “setSourceFile()” and “sourceFileIsSet()”. It stores the number of the currently selected source-file in the variable “selectedSourceFile”. The SourceController listens to the events “clickedARGEdge”, “clickedCFAEdge”, “clickedErrorpathButton”, and “clickedErrorpathElement”. It reacts to those events by calling the method “markSource()” that marks the source line whose number has been transmitted with the event.
Finally, the zoom-controller is responsible for the zoom functionality of the CFA- and ARG-graphs. It listens to input of the view and to the broadcasting-event “clearCFAZoom”. This event is sent if the displayed CFA changes. The whole CFA-tab only has one zoom-controller, so the zoom is reset every time the displayed CFA changes.

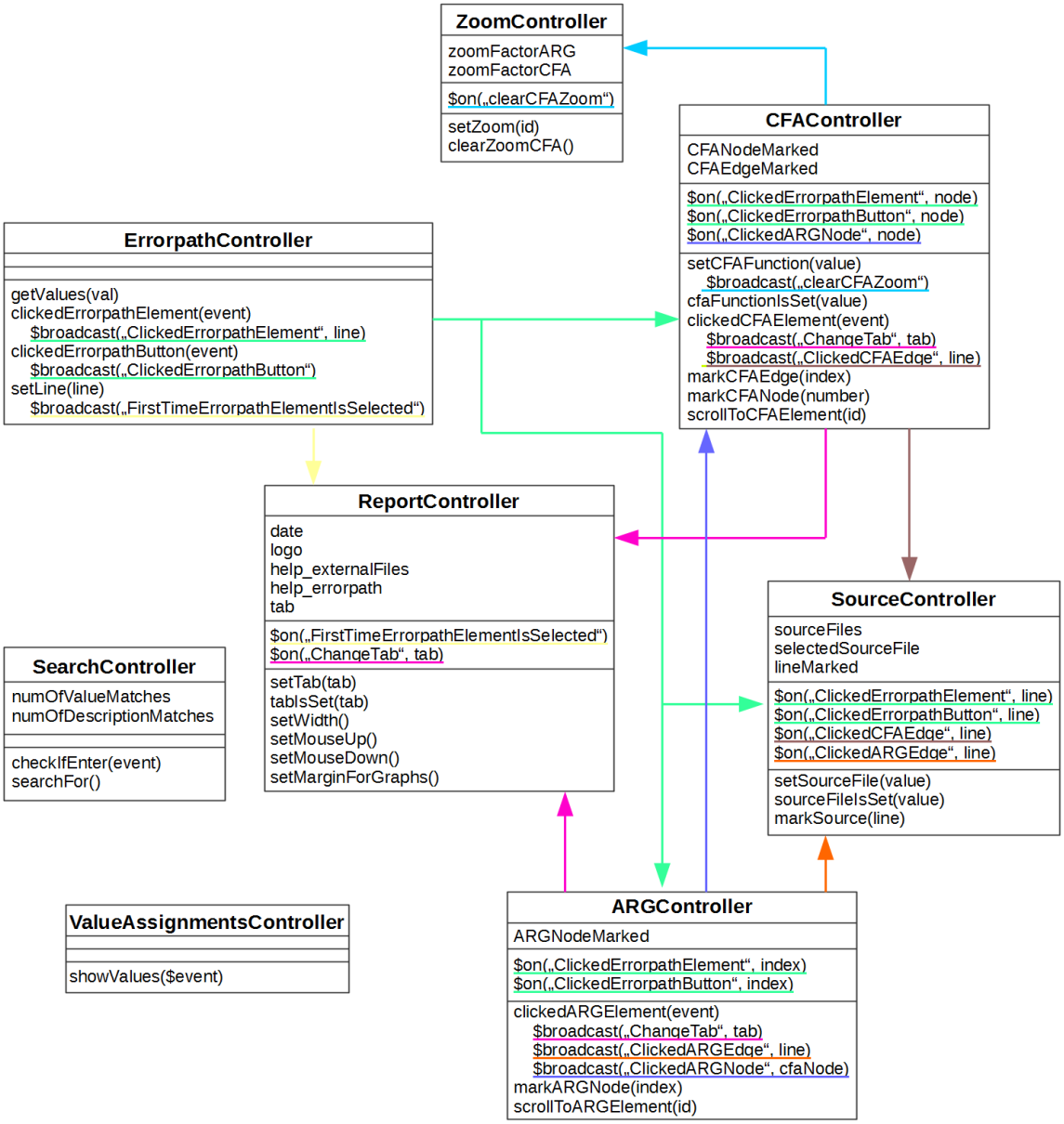


Figure 4.10.: The controllers and their communication

4.3. Problems that remain

- General: One of the purposes of this work was to get one file as counterexample-report. At the moment we have an HTML and a Javascript file. It should be possible to include the Javascript file with <script> into the HTML file without further ado, but it seems like the fact that it includes AngularJS causes problems here (it only works when the file is run out of the IDE WebStorm).

- Graphs: The graphs are still built with “graphviz”, so problems caused by this programme remain (for example that it takes a long time to generate the .svg files from the .dot files for big graphs)

- Errorpath: When navigating through the errorpath with the buttons “Next” and “Prev”, the view of the errorpath does not scroll. So it can happen, that the marked errorpath-line is outside the visible view.

- Value-Assignments: The displayed value-assignments are preprocessed by splitting a string at the “==”-sign. While implementing I was in the belief, that this is the only way value-assignments occur. In fact there are other cases where signs like "<=" or the like occur. In that case the preprocessing would not work anymore.

- Marked Elements: When changing the tab or the displayed CFA, the view does not scroll to the marked element. For example when I navigate through the errorpath (with active CFA-tab) and the next line of the errorpath is displayed in a different CFA than the one that is shown, the CFA changes automatically and the correct edge is marked, but it does not get centred. Or when I click an edge in the ARG, the source-tab automatically appears and the correct line in the source-code is marked, but it does not scroll into view. This is a problem that occured with AngularJS. When clicking the node in the ARG, a javascript-function is invoked. This function initiates all actions that should be done (for example the marking and the centering of the target element). For changing the tab, we only change a variable in the javascript code and AngularJS does the rest (two-way-data binding) – and that is the problem. The tab in the view gets changed after all the javascript functions were executed and the centering of an (at that point still) invisible element does not work. Possible solutions could be setting a timer for the function that is centering the marked element or somehow invoking the centering-function after the target tab is shown.

5. Evaluation

The main purpose of this work was to make the Counterexample-Report more usable, to improve the reachability of the given features, and to add new features that would help the users to work with the Counterexample-Report. To reach this goal, a survey among the current users was held. I did two evaluations: One with the old counterexample-report (before the implementation of the new features) and with the new report (after the implementation of the new features). In this chapter we will see how both surveys were structured, we will have a look at the main results and analyse them. The full range of the (anonymisaed) responses – protocols and all the completed questionnaires – is attached to this work[1] – so everyone will have the opportunity to draw further or different conclusions.

