INSTITUT FÜR INFORMATIK
Softwaretechnik und
Programmiersprachen

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:	Dr. Michael Leuschel
	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 ProB Tcl/Tk application uses the DOTTY graph layout tool to create visualizations of data structures within the ProB application. Unfortunately, these data visualizations are completely missing in both the Java applications. In this work, we intend to inspect the data visualizations that are available in the Tcl/Tk application and recreate them in the Java 2.0 API. The visualizations should all use D3 and the existing webserver structure. After we are done implementing a few chosen visualizations, it should be possible to create similar visualizations using the principles that we have used.

The concept of the state space is central to the ProB application. 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.

1.1 Planned Visualizations

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 because the algorithm for drawing graphs is not trivial. We had to find a graph library that would be able to draw a such a 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. We chose D3 because it met all of these requirements.

We also wanted the user to be able to manipulate and interact with the graph. For instance, the ProB CLI supports the ability to create smaller graphs that are derived from the original state space [LT05]. Because state spaces become so large so quickly, a derived graph can be much more meaningful and useful for a user. One of the features that we want to implement for the state space visualization is the ability for the user to apply these algorithms to their state space to simplify its representation. The calculation of these algorithms needs to take place in the ProB CLI, but a seamless transfer between the two graphs should take place from within the visualization.

Although the visualization of the state space is the focus of this work, there are other sets of data that need to be visualized. The ProB Tcl/Tk version supports a useful visualization of B formulas [LSBL08]. 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 Plugin, but not in ProB 2.0.

We also wanted to create a visualization of the value of a user defined formula over the course of an animation. No such visualization exists in any of the ProB applications yet, but it is especially useful for visualizing cosimulation and continuous models.

1.2 Integration into the ProB 2.0 Eclipse Plugin

Because ProB 2.0 is an Eclipse plugin, it is necessary to be able to integrate any visualization into the framework. Because D3 is JavaScript based, we will create Eclipse views that contain a browser displaying the visualization that we create. A similar technique has already been used to create the groovy console view in the ProB 2.0 application.

It is also necessary that the Java servlets responsible for generating and manipulating the visualizations can manage several different visualizations at once. We have not integrated any server side language into our webserver, so the visualizations need to consist only of static html pages and the JavaScript programs that are responsible for generating the visualization. Data will not be able to be sent to any of the visualizations directly. Instead, the JavaScript scripts will have to set up an interval to poll the servlet and ask for any changes in their visualization.

It should also be possible for the user to edit the visualization. Using D3 selectors, it is possible to change the style of D3 visualizations. We should find a way to lift this functionality from the JavaScript level into the Java application so the user has as much control over the generated visualization as possible.

During the course of this paper, the different tools and concepts that were necessary in the scope of this work will be introduced. Then the motivation and the requirements for the desired visualizations will be described in detail. The actual visualizations that were created will then be presented, followed by further ideas for implementations and related work.

2 Background

2.1 B-Method

The B-Method is a method of specifying and designing software systems that was originally created by J.R. Abrial [AAH05]. Central to the B-Method are the concepts of *abstract machines* that specify how a system should function [Sch01].

An *abstract machine* describes how a particular component should work. In order to do this, the machine specifies *operations* which describe how the machine should work. An *operation* can change the *state* of the machine. A *state* is a set of *variables* that are constrained by an *invariant*. In order for the model to be correct, the *invariant* must evaluate to true for every state in the model.

Two specification languages used within the scope of the B-Method are Classical B and

Event-B. There are several tools available to check and verify these specification languages [LIST TOOLS].

2.2 ProB

ProB is a tool created to verify formal specifications [LB08]. In addition to verifying Classical B and Event-B specifications, ProB also verifies models written in the CSP-M, TLA+, and Z specification languages. ProB differs from other tools dealing with model verification in that it is fully automated. Several tools have also been written to extend ProB and add functionality.

ProB verifies models through consistency checking [LB03] and refinement checking [LB05]. Conistency checking is the systematic check of all states within a particular specification. In order to do this, ProB checks the state space of the specification in question. The state space is a graph with *states* saved as vertices and *operations* saved as edges. Refinement checking examines the refinements of a machine to ensure that they are valid refinements.

2.2.1 ProB Tools

ProB consists of several different tools that will be referenced throughout the course of this work.

ProB CLI
: The ProB kernel is written in primarily in SICStus prolog [LB08]. The ProB CLI is available as a binary executable, and all of the other tools listed here are built on top of it. The ProB CLI provides support for the interpretation of Classical B, Event-B, CSP-M, TLA+, and Z specification languages. These specifications are interpreted and translated into an internal represenation that can then be animated and model checked.

ProB Tcl/Tk
: The ProB Tcl/Tk was created at the same time as the ProB CLI to provide a user interface for the ProB CLI. In addition to providing a UI for the ProB CLI, ProB Tcl/Tk also enables the user to edit specification files before animation, and provides the user with visualizations of different data that is generated during animation or model checking. For instance, ProB Tcl/Tk includes graphical visualizations for the state space, the current state, and for B predicates that are evaluated at the current state in the animation [LSBL08] The ProB Tcl/Tk application uses the DOTTY tool available from the Graphviz graph layout software.

ProB Plugin
: Work on the ProB Plugin began in 2005. This is an Eclipse plugin for the Rodin software [BH07], which is an easy to use and extensible tool platform for editing specifications written in the Event-B specification language. At this point, a socket server was integrated into the ProB CLI which allowed the ProB Plugin, which is

written in Java, to communicate with the ProB CLI. The communication between the two takes place using queries and answers.

ProB 2.0 API

Development of 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. One of the main improvements made available in this tool was the introduction of a programmatic abstraction of the state space. The ProB 2.0 API also provides a programmatic abstraction called a *history* for the represenation of animations. This history consists of the trace of operations that have been executed for a given animation. A given state space can have an arbitrary number of animations. The user can switch between animations and work on any given animation at any given time. Thus, for the ProB 2.0 API, the notion of a *current state* corresponds to the current states of the animation that the user is currently executing.

