INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Data Visualization in ProB

Joy Clark

Bachelorarbeit

Beginn der Arbeit:	14. März 2013
Abgabe der Arbeit:	14. Juni 2013
Gutachter:	Prof. Dr. Michael Leuschel
	Prof. Dr. Frank Gurski

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 14. Juni 2013 _____

Joy Clark

Abstract

Hier kommt eine ca. einseitige Zusammenfassung der Arbeit rein.

Contents

1 Introduction

The introduction.

2 Background

2.1 B-Method

The B-Method is a method of software development that uses the Classical B specification language to develop software. Classical B is a formalism that uses the concepts of set theory, logic, and numbers as a mathematical basis, and the concept of an Abstract Machine in order to model the behavior of a system.

Define state, operation, variable, formula, invariant, etc.

[INSERT CITATION] defines an Abstract Machine as:

> Abstract Machines [...] are machines that encapsulate:
> **state** consisting of a set of variables constrained by an invariant
> **operations** operations may change the state, while maintaining the invariant, and may return a sequence of results.

2.2 ProB

ProB is a software developed in order to animate and check models written in the Classical B, Event-B, CSP-M, TLA+, and Z specification languages. The software is written in the Prolog language, which is a logical programming language. There is a standalone version of the ProB software available with the graphical user interface written in Tcl/Tk. A binary command-line interface is also available for the software.

Two different projects are underway to create Java based APIs for the ProB CLI. The first version of the Java interface began in 2006. This version integrated the exiting ProB CLI into the Rodin platform.

Planning for the ProB 2.0 API began in 2011. The main goal of the ProB 2.0 API was to adapt and optimize the existing Java API to build a user interface on top of a programmatic API. Functional programming techniques were used in the development of the software as much as possible. To meet these ends, the Groovy scripting language was heavily integrated into to the ProB 2.0 core. The ProB 2.0 API includes a fully functional webserver with servlets that allow the extension of the Java core into JavaScript and HTML. The fully functional webconsole available in the API makes use of this technology.

2.3 d3 and JavaScript

The web console interface in the ProB 2.0 API is written JavaScript and HTML. Since the web server structure is already available in the Java API, it made sense to develop data visualizations that could make use of the same structure.

D3 (Data-Driven Documents) is an open source JavaScript library that was developed for the visualization of different types of data. The output of the D3 functions is a pure SVG and HTML document object model that implements the W3C Selectors API. Because of this, the entire docmument can be styled using CSS and the developer has more control over how the resulting visualizations appear. Particular elements from within the DOM can also be selected using JQuery.

D3 includes support for visualizing both simple datasets and datasets that are more complex. Examples for simple data visualization include pie charts, line graphs, and bar graphs. More complex data structures such as the graph and tree data structures can be represented using the spring and tree layouts respectively.

3 Motivation

The ProB standalone application written in Tcl/Tk uses Dotty as library to create visualizations of certain data structures within the ProB application. Unfortunately, these data visualizations are completely missing in both the Java APIs. The intention of this work is to inspect the data visualizations that are available in the Tcl/Tk application and recreate these in the Java 2.0 API. The visualizations should not be hard coded, but should use D3 and the existing webserver structure to create a framework so that similar visualizations can be created in the future using the same principles.

Central to the ProB application is the concept of the state space. The state space is a directed multigraph. The states are saved as vertices in the graph and the operations within the graph are saved as directed edges that transition from one state to another. The main purpose of the ProB software is to verify this state space for inconsistencies. For instance, it is possible to use ProB to find states within the graph that violate the invariant for specification. It is also possible to find states from which there are no further operations possible. This is called a deadlock.

The ProB 2.0 API extracts the information about the existing state space from the ProB CLI and saves it in a programmatic abstraction of the state space. This abstraction saves the information about the different states in a graph data structure using the Java JUNG graph library. The state space object already supports the use of Dijkstra's algorithm to find the shortest trace from the root state to a user defined state. This can be used to find traces that show how an invariant violation or deadlock can be found. What is missing, however, is a visualization of the actual state space itself.