5.1. Before Implementation

Before implementing the new features for the new counterexample-report there was an evaluation of what the users wished for (B First Evaluation (Survey), page 50).

5.1.1. Concept

I used a "form" from Google, that is a data type that you can use for creating a survey. The finished document can be distributed over a link and the responses are automatically collected in a "table" from Google (similar to excel). In this case, the survey was distributed through the CPAchecker mailing-list. The first few questions were meant for categorizing the participants. They asked about the frequency of using the counterexample-report and for how long the participants were working with it. I wanted to find out how familiar the participants were with the topic. The second bunch of questions was about rating the existing solution, the existing

[1] Electronically attached: 1Evaluation_AllResponses, 2Evaluation_AllProtocols (directory)

counterexample-report regarding usability, design and user experience. This information should be used to compare the results with those from the second evaluation. The third and most extensive part asked about critics the participants had and what they would suggest as improvement or possible features they could think of. I also provided several suggestions for possible features that the participants should rate from 1 (in no case) to 5 (absolutely) (B First Evaluation (Survey), page 50).

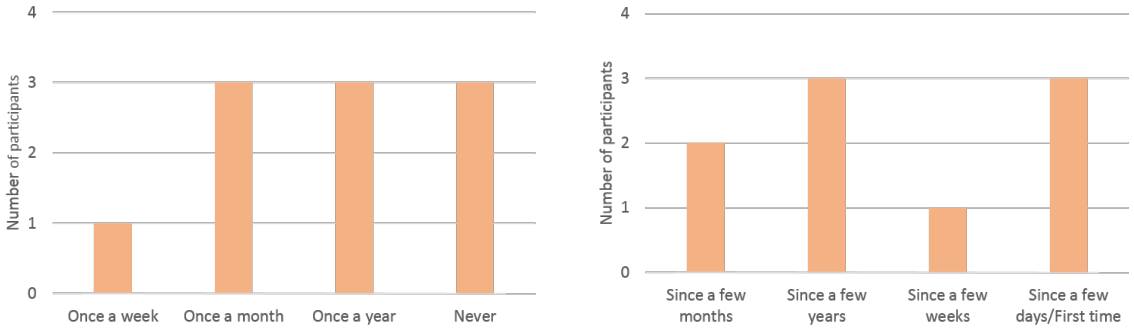
5.1.2. Results

In the following subsection we will discuss the results of the first evaluation.
The survey had 11 participants. On the one hand that is difficult, because it was planned as a quantitative study so we could decide for new features on a broad user base. On the other hand it was not that bad, because even if the study was meant to be quantitative it was qualitative as well. We have to consider that all participants were known as working with CPAchecker for at least a while and so their opinion can be counted as more valuable than the opinion of people who would not have been familiar with CPAchecker or even the field of software verification.

As mentioned earlier, the first three questions were meant to provide an opportunity to categorize the responses. They asked about how long and how frequently the counterexample-report was used by the participant (Fig. 5.1a, Fig. 5.1b). It turned out, that three of the participants have never used the counterexample-report before. One person knew it only since a few weeks, but the other seven participants knew it for months or even years. About the frequency we can say that most participants use it rarely, once a month or once a year. One person uses it once a week and of course the three people that have not seen it before never use it. All participants said that they only use it when they cannot find the error manually or they just said "I use it rare/never". No participant answered that he/she would use it at all times CPAchecker shows an error.
I used the information about the frequency for weighting the results regarding the desired new features (Fig. 5.2).

The results of the second part - Interaction - will be discussed later (5.3 Comparison of old and new counterexample-report, page 42) in direct comparison to the equivalent results of the second evaluation.

The last part - Features of the Counterexample-Report - asked the participants about criticism of the counterexample-report, their assessment of 12 proposed new features, and their own ideas for new features. I collected the results of the partic-



(a) How frequently participants used the counterexample-report

(b) For how long people have used the counterexample-report

Figure 5.1.: Categorization of Participants

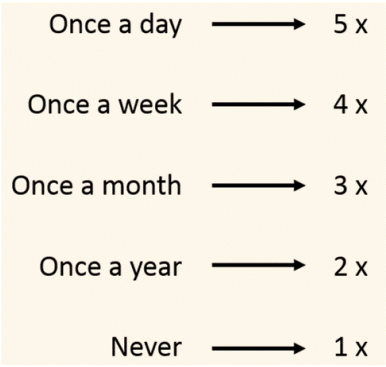


Figure 5.2: I used the information about how frequently participants use the Counterexample-Report for the weighting of the responses about the desired new features.

ipants" criticism and own ideas[2], but as they had no common denominator (never more than two participants had similar answers), I did not include them in the search for new features. Instead, I based it on the rating of the proposed features. I calculated a weighted average rating for each of them.

One question of this part asked about which parts of the counterexample-report the participants used (5.4). The intention behind this specific question was to find out if elements were redundant and to get a feeling for what parts are most important. Since the survey only had 11 participants I did not want to jump to conclusions whether parts of the counterexample-report were redundant or not, but from the result we can see, that the "static" files (statistics, log, configuration) are usually not used and that was at least a confirmation for not working on those parts of the counterexample-report. I did not remove them, but I did not put effort in trying to improve them either.

[2] Electronically attached: Evaluation_1_qualitative.pdf

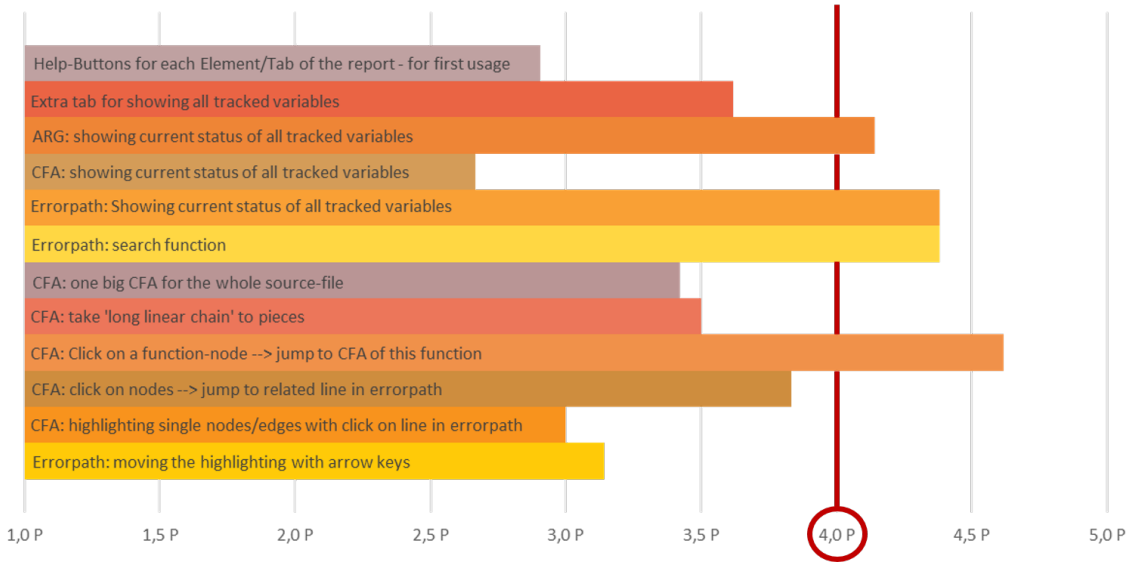


Figure 5.3.: The weighted average rating for the proposed new features - the ones with more than 4 points should get implemented