During the course of animation and model checking, the state of the ProB 2.0 API changes. The state space grows (i.e. new operations and states are cached), and the current state changes. In order for developers to be aware of these changes, a listener framework is offered. A developer can therefore implement classes that react when the current state in the animation changes, when new states are added to the state space, and when the specification that is being animated changes.

The API is programmatic and harnesses the power of the Groovy programming language. The ProB 2.0 API integrates a fully functioning groovy console into the final product. It is now possible for users and developers to write Groovy scripts that carry out desired functionality. There is also support for creating web applications that communicate with ProB. The console is actually a user interface that makes use of the jetty server that is integrated in the ProB 2.0 API. There are no eclipse dependencies present in the ProB 2.0 API, so it can be deployed as a jar file and integrated into any Java based application.

ProB 2.0 Plugin

Similar to the original ProB Plugin, the ProB 2.0 Plugin is an Eclipse plugin created for the Rodin platform. Much of the UI code has been directly imported from the original Plugin, so it appears to be very similar. However, the graphical interface is now built on top of the new programmatic abstractions that are available from the ProB 2.0 API, and changes that take place within the graphical interface are triggered by the ProB 2.0 API listener framework.

2.3 D3 and JavaScript

Since a jetty server was already available in the ProB 2.0 Plugin, it was plausible to create visualizations using javascript and HTML. Because the ProB 2.0 Plugin is an Eclipse application, it also would have been possible to create visualizations using a native Java or Eclipse library. I carried out an experiment at the beginning of this work to determine the feasibility of the different graph libraries. JUNG was considered because it is the software framework that is the ProB 2.0 API currently uses. It would have been relatively simple to embed the visualizations into the existing ProB 2.0 Plugin, but customizing

JUNG graphs is extremely difficult. The ZEST graph library was a feasible option, but in the end, I chose to use the D3 library.

D3 (Data-Driven Documents) is "an embedded domain-specific language for transforming the document object model based on data" which is written in JavaScript [BOH11]. Developers can embed the library into a JavaScript application and use the D3 functions to create a pure SVG and HTML document object model (DOM). The focus of D3 is not on creating data visualizations. It is on providing the user the capability of defining exactly which elements the DOM should contain based on the data that the user has provided. Because the objects that are being manipulated are pure SVG and HTML, the user can use D3 to create objects that can be styled using CSS or by dynamically manipulating the style tags of the elements.

2.3.1 Core Functionality

D3 provides a selector API based on CSS3 that is similar to jQuery [1]. The user creates visualizations by selecting sections of the document and binding them to user provided data in the form of an array of arbitrary values [BOH11]. D3 provides support for parsing JSON, XML, HTML, CSV, and TSV files. Once the data is bound to the desired section of the document, D3 can append an HTML or SVG element onto the section for each element of data. This is where the real power of D3 lies because the user can define the attributes of the element dynamically based on the values of the datum in question. By changing these attributes (e.g. size, radius, color) the resulting document already presents the data in a way that the viewer visually understands. The core also provides support for working with arrays and for defining transitions that can be used to animate the document. In order to better understand how D3 works, we have provided a simple example of a how a developer can use D3 to create an HTML dropdown menu (see Listing 1). The generated HTML snippet is also provided (see Listing 2). A more complicated example using the force layout is available in the appendix (see Appendix A).

2.3.2 Further Functionality

D3 also provides further functionality for manipulating the DOM. Developers can define a scale based on the domain and range of values that are defined in the data provided by the user. The placement of elements within the document can then be placed according to the desired scale. D3 provides support for many different types of scales including linear scales, power scales, logarithmic scales, and temporal scales. Axes can also be created to correspond to the defined scale.

The user has the ability to change the DOM as needed. However, D3 also supports a large number of visualization layouts so that the user does not have to define the positions for the elements in a given visualization. The two layouts that are of relevance for this work are the tree layout and the spring layout.

The tree layout uses the Rheingold-Tilford algorithm for drawing tidy trees [RJT81]. The force layout uses an algorithm created by Dwyer [Dwy09] to create a scalable and con-

[1] http://jquery.com

strained graph layout. The physical simulations are based on the work by Jakobsen [Jak03]. The implementation "uses a quadtree to accelerate charge interaction using the Barnes–Hut approximation. In addition to the repulsive charge force, a pseudo-gravity force keeps nodes centered in the visible area and avoids expulsion of disconnected subgraphs, while links are fixed-distance geometric constraints. Additional custom forces and constraints may be applied on the "tick" event, simply by updating the x and y attributes of nodes" [Bos12].

To help the viewer interact with the visualization, D3 provides support for the zoom and drag behaviors. This listens to the mouse clicks commonly associated with zooming (i.e. scrolling, double clicking) and enlarges the image as would be expected. With this same mechanism, the developer can enable the user to grab hold of the canvas and pan through the image to inspect it closer.

Despite the considerable functions that D3 offers, it is very easy for the user to begin developing with D3. The API is described in detail on the D3 Wiki [Bos12], and the D3 website [2] includes an extensive array of examples that new developers can use as a jumping off point. The D3 developer community is very large, so it is easy to find answers to almost every question online.

Listing 1: Dynamically create a dropdown menu.

```
// Select element with id "body" and append a select tag onto it. When it
    changes, the defined function will be triggered.
var dropdown = d3.select("#body")
            .append("select")
            .on("change", function() {
              var choice = this.options[this.selectedIndex].__data__;
              // handle choice
            });

var options = [{id: 1, name: "Option 1"},
         {id: 2, name: "Option 2"},
         {id: 3, name: "Option 3"}];

// Create an option tag with id and text elements for each option that is
    defined in options
dropdown.selectAll("option")
    .data(options)
    .enter()
    .append("option")
    .attr("id", function(d) { return "op" + d.id; })
    .text( function(d) { return d.name; });

// Select option 3 by default
dropdown.select("#op3")
    .attr("selected", true);
```

Listing 2: Html generated from Listing 1

```
<div id=body>
    <select>
```

[2] http://d3js.org

```
        <option id="op1">Option 1</option>
        <option id="op2">Option 2</option>
        <option id="op3" selected=true>Option 3</option>
    </select>
</div>
```

2.4 GraphViz integration with emscripten

The current visualizations available in the ProB Tcl/Tk application are powered using the GraphViz [3] graph visualization software. In the ProB CLI, there is support for generating graphs for the state space in the DOT graph language. The problem with this, and the reason that we are researching other alternatives, is that drawing GraphViz graphs is extremely inefficient. However, for the state spaces that are derived using the signature merge algorithm, or for the transition diagrams that can be created from a state space, these graphs are quite pleasing to the eye. The derived graphs are also usually small enough that they can be drawn efficiently.