Because the state space is a directed multigraph, this visualization problem is not trivial. It was necessary to find a graph library that would be able to draw a complicated graph. Because the state space varies drastically depending on the machine that is being animated, it was also necessary that the graph library be able to handle graphs of all different shapes and sizes.

A useful feature for the visualization of a state space would also be the ability of the user to manipulate the graph. For instance, the Tcl/Tk version of ProB supports the capability for the user to specify a formula and to merge all states for which the formula evaluates to the same result. Similar functionality was desired for the visualization in the ProB 2.0

API. A useful visualization would also allow the user to specify how the graph should be colored.

Although the visualization of the state space was the focus of this work, there were other sets of data for which a visualization would be useful. The ProB Tcl/Tk version supports a useful visualization of B formulas. The user specified formula is broken down into subformulas and colored so as to specify the value of the formula (e.g. if a given predicate evaluates to true at the specified state, the predicate would be colored green). A similar visualization exists in the ProB 1.3.6 API but not in the ProB 2.0 API.

Another useful visualization that falls into the scope of this work was a visualization of the value of a user defined formula over time. No such visualization exists in any of the ProB applications yet, but it was thought that such a visualization would be relatively simple to generate and useful.

4 Contribution

4.1 Visualization of the State Space

During the preliminary experiments for State Space visualization, several different graph libraries were tested out. D3 was chosen because it could process graphs of relatively large size in a way that was eye pleasing for users. Visualization of the state space uses the Spring layout that is available from the D3 library. Unfortunately, when visualizing state spaces with a very large amount of states and inputting them all at the default intitial position, it took rather long for a good visualization to emerge. Therefore, the FRLayout from the JUNG graph library was used as a static rendering engine to calculate the ideal initial positions for the states to be visualized in the graph.

One of the main problems with the web framework that was discovered at the start of the development process was the problem of how different state spaces should be visualized at the same time. The ProB 2.0 API supports the animation of multiple state spaces at any given time. When a state space visualization is created, it is created using the state space that is currently being animated. When the animation is switched, a new state space visualization can be created using the new state space that is being animated. The problem is that a state space visualization is not static. Since the state space that is being visualized changes over time when states are added into the graph, the visualization also needs to adapt and grow correspondingly. The solution to this is to have the instance of the state space visualization poll the state space regularly to get any new states that have been discovered. The problem occured because the servlet responsible for dealing with the state space was static. When the polling occured, the servlet did not know which instance of the state space was supposed to be polled.

4.2 Visualization of the Value of a Formula Over Time

During animation in the ProB 2.0 API, the animation steps that have been taken are saved in a trace. A particular formula can take on different values over the course of a trace. In

the implementation of the B state space, a particular state is determined by the different values that a variable takes on. Therefore it is particularly interesting to be able to examine the value of a variable over the course of a trace when dealing with a Classical B or Event B formula.

Evaluation of a formula for takes place in the ProB prolog core. In the ProB 2.0 API, a feature was implemented that takes the list of all the states that the trace covers and a particular formula and returns the list of the values that the formula takes on in the course of the trace. This feature was used in order to create a line plot of the values that the formula takes on. This works for all formulas that take on either an integer value or a boolean value (IMPLEMENT BOOLEAN VALUE).

4.3 Visualization of a formula

The ProB CLI already supported the functionality of expanding a formula into its subformulas and finding its value at a given state. However, this functionality would only return the subformulas that were directly beneath the desired formula. For the visualization, it was desired that a given formula could be completely expanded and evaluated and then sent to the ProB 2.0 API. This would improve the performance of the process.

5 Related Work

d3, Alloy, etc.

6 Conclusion

The Conclusion

7 Future Work

Future work

List of Figures

List of Tables