Weighting	***4***	***3***	***3***	***3***	***2***	***2***	***2***	***1***	***1***	**Total**	***Total (weighted)***
Errorpath (on the left)	1	1	1	1	1	1	1			**7**	***19***
CFA(-Tab)	1	1	1	1	1	1		1		**7**	***18***
ARG(-Tab)	1	1	1	1	1	1		1		**7**	***18***
Click on elements in errorpath --> jump to source/CFA/ARG		1		1	1	1	1	1		**6**	***13***
Source(-Tab)		1	1	1	1				1	**5**	***12***
Click on elements in CFA --> jump to source			1	1		1	1		1	**5**	***11***
Click on elements in ARG --> jump to CFA/source			1	1		1	1			**4**	***10***
Statistics(-Tab)										**0**	**0**
Log(-Tab)										**0**	**0**
Configuration(-Tab)										**0**	**0**

Figure 5.4.: Every participant should name the parts of CPAchecker that he/she uses. The weighting follows the same pattern as that from the new features.

5.1.3. Analysis

In the following subsection we will analyse the results of the first evaluation.
The first questions about how long and how frequently participants use the CR turned out to be problematic. After the evaluation I heard from several people, who started the survey and then stopped after seeing these questions, because they (as not regularly or not at all using the counterexample-report) thought the study did not affect them or they were not its target group. In fact I just wanted to have that information for categorizing, but would have been interested in the opinion of people who never used the counterexample-report before as well.

I used the information about the frequency of the usage of the counterexample-report for weighting the results about desired new features. The idea behind that was, that people who use the report more often could indicate features that would help to improve the report best. If we use something very often, we get a better idea of what features would support our work and what current behaviour might be obstructive to it. To be sure, that the weighted result was stable and did not only occur because of the weighting, I also calculated the average points for each proposed feature without weighting and even with reverse weighting. The result with no weighting was the same as with weighting, of course with different average points for each feature, but the same four features had more than 4 points. Even the results with reverse weighting was nearly the same (again with different average points). Only the feature "Errorpath: search function" did not reach more than 4 points, but it still was the fourth best rated feature. With that stability test we can say, that even if my assumption (that the most frequent user know best what features would improve the counterexample-report), we have selected the four best features to complement the existing counterexample-report.

I made suggestions for new features. This could have inhibited the creativity of the participants when it came to their own recommendations and criticism. Potentially they were dragged in one direction and after already evaluating 10 features they did not want or could not think of more features or suggestions in a different direction. At the end I only took the highest rated features of those I had suggested, because there were not enough matches between the participants within their "free" recommendations or criticism. But like I said before - it is possible that this lack of matches can be put down to the fact that I provided suggestions in the beginning.

5.2. After Implementation

After implementing the new counterexample-report with its new features another survey should evaluate the usefulness of the result and its advantages in comparison to the old counterexample-report.

5.2.1. Concept

It was a combination of a survey similar to the first evaluation and some tasks that the participants should perform. Unlike the first evaluation this one was held as an interview. I created one concept that I followed with every participant of the study. At the beginning were again two questions to categorize the participants: If they were familiar with CPAchecker respectively the old counterexample-report. The next section covered the tasks and for those who were unfamiliar with the old counterexample-report I asked for a short comparison of the old and the new counterexample-report without telling which was old and which was new. After performing the tasks the participants should rate usability, design and user experience after the same scheme as in the first evaluation. At any time of the interview the participants were requested to think aloud. The three main purposes of the second evaluation were:

- Evaluating if the new counterexample-report was better rated than the old one.
- Looking out for spots where user struggle that I did not consider.
- See if user understand difficult spots the same way I did.

5.2.2. Result

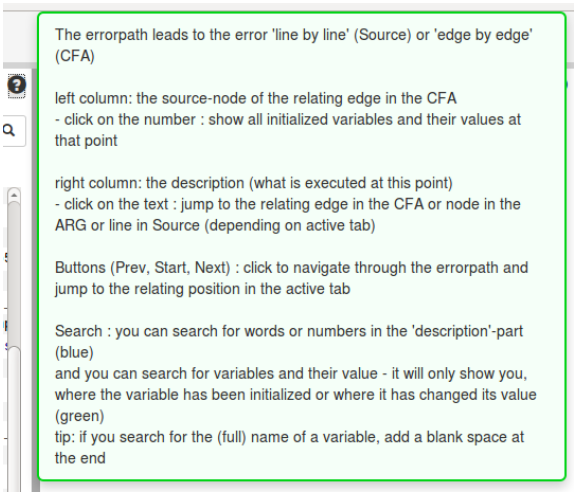
In the following subsection we will discuss the results of the second evaluation.
It had 11 participants and each interview lasted from half an hour up to one and a half hours. This is how the study was planned - it should be more qualitative than quantitative.

Apart from two people every participant has been working with CPAchecker at least a few times and had seen the old counterexample-report before. Therefore, the first task – comparing both counterexample-reports at first glance was not performed in most interviews.

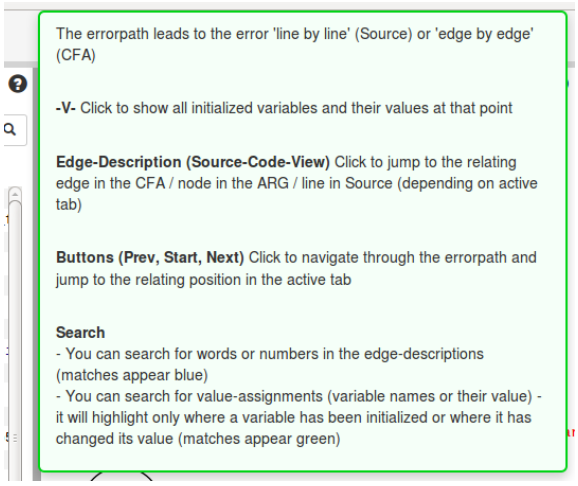
The tasks and their order can be seen in "evaluation2 Second Evaluation (Survey)", page 54. I want to point out the first task. It asked the participants to describe their first impression and "click around", explore the application as far as they want to. Especially those participants who knew the old counterexample-report better, they partially explored every functionality that should be tested in the following tasks. In this case I let them go ahead and took notes about every difficulty they had and every comment they made (they were urgently requested to think aloud). In this case the other tasks were still finished afterwards, but it naturally was very smooth (every task they accidentally performed before was marked). The anonymised protocols of every survey can be found in the annex[3]. Here we will just have a look at those results that led to a change of the implementation, because they were common and fit well into the concept of the new counterexample-report (based on my experience developing it).