For this reason, we wanted to some how be able to visualize small graphs written in the DOT language. In order to do this, we took advantage of the emscripten compiler [Zak11]. This is a compiler that compiles LLVM bitcode to JavaScript so that it can be run in any browser. Any C programs can be compiled to LLVM. We used the Viz.js JavaScript library [4] developed by Mike Daines which has compiled GraphViz from C to JavaScript and provided a wrapper function to produce svg visualizations (see Listing 3).

Listing 3: Create a visualization with viz.js and insert it into an html page.

```
svg = Viz("digraph { a -> b; }", "svg");
$("#elementId").replaceWith(svg)
```

3 Contribution

3.1 Visualization framework

3.1.1 Structure

One of the main issues that we had to be deal with at the beginning of the development process was the issue of how to integrate the visualizations into the ProB 2.0 API. A functioning jetty server was already available, so we had to integrate our JavaScript visualizations into the existing framework. We ran into problems at the beginning, because a Java servlet is a singleton object. We needed to create a way to let the servlet know for which visualization it should be calculating. In order to do this, we created a session based servlet. When the user wants to open a visualization, the servlet is contacted. The servlet then creates a servlet responsible for the visualization and a unique

[3] http://www.graphviz.org
[4] https://github.com/mdaines/viz.js

session id. When a visualization sends a `GET` request, it includes its session id. The session based servlet then forwards the request to the the servlet that is responsible for the data calculations.

The visualization communicates with the servlet by setting up a polling interval. It tells the state space if it needs the complete data set or just the changes since the last polling interval. The servlet then responds by sending the correct data (see Figure 1).

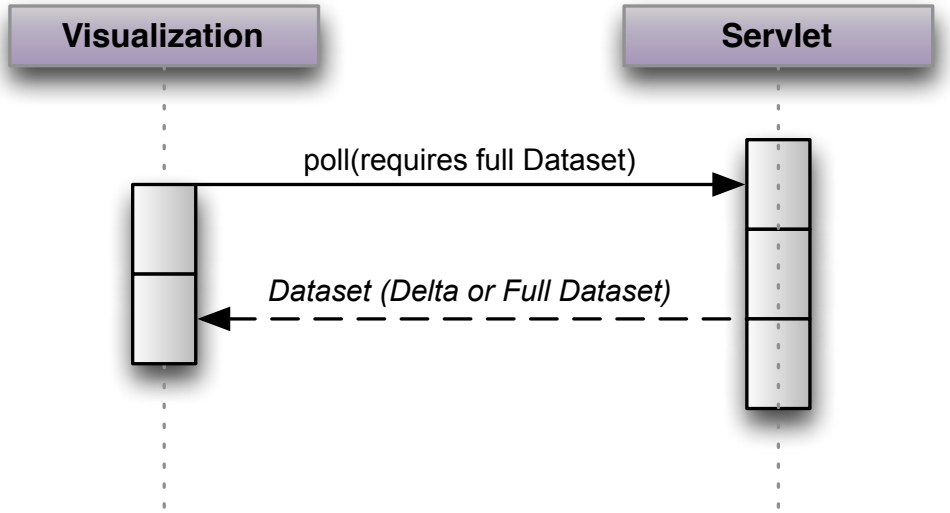


Figure 1: How visualizations communicate with the servlet that is responsible.

The servlet is also connected to the ProB 2.0 API through the listener framework. Depending on the data that is needed for a particular visualization, the servlet registers either to receive notifications about changes in the current animation or about changes in the state space (see Figure 2). Every time the servlet receives a new notification, the data needed for the visualization is recalculated.

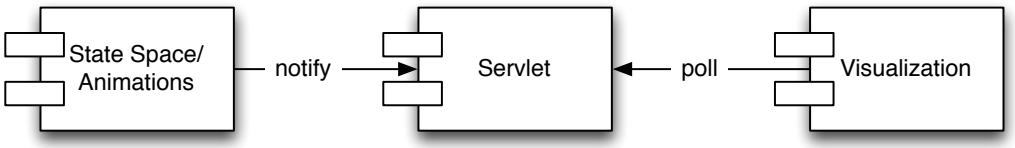


Figure 2: Model of the program flow.

3.1.2 User interaction with the visualizations

Once we implemented a way to integrate the visualization servlets into the ProB 2.0 application, it was still necessary to implement an easy way for the user to interact with the visualizations. One of the main advantages of the D3 visualization framework is the flexibility for the user. Using the D3 selectors, it is possible for the user to select and change

the attributes of any of the elements of the visualization. In order to offer this functionality to the users from within the ProB 2.0 application, we decided that we needed to lift the functionality from the JavaScript level into the existing groovy console in the Java 2.0 API. This was accomplished by creating a `Transformer` object that represents the action that the user wants to carry out in the visualization. Then the `Transformer` object is added to the particular visualization and is applied the next time the visualization is redrawn. The `Transformer` object was written so that its functionality is similar to what the user would actually write using the D3 library.

We wanted how the user interacted with the visualization to be as natural as possible. The user should not have a hard time learning how to manipulate the visualization. For this reason, we decided to create a small DSL that would enable the user to specify which attributes should change within a particular visualization. It is possible for the user to create `Transformers` in the console (see Listing 4). To do this, the user can specify a selection based on the W3C Selection API. It also specifies the attributes that should be changed for the specified selectors. The user can then apply this `Tranformer` to the servlet that is responsible for generating the data for a given transformation. To this purpose, when a new servlet is created, a variable is created in the groovy console that corresponds to the servlet in question so that the user can apply `Transformers` to the visualization directly.

Listing 4: Define rules for the transformation of visualization elements

```
// Select elements with ids "rroot" and "r1" and set their fill and stroke
    attributes
x = transform("#rroot,#r1") {
        set "fill", "red"
        set "stroke", "gray"
}

// Apply to visualization
viz0.apply(x)
```

It is also possible to harness the power of the Groovy closure in order to create a Transformer that can be parameterized (see Listing 5).

Listing 5: Use Groovy closures to generate Transformers

```
// Create a closure that can be parameterized
colorize = { selection, color ->
                transform(selection) {
                    set "fill",color
                }
           }

// Color elements "rroot" and "r1" green
viz0.apply(colorize("#rroot,#r1","green"))
```

These `Transformers` are sent to the visualization every time that it is polled. The visualizations can then apply them. It is therefore possible for the user to manipulate the visualization. Unfortunately, in order to define these `Transformers`, the user needs to have some idea of how the elements are named internally. For the state space visualiza-

tion, we also introduced an abstaction that filters a state space by predicate and produces a `Transformer` based on the states that are calculated (see Listing 6). When new states are added to the state space, these will also be filtered based on the given predicate and the `Transformer` will be updated accordingly.

Listing 6: Create a `Transformer` based on the states that match a given predicate

```
// Define your formula
predicate = "active\\/waiting={}" as ClassicalB

// Create a Transformer that will filter the given state space (saved in
    space0) according to the given predicate
x = transform(predicate, space0 ) {
    set "fill", "blue"
    set "stroke", "white"
}

// Apply the Transformer to the visualization
viz0.apply(x)
```

3.1.3 Extensibility

In addition to creating visualizations that are useful to the user, we also wanted to make it easy for other developers to create similar visualizations. In order to help with this, we encapsuled certain elements common to all of the visualizations into a separate script. In order to have access to these elements, a developer simply needs to include the script before that of his visualization. Currently, there are two main elements included in this script. Firstly, the user can use the `createCanvas` function to create a D3 selection that includes support for zooming (see Listing 7). Secondly, if the user wants to be able to apply the `Transformers` that are described in the last section, there is a built in function to apply a list of `Transformers` to the visualization (see Listing 8). In the future, it will also be possible to add more functions to this script to make it even easier to create visualizations for ProB.

Listing 7: Append a D3 selection to an element that includes support for zooming

```
var width = 600, height = 400;
var svg = createCanvas("#elementId", width, height);

// Append all further elements to this selection
svg.append(...);
```

Listing 8: Apply list of `Transformers` received from servlet

```
// The servlet has been polled, and a response has been received
var styling = response.styling

// ... Render the visualization
// ... Then apply the user defined styling
applyStyling(styling);
```

3.2 Visualization of the State Space

3.2.1 Main visualization

When a state space visualization is opened, the visualization servlet responsible for the state space visualization takes the state space associated with the current animation and extracts the information about the nodes and edges contained within the graph. This information is then processed by D3 using the force layout and rendered to create a visualization (see Figure 3). As a basis for the visualization, we used the Force Directed Graph example created by Michael Bostock (see Appendix A).

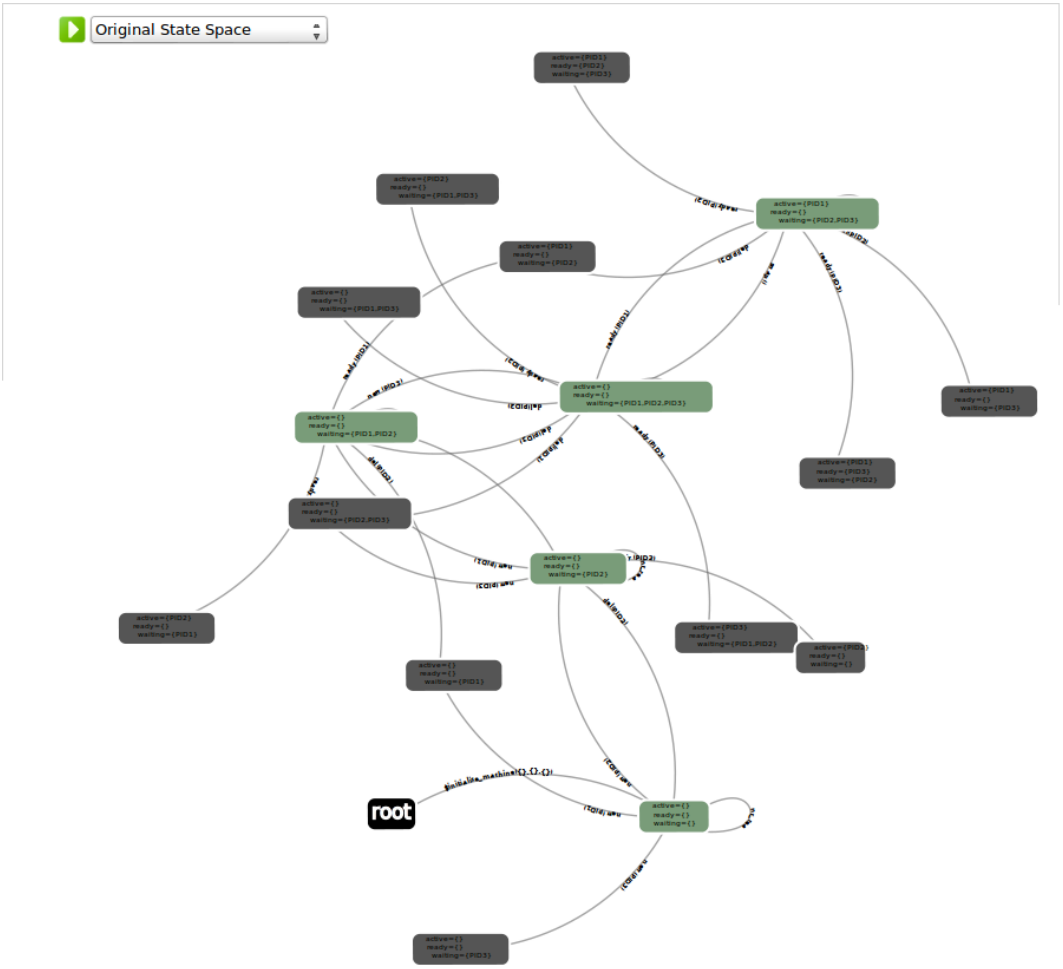


Figure 3: Visualization of the state space for the Scheduler example

Unfortunately, the state space contains an extremely large amount of information that has to be processed. This includes the values of the variables and the invariant for every state in the graph and the names and parameters of the operations that correspond to every edge in the graph. Because of this, it is rather difficult to create a useful visualization of the whole state space because the user not only wants to inspect how the state space