- **Help-Texts:** The format was changed (participants criticized that the headings were not highlighted and the format did not look nice) and so were some formulations that were responsible for misunderstandings or were not understood at all (especially the part about the search functionality)(Fig. 5.5).
- I added explanations about the node shapes in the CFA and the colours of some nodes in the ARG to the help texts (A few participants asked about that)(Fig. 5.5).
- **Search Functionality:** The section with the matches is now hidden until the first search is done (with the intensive background-colours they were a real eye-catcher, but people did not immediately know what they were referring to). The words "description" and "variables" are replaced by "edge-description" and "value-assignments" (they were criticized because the users did not understand their meaning). The search can now be initiated with the "return" key (every participant tried this, but it could only be initiated by pressing the search-button) (Fig. 5.6).

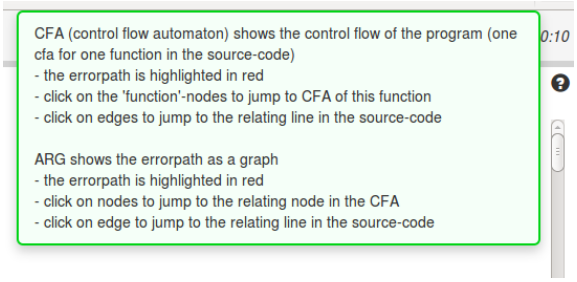
[3] Electronically attached: 1Evaluation_AllResults, 2Evaluation_Protocols (directory)



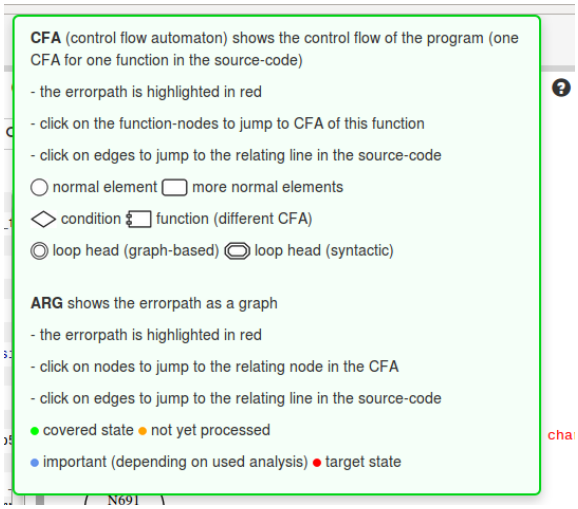
(a) The old help text for the left panel



(b) The new help text for the left panel



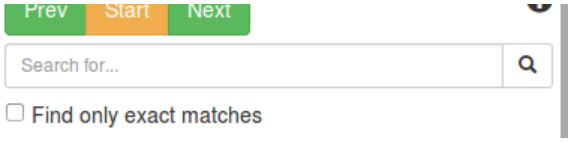
(c) The old help text for the right panel



(d) The new help text for the right panel

Figure 5.5.

(a) The old search functionality

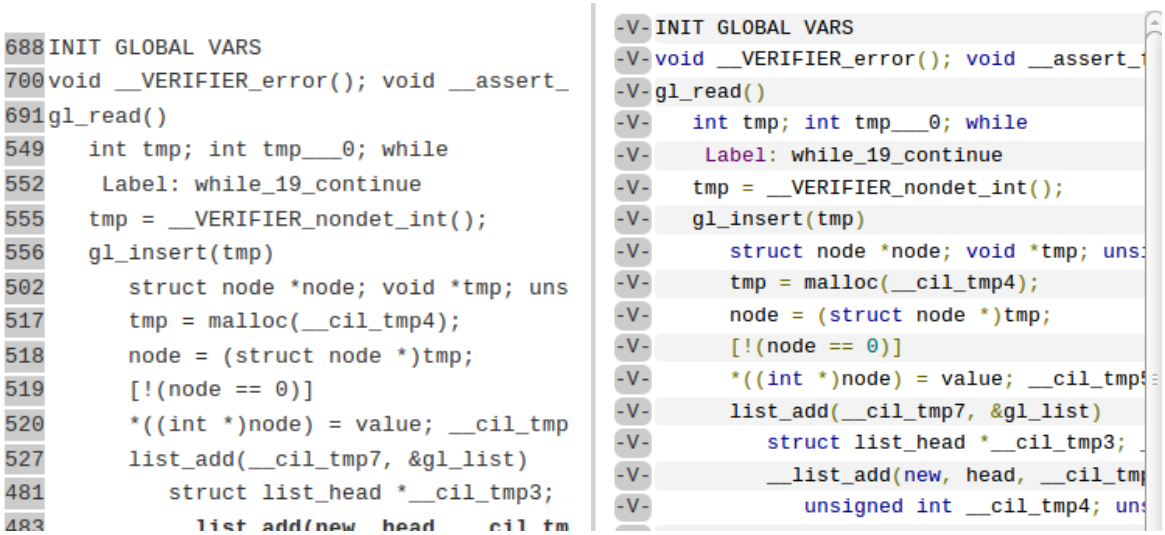


(b) The new search functionality (before the first search)

Figure 5.6.

- **Value-Assignements:** The value-assignments (called "variables") could be seen by clicking on the node-number in front of the line. Almost no participant found this out (they thought you only could click the whole line - that

would jump to CFA/ARG/source). Furthermore, no participant used this node-number, on the contrary some participants erroneously considered that a line number. That is why I decided to remove the node-number and place a "-V-" that is formed like a button instead, hoping it would invite users to click on it (Fig. 5.7).

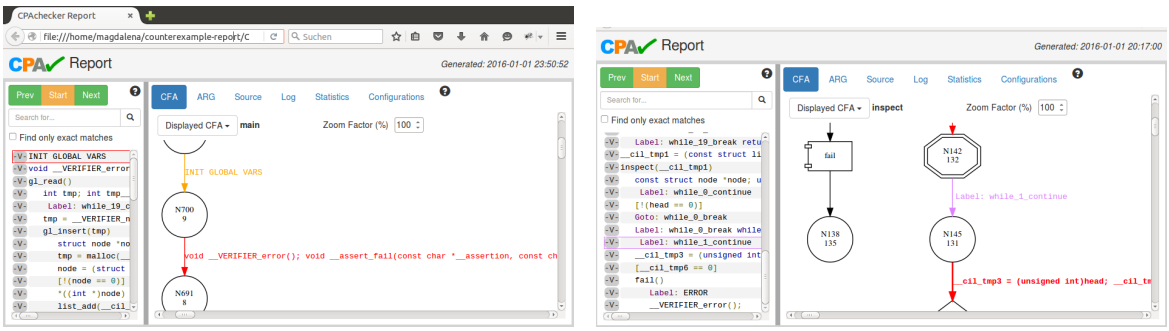


(a) The old Errorpath (b) The new Errorpath

Figure 5.7.

- **Marking:** The marking in CFA/ARG/source was done by highlighting the selected element in orange and scrolling it to the upper left corner. Several participants criticized it should be scrolled to the centre of the view. The orange colour was criticized too: many participants did not recognize it (probably the contrast from the red errorpath was not big enough) and some nodes in the ARG are highlighted in the same colour per se. That is why I changed the marking colour to purple. Purple is not used in any other context and is easy to distinguish from the red errorpath and the colour that are used for node-highlighting in the ARG. I also changed the marking in the errorpath from red to purple, so all markings (errorpath, CFA, ARG, source) have the same colour now (Fig. 5.8).

- **Graphs:** It is possible to jump from the CFA to the source-tab and from the ARG to the CFA- or source-tab. During the evaluation many participants jumped from one to the other tab by accident and were confused and annoyed that they had to manually switch the tabs back.



(a) The old marking in errorpath and CFA (b) The new marking in errorpath and CFA

Figure 5.8.

The first evaluation showed that one of the desired features was to see the value-assignments by hovering over ARG-nodes with the mouse. I did not implement this at first, because I could not find a way to show all instantiated variables in a window without scrolling (but the window would disappear when the mouse is moved from the node into the window to scroll).
These two facts lead to implementing the jump through a double-click: users would not easily do this by accident and the normal click is free for showing the variables in the ARG (not yet implemented).

The last section about the participants' rating of usability, design and user experience will be discussed below – in direct comparison to the results of the first evaluation.

5.2.3. Analysis

In the following subsection we will analyse the results of the second evaluation. That means we will critically review what could have influenced them. The fact that the evaluation had the form of an interview contains some risks, that are even higher because I - the person who implemented the new counterexample-report - was the interviewer. Naturally I have a subjective opinion towards the work and I could have influenced the participants unconsciously. Furthermore, many participants were people I knew before and everybody was aware that we were talking about my work. These facts could have inhibited the participants in criticising it, because they would not want to hurt my feelings.
There is also a positive fact about the survey being held as an interview. In that way I could observe the participants in detail and note things they maybe would

not have mentioned themselves. Especially for the tasks, the fact that I was the interviewer has a positive effect too, as I know how I planned things to work out and I could notice if something was not working the way it was planned.
In summary I would say that the fact that the survey was an interview, held by myself was more positive for the task section, because I could observe the participants and more negative for the section where I asked the participants to rate Usability, Design, and User Experience, because they possibly tried to be nice.

This possibly effects the results even more when we consider that this part is used for comparing the old and the new counterexample-report. The old one was rated anonymously and the users knew they rated a work, that was not done by the person holding the survey.
Another fact to consider is that the participants of the first study were not the same as the participants from the second study, even if some people participated in both studies. That can be seen as an advantage or disadvantage, but it definitely implies the possibility for wrong conclusions.

5.3. Comparison of old and new counterexample-report

In this section we will now compare the results of both evaluations regarding usability, design and user experience (Fig. 5.9).

The following diagrams show the results that were achieved in the first and second evaluation (Fig. 5.10, Fig. 5.11, Fig. 5.12). The average rating of the second report is calculated in three different ways. One time we simply take the average rating of all participants (A), one time we take the average rating only of those participants who knew CPAchecker before (K), and one time we take the average rating of the participants that did not know CPAchecker before (U). This is important for comparability reasons, because the people that rated the old counterexample-report were all familiar with CPAchecker and we can see in the diagrammes, that the rating of people who did not know CPAchecker before is always about 1 point below the average of all participants' rating. We can already see, that the rating for the new counterexample-report is better if we only count the people who knew

1. How would you rate the user-friendliness/usability of the report?

	1	2	3	4	5	
I have to look up/try out the available options everytime.	○	○	○	○	○	*I know/see immediately which options/functions are available.*

2. How would you rate the design of the report?

	1	2	3	4	5	
Not at all attractive.	○	○	○	○	○	*Very attractive.*

3. How would you rate the user-experience of the report?

	1	2	3	4	5	
I don't like working with the report.	○	○	○	○	○	*I really enjoy working with the report.*

Figure 5.9.

CPAchecker and even if we count all participants of the second evaluation it is at least as good as the old counterexample-report.

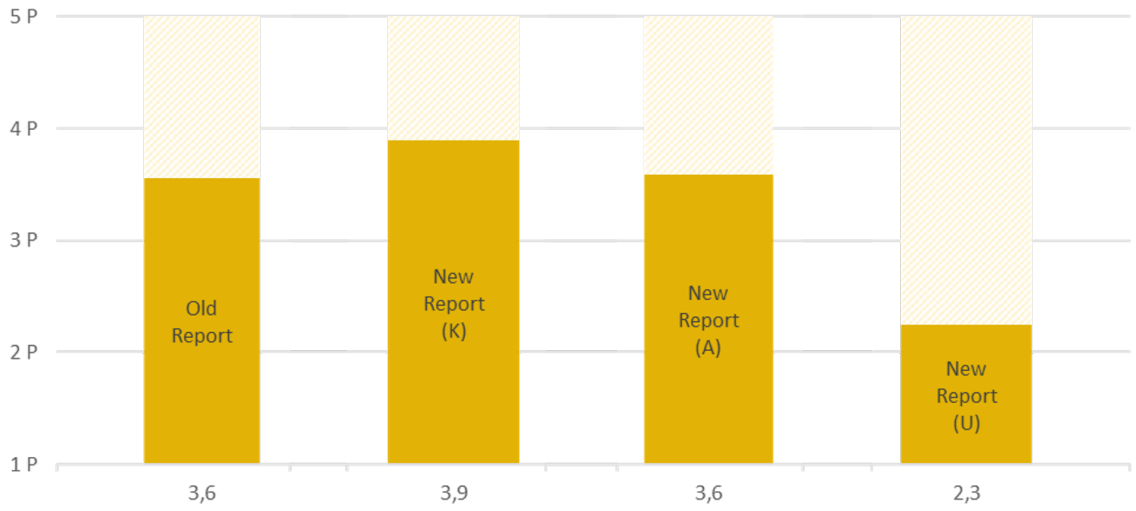


Figure 5.10.: Rating of usability (K = participants who knew CPAchecker, A = all participants, U = participants who did not know)