appears as a whole but also the individual states within the operation. As a proposed solution of this problem, we used the zoom functionality that is available in D3. The main problem was that if the visualization of the nodes was large enough for the user to read the values of the variable at the given state, it would no longer be possible to see the state space as a whole. Instead of trying to meet both requirements at once, we simply made the text that is printed on the node and edge objects very small. When the visualization is created, the user can inspect the graph as a whole how the graph appears as a whole. The text for the given nodes, however, is virtually indiscernable. If the user wants to inspect a particular node, they can do so by zooming into the visualization. The text is then larger, and the user can see the nodes that are in the direct neighborhood of the node in question (see Figure 4). Then user can also click on the background of the visualization in order to pan through the visualization and inspect other nodes and edges. The visualization is also interactive. The user can grab a node and move it around to a desired position.

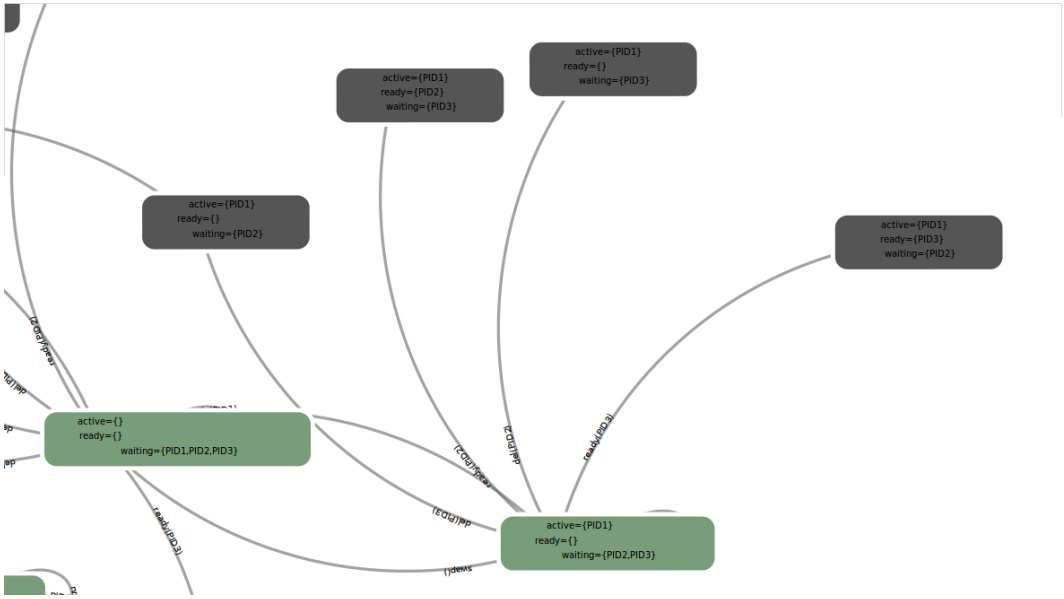


Figure 4: By zooming in, the user can inspect the neighborhood of a given state.

We had some performance issues that were associated with very large state spaces. The force layout keeps adjusting the graph until it reaches a fix point. The problem was that as the state space grew, there were more and more objects that had to be accounted for. The force layout just kept calculating and moving the the nodes. This didn't only affect the appearance of the visualization. It cost enough resources that the whole eclipse plugin would become unresponsive. A quick fix to this problem was adding a play/pause button to the upper left hand corner of the visualization (see Figure 9). When the user is satisfied with the visualization and how it is laid out, he can press the pause button, and the graph will stop being rendered. When he presses the play button, the rendering will begin again. The iterative force layout keeps running in the background, so when the rendering begins again, the visualization has had time to stabilize.

When new states are added to the state space, these states are added to the graph when the visualization receives them in the next poll. The graph is then updated. This results

in a nice animation.

Status about the invariant is a available in the graph based on the color of the nodes. If an invariant violation is present, the node is colored red. If the invariant is ok, the node is colored green. Otherwise, if the invariant has not yet been calculated for the given node, the node is colored gray.

3.2.2 Visualization of Derived Graphs

There is also support for the visualization of graphs that are derived from the state space. The ProB CLI support the generation of smaller graphs that are derived from the state space in question. These graphs show information about state space that is not obvious when dealing with the state space as whole. Two algorithms for derived graph generation are currently supported. The signature merge algorithm is supported (see Figure 5). The user can also create a transition diagram based on a user defined expression (see Figure 6).

If the user wants to use the GraphViz algorithms and rendering engine, there is also support for both the signature merge (see Figure 7) and transition diagram (see Figure 8).

3.2.3 User Interface

There is also a basic user interface for the user to interact with the state space visualization. There is a drop down menu from which the user can decide which visualization he wants to view (see Figure 9).

The signature merge algorithm is calculated based on the events in the model that is being animated. By default, all events are selected, but it is also possible for a user to disable events when calculating the signature merge algorithm. In the state space visualization, if the user is dealing with the signature merge graphs, a settings icon appears and the user can click on it and select or deselect the events that are of interest to him (see Figure 10).

When the user chooses to create a transition diagram from the menu, a prompt appears and the user can input the desired transition. When the transition diagram is calculated, a text field appears. In order to change the expression for a given transition diagram, the user can type it in the text field and submit it (see Figure 11). The algorithm will then be recalculated and the graph will be rerendered.

3.3 Visualization of a B-type Formula

The ProB CLI already supported the functionality of expanding a formula into its subformulas and finding its value at a given state. However, the expanding of the formula took place lazily. A formula would be sent to the ProB CLI and then the direct subformulas of this formula would be sent back. The software would then have to contact ProB CLI recursively until all of the subformulas had been calculated and cached on the Java side. For the predicate visualization, we wanted the formula to be completely expanded

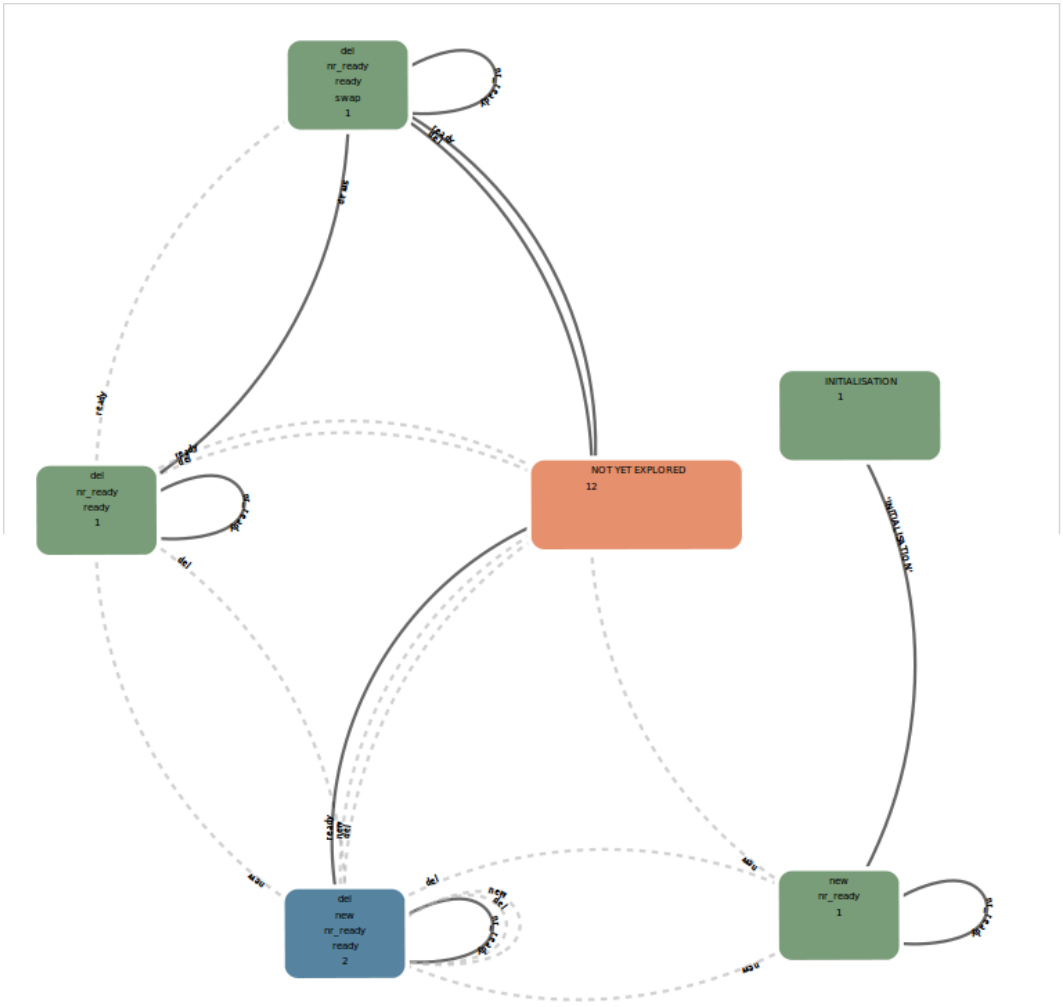


Figure 5: D3 Visualization of signature merge for Scheduler example

and then sent in its entirety to the ProB 2.0 API. In order to do this, we implemented a prolog predicate within the ProB CLI that performs the recursive expanding of a B formula before it is sent back to the Java API. The predicate also delivers the value of each subformula for the given state. This ensures that performance will not an issue.

The final visualization is interactive (see Figure 12). If a formula has subformulas, the user can select it from within the visualization to expand or to retract the subformulas. The subformulas are always either predicates or expressions. If they are expressions, they are colored white or light grey depending on whether they have subformulas or not. If the formula is a predicate that has evaluated to true for the given formula, the node is colored green. If the formula is a predicated that has evaluated to false, node is colored in red. The value of the given formula is also printed beneath the formula. This allows the user to visually identify the parts of the formulas and their given values.

In the implementation of the formula, the D3 tree layout is used. The expanding and