To see if the result is significantly in a statistical meaning I calculated the p-value. It should show how high the probability is, that this (or a better result) occurs randomly. Each category was rated from 1 point (bad) to 5 points (good). Excluding one person from the first evaluation that did not rate "design" and the two participants from the second evaluation that have never worked with CPAchecker and

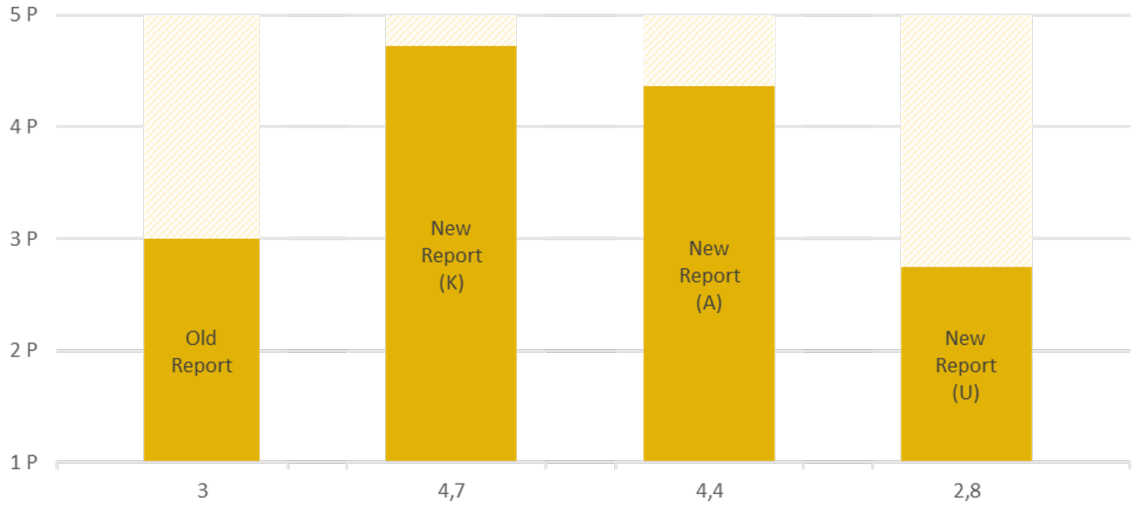


Figure 5.11.: Rating of design (K = participants who knew CPAchecker, A = all participants, U = participants who did not know)

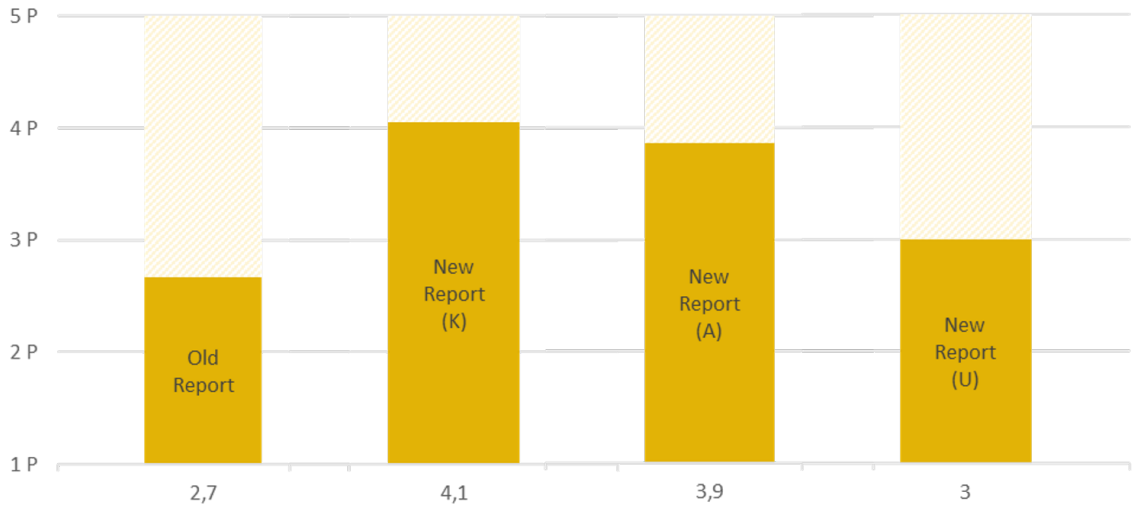


Figure 5.12.: Rating of user experience (K = participants who knew CPAchecker, A = all participants, U = participants who did not know)

had no experience in software verification at all, there were 9 participants from each evaluation left. Because of that we can just sum up all ratings for each category, for old and new report.

old	Usability	**32**	Design	**27**	User Experience	**25**
new	Usability	**35**	Design	**42.5**	User Experience	**36.5**

The sum of the ratings for the old report is subtracted from the sum of the ratings for the new report, so we get the difference between the two ratings.

Usability	Design	UserExperience
+ 3	+ 15,5	+ 11,5

The possible range is [-36,36] with 145 possible values (0.5-steps). With that information we can calculate the probability, that the actually occurred values (or better ones) would occur randomly - the p-value: (Amount of possible values that are as good or better) / (Amount of possible values) = p-value

Usability	Design	User Experience
67 / 145 = 46,20 %	42 / 145 = 28,97 %	50 / 145 = 34,48 %

These results are not below the significance level of 5%. They tell us how big the probability for a result is, that is at least as good as this, when only coincidence would be decisive. For example: to get a result that is 15,5 or better randomly, the probability is 28,97%.

It gets more interesting when we combine the 3 categories to the more general statement “The new report is better than the old one” in terms of “It gets higher rated in each of the categories”. To get this, or a better result in all the categories randomly the probability is 46,20% * 28,97% * 34,48% = 4,6%, which is (with a significance level of 5%) scientifically significant.

However, when we take a step back at the closer look on both evaluations we see a lot of factors that could have influenced our results and as the main feature of this work was to build the new report on a stable architecture and add new features, that is a satisfactory result.

6. Conclusion

At the end of this project we now have a new counterexample-report that provides the same functionality as the old one and more. It is better integrated in CPAchecker, we get a usable report just by running CPAchecker (CFA- and ARG-graphs can be inserted by running an extra Python script). It has a more modern design, it solves some issues that were known about the old report and provides new features that give users more possibilities to work with the data.

6.1. Critical Review of Decisions

With AngularJS I selected a mixture of MVC- and MVVM-Model. Both models prescribe at least a division into view and data. I use an HTML and a Javascript template with key words, where the data should be inserted. Static files like statistics or configurations are inserted into the HTML template. This violates the principle of division into view and data, but several different approaches did not work out (I tried to write the data into the javascript template where the data should be after the MVVM-Model, but the "-signs that occurred in the files caused problems; I tried to integrate these files as <object>, but this conflicts with the goal to have one file at the end; I tried to load these files as extra templates through AngularJS, but again this conflicts with the goal to have one file at the end). I made a compromise and tried to put as little data as possible into the HTML file.