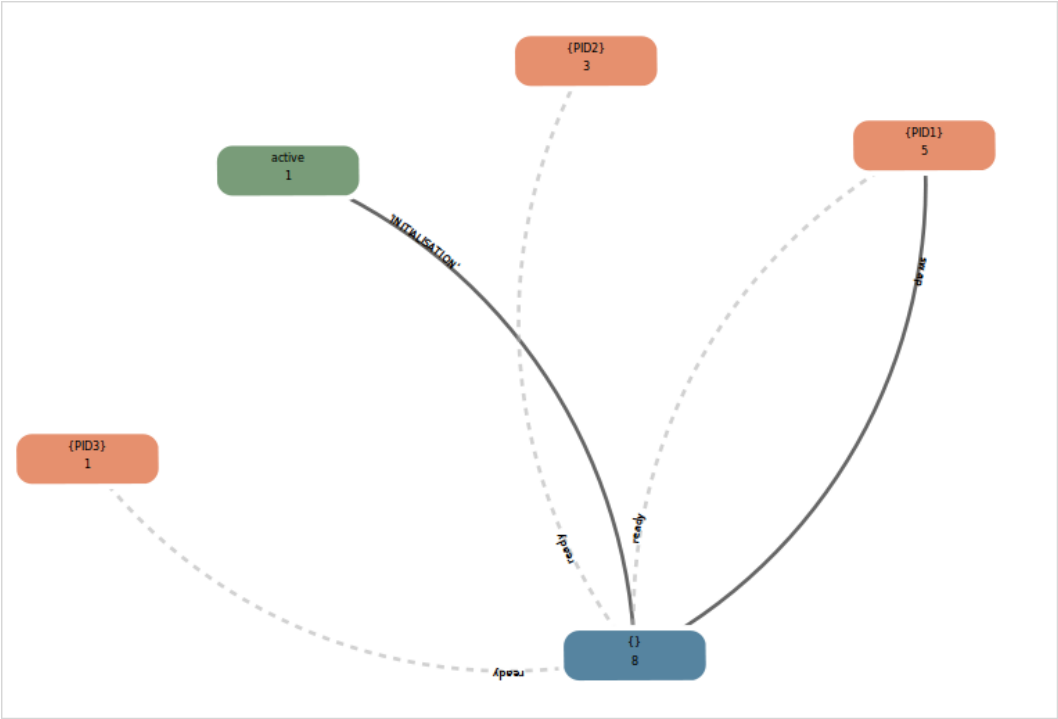


Figure 6: D3 Visualization of the transition diagram of `active` in Scheduler example

collapsing of the nodes takes place with a simple JavaScript function. This visualization is based on the Collapsible Tree Layout from the D3 website (see Appendix B). By harnessing the power of the D3 zoom behavior, it is also possible to zoom in and out of the visualization and to pan the image to inspect it closer. The servlet responsible for the visualization implements a listener to identify if any changes in the animation occur. If they do, the formula is recalculated for the new current state, and the visualization is redrawn.

3.4 Visualization of the Value of a Formula Over Time

The ProB 2.0 API supports the evaluation of a given formula over the course of the history of an animation. Because a state is defined by the values that the variables take on when dealing with B type specification languages, it can be 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.

In order for the ProB 2.0 API to evaluate a formula, it extracts the list of states that the animation visits over the course of its history. Then it contacts the ProB CLI and extracts the value that the formlua takes on for each state in the list. This information is then processed by D3 to produce a simple line plot (see Figure 13). As of now, formulas can only be visualized if they take on integer values. In the future, we plan to support the visualization of formulas that take on boolean values. The visualization also interacts with the ProB 2.0 API. If the current state changes, the formula is recalculated and a new

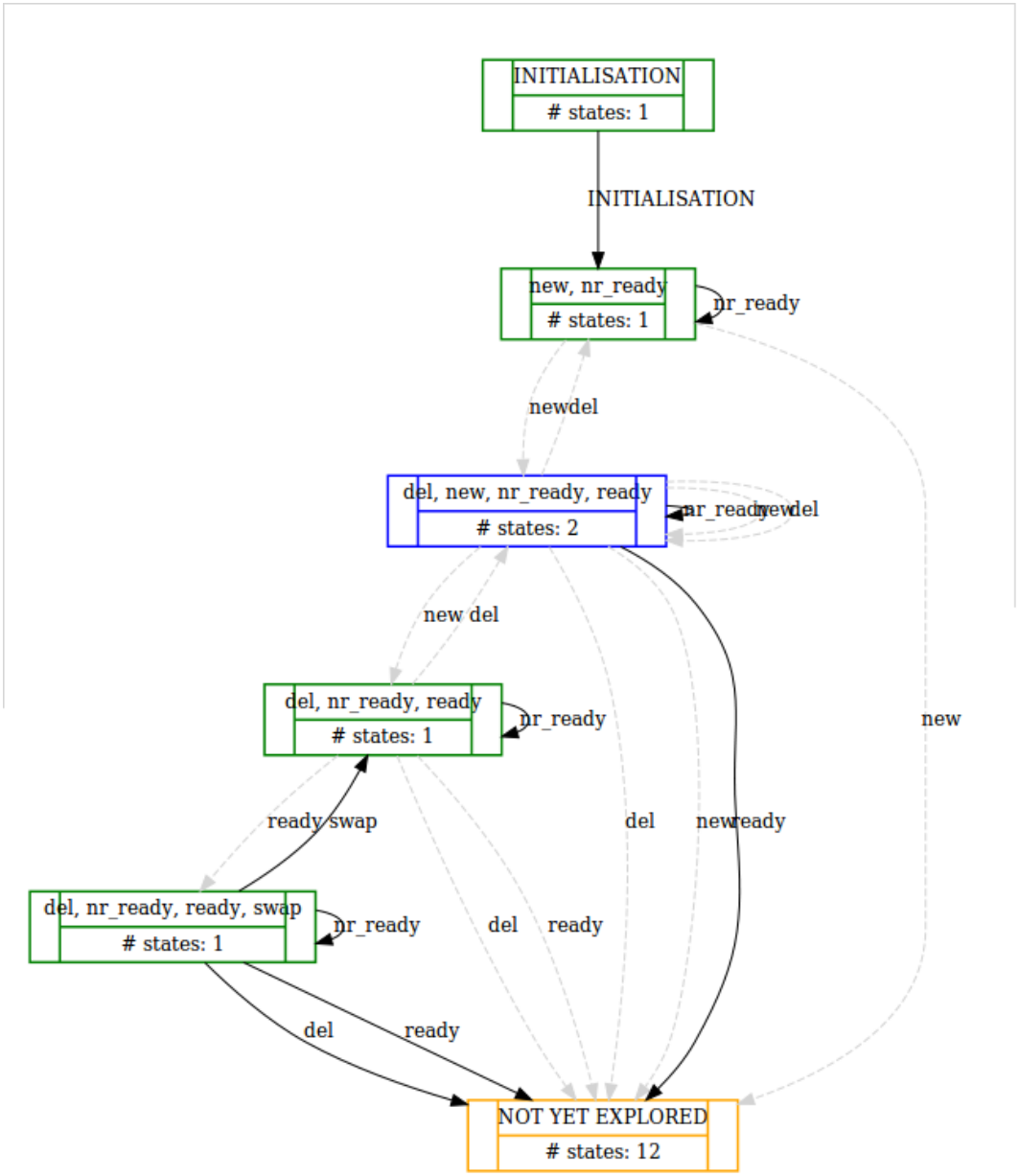


Figure 7: GraphViz powered Visualization of signature merge for the Scheduler example

plot is produced.