Now to AngularJS itself: I can say that this framework was probably the best choice, but for the usage we do have with the counterexample-report, we cannot use the full power of AngularJS. In the report we do not change the data, but only make changes in the representation of the data. As far as I see, this is not what AngularJS was made for and I often violated one of the principles of AngularJS, not to directly access or change the DOM.

6.2. Outlook/Future Work

During my work there came several ideas to mind. These ideas will be listed here as a motivation for further work:
First it attracted attention that there were 2 DOTBuilder-classes within CPAchecker. It appears that this is not necessary, the DOTBuilder2 does nearly the same and in my opinion it would be a good idea to fuse the classes to one.
Second, it is a known problem that for very big graphs the graphviz-program does not work or takes too long to bear. In the first evaluation some participants mentioned that they would like to work with the counterexample-report when they have big ARG-graphs, with small graphs they do not need the report. So this would be another point that deserves some attraction. The approach that I think of would be to write a special program that creates the .svg for the counterexample-report so we do not need "graphviz" anymore.
The second evaluation showed that many users would like to have a back- functionality, so they could jump back to their last position or their last open tab. We could analyze how such a functionality should be realized within the counterexample-report (should it just be for tab-changes or for every state of the report, for example every step when navigating through the errorpath) and then add this functionality to the back-button of the browser or introduce a shortcut. As a last proposal I would like to mention an introducing tutorial. Users could have the option to be guided through all the functionalities of the counterexample-report in a video-like tutorial. Like I mentioned before I had two participants in the second evaluation that never used the counterexample-report or even CPAchecker before. They had really big problems even to understand what data was shown to them, let alone understand what they could do with the data. Such a tutorial could be a possibility to smooth the way of understanding counterexamples for people who have not touched the topic of software verification before.

A. Data from CPAchecker

JSON-Data CPAchecker provides different .json files with information about the errorpath, its connections to CFA/ARG/source, and information about the CFA-graphs.
There are the following files: “cfaino.json”, “fcalledges.json”, “combinednodes.json”, and “ErrorPath.*.json” (* stands for a number: as we can have more than one errorpath, there is one json-file for each errorpath).
“Errorpath.*.json” contains an array filled with objects. Each object stands for an edge in the CFA/ARG and has the following keys:

val	CPAchecker tracks certain variables, when one or more of the variables are instantiated or change their value it is noted here
desc	the line of code that is executed at this point (label of the edge in CFA and ARG)
source	the source-node of the edge (“desc”) in the CFA
file	the sourcefile where the line of code (“desc”) appears (one errorpath can cover more than one sourcefile)
target	the target-node of the edge (“desc”) in the CFA
line	the related line in the sourcefile
argelem	the source-node of the edge (“desc”) in the ARG

“combinednodes.json” contains all nodes that are taken together in one node (because they only contain “normal” edges (see “2 Existing Solution”, page 9). It contains an object and every key is a node-number that is in such a multistatement-node and its value is the first node of this multistatement-node. “fcallEdges.json” contains all nodes that come directly before or after a function-node. It is an object, too: its keys are the nodes before a function-node and the value is an array with the number of the function-node at index 0 and the number of the node after the function-node at index 1. “cfainfo.json” - as the name says contains information about the CFA-graphs. It consists of an object with two keys: “edges” and “nodes”.

Their values are objects themselves with the node- or edge-number as keys. The values of the edge-numbers are objects with the following keys:

source	the number of the source-node
file	the name of the source-file (can be more than just one)
stmt	the content of the line of code
target	the number of the target-node
line	the line where you find this text in the sourcefile
type	the type of the edge, for example "Blank Edge" or "Multiple Edge"

The values of the node-numbers are objects with these two keys:

no	the number of the node (pointless as this is the key of the parent object)
func	the function where this node appears

At the moment ARGStatistics.java is writing the .json data in the output directory. As they only are required for the report, this would be unnecessary if they would be written directly into the counterexample-template.

Sourcefile The source-file and its path are defined by the user when he/she performs an analysis with the CPAchecker.

Log, Statitics and Configuration-Properties These files are written into the output directory by CPAchecker with fixed names (that could be changed through user-configuration).

Graphs They are generated by the programme "graphviz" out of the .dot files that are generated in DOTBuilder2.java. The CFA-nodes have globally unique nodenumbers (which can appear multiple times within one errorpath). The ARG-nodes have nodenumbers that are unique within the ARG-graph (as the ARG-graph shows the errorpath, the CFA-node-numbers eventually wouldn't be unique).

B. First Evaluation (Survey)

Usage of the counterexample-report

1. On which occasions do you use the counterexample-report?
   - ☐ Everytime CPAchecker shows an error
   - ☐ When I cannot find the error "manually"
   - ☐ Rare/Never
2. How often do you use the counterexample-report?
   - ○ Once a day
   - ○ Once a week
   - ○ Once a month
   - ○ Once a year
   - ○ Never
   - ○ _____
3. Since how long do you use the counterexample-report?
   - ○ Since a few days
   - ○ Since a few weeks
   - ○ Since a few months
   - ○ Since a few years
   - ○ Since more than 5 years
   - ○ _____

Interaction

1. How would you rate the user-friendliness/usability of the report?

	1	*2*	*3*	*4*	*5*	
I have to look up/try out the available options everytime.	○	○	○	○	○	*I know/see immediately which options/functions are available.*

2. How would you rate the design of the report?

	1	*2*	*3*	*4*	*5*	
Not at all attractive.	○	○	○	○	○	*Very attractive.*

3. How would you rate the user-experience of the report?

	1	*2*	*3*	*4*	*5*	
I don not like working with the report.	○	○	○	○	○	*I really enjoy working with the report.*

Features of the Counterexample-Report I would like to know which of the features/files in the report you use right now and which features you would like to have in addition: I made some proposals that you can evaluate and at the end there is space for your own ideas/proposals.

1. Which of the existing features do you use at the moment?
   How exactly do you use the Counterexample-Report?
   - ☐ Configuration(-Tab)
   - ☐ Statistics(-Tab)
   - ☐ Log(-Tab)
   - ☐ Source(-Tab)
   - ☐ ARG(-Tab)
   - ☐ CFA(-Tab)
   - ☐ Errorpath (on the left)
   - ☐ Click on elements in ARG → jump to CFA/source
   - ☐ Click on elements in CFA → jump to source
   - ☐ Click on elements in errorpath → jump to source/CFA/ARG
   - ☐ _____

2. Are there features that do not work right or is there something about the report that is bothering you?

3. Possible Feature
   Errorpath: moving the highlighting with arrow keys (not only with buttons "prev", "next", "start")

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

4. Possible Feature
   CFA: highlighting single nodes/edges with click on line in errorpath (not highlighting the whole path)

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

5. Possible Feature
   CFA: click on nodes → jump to related line in errorpath

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

6. Possible Feature
   CFA: Click on a function-node → jump to CFA of this function

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

7. Possible Feature
   CFA: take "long linear chain" to pieces

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

8. Possible Feature
   CFA: one big CFA for the whole source-file (CFAs for several functions appear as nodes)

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

9. Possible Feature
   Errorpath: search function (when I want to search for appearance of a special variable

for example)

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

10. Possible Feature
    Errorpath: Showing current status of all tracked variables when hovering over a line

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

11. Possible Feature
    CFA: showing current status of all tracked variables when hovering over a node

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

12. Possible Feature
    ARG: showing current status of all tracked variables when hovering over a node

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

13. Possible Feature
    Extra tab (besides statistics, log, ...) for showing the change of the tracked variables and where this change takes place

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

14. Possible Feature
    Help-Buttons for each Element/Tab of the report - for first usage (can be switched off)

	1	*2*	*3*	*4*	*5*	
In no case.	○	○	○	○	○	*Absolutely.*

15. Are there other features you would like to have?
    Please mark: "5 - Urgent", "4 - Would be desirable", "3 - Would be a nice feature"

C. Second Evaluation (Survey)

1. Sind Sie mit CPAchecker vertraut?
2. Sind Sie mit dem (alten) Counterexample-Report vertraut?
3. TASKS
4. Wie würden Sie die Nutzerfreundlichkeit/Usability bewerten?

	1	*2*	*3*	*4*	*5*	
Ich muss ständig ausprobieren, welche Funktionen wo sind. Die Benutzung ist unintuitiv.	○	○	○	○	○	*Ich sehe sofort welche Optionen es gibt. Die Benutzung ist intuitiv. Es erfordert keinen großen Aufwand um die Funktionalität zu verstehen.*

5. Wie würden Sie das Design bewerten?

	1	*2*	*3*	*4*	*5*	
Gar nicht ansprechend.	○	○	○	○	○	*Sehr ansprechend.*

6. Wie würden Sie die User-Experience bewerten?

	1	*2*	*3*	*4*	*5*	
Die Arbeit mit dem Report ist sehr unangenehm.	○	○	○	○	○	*Die Arbeit mit dem Report macht mir Spaß.*

TASKS

1. Vergleichsbewertung (nur für die User, die den alten Report nicht kennen)
   a) Womit würden Sie lieber arbeiten? Warum?
   b) Wie würden Sie das jeweilige Design bewerten?

	Gar nicht ansprechend.	*1*	*2*	*3*	*4*	*5*	*Sehr ansprechend.*
neu		○	○	○	○	○	
alt		○	○	○	○	○	

2. Tasks

a) Erster Eindruck/Rumprobieren *Was versuchen Nutzer als Erstes? Was sticht positiv/negativ ins Auge?*

b) Was ist ein CFA? *Findet der Teilnehmer den (richtigen) Hilfe-Button?*

c) Verfolgen Sie den Fehlerpfad (errorpath) im CFA. *Versteht der Teilnehmer, dass der Fehlerpfad rot markiert ist? Nutzt er das neue Feature „Klick auf Funktionsknoten → Sprung in CFA der Funktion“?*

d) Finden Sie eine Kante des CFA (die 2.Kante im CFA von „read()“) in dem Source-Tab. *Nutzt der Teilnehmer den Klick auf die Kante?*

e) Stellen Sie den CFA von „gl_read()“ so dar, dass man den gesamten Graph sehen kann. *Kommt der Teilnehmer mit der Zoom-Funktionalität zurecht?*

f) Finden Sie eine Stelle im Errorpath (Kante aus 455) im Source-, CFA- und ARG-Tab. *Versteht der Teilnehmer, dass er nach einem Tab-Wechsel nochmal klicken muss um zur richtigen Stelle zu springen?*

g) Welche Variablen sind an dieser Stelle instanziiert? *Findet der Teilnehmer die Anzeige der Variablen? Versteht er den zugehörigen Hilfstext?*

h) Wie oft wurde die Variable „__cil_tmp7“ verändert? Und an welchen Stellen? *Kommt der Teilnehmer mit der Suchfunktion zurecht? Versteht er den zugehörigen Hilfstext?*

i) Finden Sie den Knoten 54@549 aus dem ARG im CFA-Tab. *Klickt der Teilnehmer den Knoten?*

D. Documentation of Source-Code

D.1. Manual

To generate the counterexample-report without graphs (CFA/ARG), just run CPAchecker. In the directory “Counterexample_Report” one “report_withoutGraphs _*.html” for each errorpath (with * is the number of the errorpath, starting at “0”) is generated. If the analysis did not find an error (and therefore no errorpath exists) one file “report_withoutGraphs.html” will be generated.
We will also get one “app_*.js” for each errorpath (respectively “app.js” if there is no errorpath) in the directory “Counterexample_Report/app”. By running the Python script “generate-report-with-graphs.py” (in the directory “Counterexample_Report”), one “report_*.html” is generated for each “report_withoutGraphs_*.html (respectively a “report.html” for “report_withoutGraphs.html”) in the same directory. The libraries that are used are received from the internet. We need Google’s “prettify.js” and “prettify.css” for the syntax highlighting of the errorpath and the source-code, “angular.min.js”, “bootstrap.min.js” and “bootstrap.min.css”, and Bootstrap requires “jquery.min.js”.

D.2. Further Explanations

CPAchecker The class “GenerateReportWithoutGraphs.java” (src/org/sosy_lab/cpachecker/core/counterexample) writes all the necessary data directly into the HTML- and Javascript-template.
The process is getting invoked within the class "MainCPAStatistics.java" (src/org/-sosy_lab/cpachecker/core) in the function “printStatistics(PrintStream out, Result result, ReachedSet reached)”. It has to import “org/sosy_lab/cpachecker/core/counterexample/GenerateReportWithoutGraphs” and the Option “newCounterexampleReport” has to be set “true”.
The counterexample-report templates provide key words (commented out) and it

expects CPAchecker to insert the data with the correct HTML/Javascript syntax.
The name of the Javascript file, the source file, the log file, the statistics file, and configuration file are inserted directly into the HTML-file. The source file has to be inserted as table, the other files are written line by line as single elements.
All other relevant data is written into the Javascript template as object or array.
The inserting of all these files is realized as follows: The template is read in line by line and not cached, but immediately written into the output file. It interrupts as soon as a line contains a key word (like “SOURCEFILE”), then reads the according file (which is also not cached, but written in the output file immediately).

Python The CFA and ARG graphs are included via a Python script "generate-report-with-graphs.py". At first, the script generates .svg files from the .dot files. The .svg file are then inserted into each “report_withoutGraphs_*.html”. As .svg files they already have the correct syntax for a <svg>-element in HTML, but the script also has to insert the ng-click directives that are needed for the clickability of the graph-edges and -nodes.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, den 6. Januar 2016

_____

Magdalena Murr