4 Conclusion

Over the course of this work, we have shown the feasibility of using D3 to create data visualizations within ProB. It is not only possible to create the desired visualizations, it is also easy to adapt the servlets so that the visualizations are updated to reflect changes

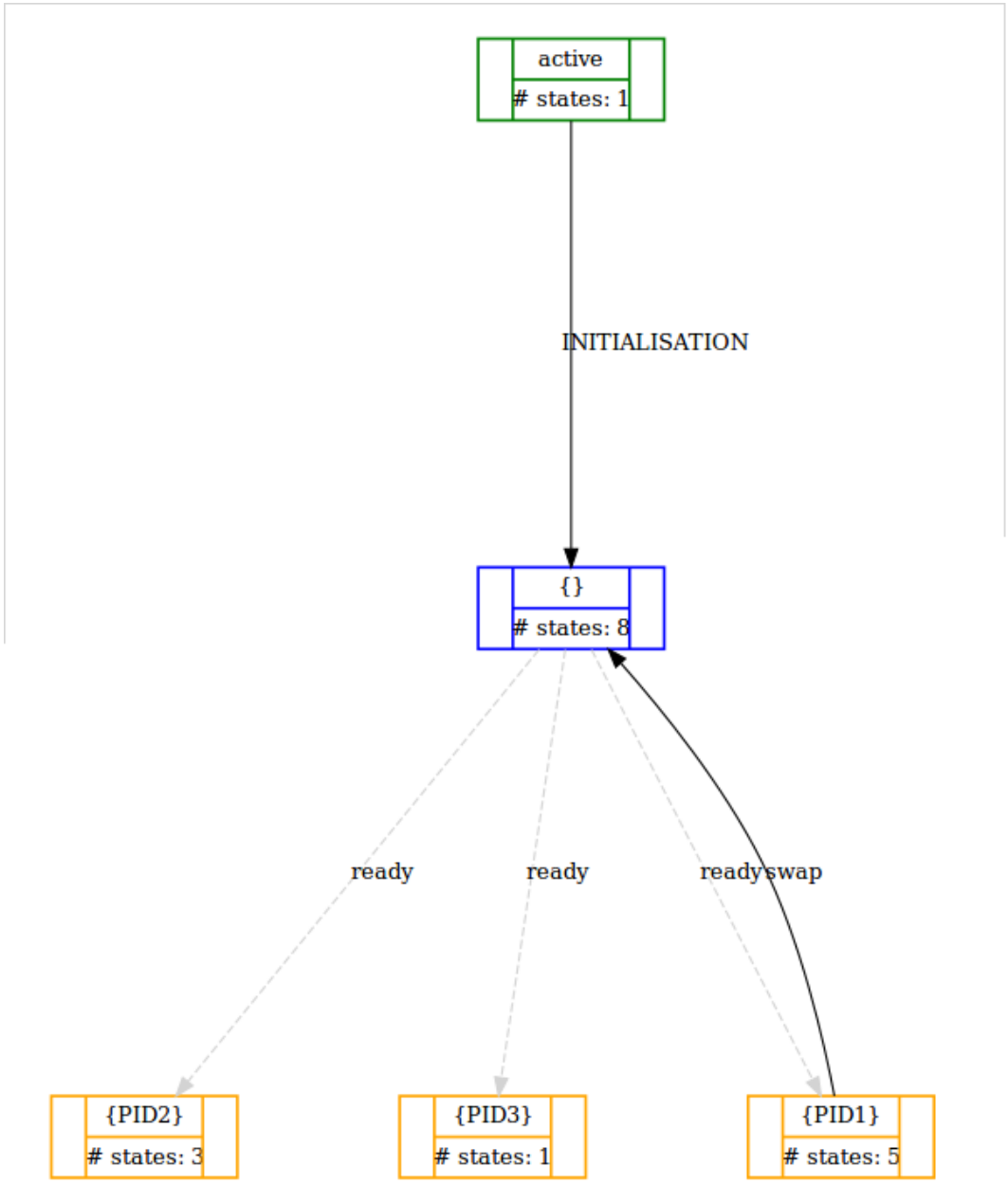


Figure 8: GraphViz powered Visualization of the transition diagram of `active` in Scheduler model

made in the course of model animation. The styling for these visualizations can be easily defined by the user. This gives the user a large amount of control over the visualization.

The visualization of the state space was the focus of this work. The visualization that was created is interactive and is automatically updated as soon as states are calculated and cached in the state space. It also can handle relatively large state spaces (TEST THIS OUT TO DETERMINE HOW LARGE!!!!).

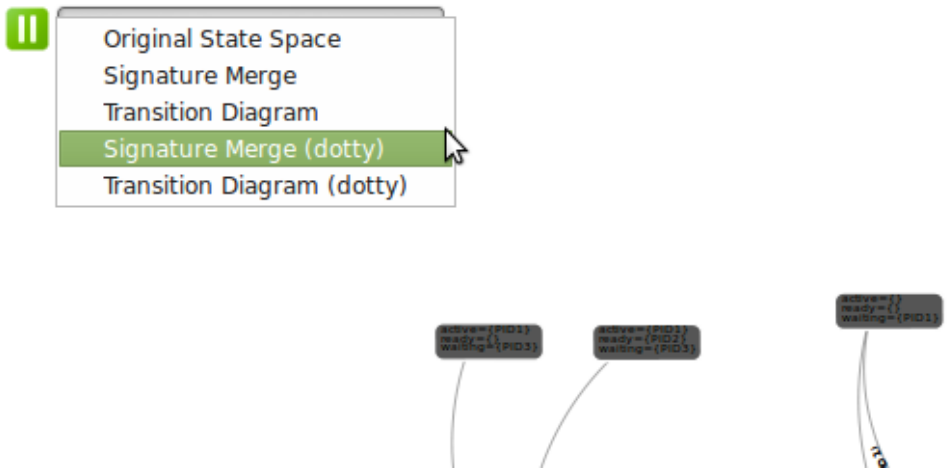


Figure 9: The user can select the desired visualization and play and pause the rendering.

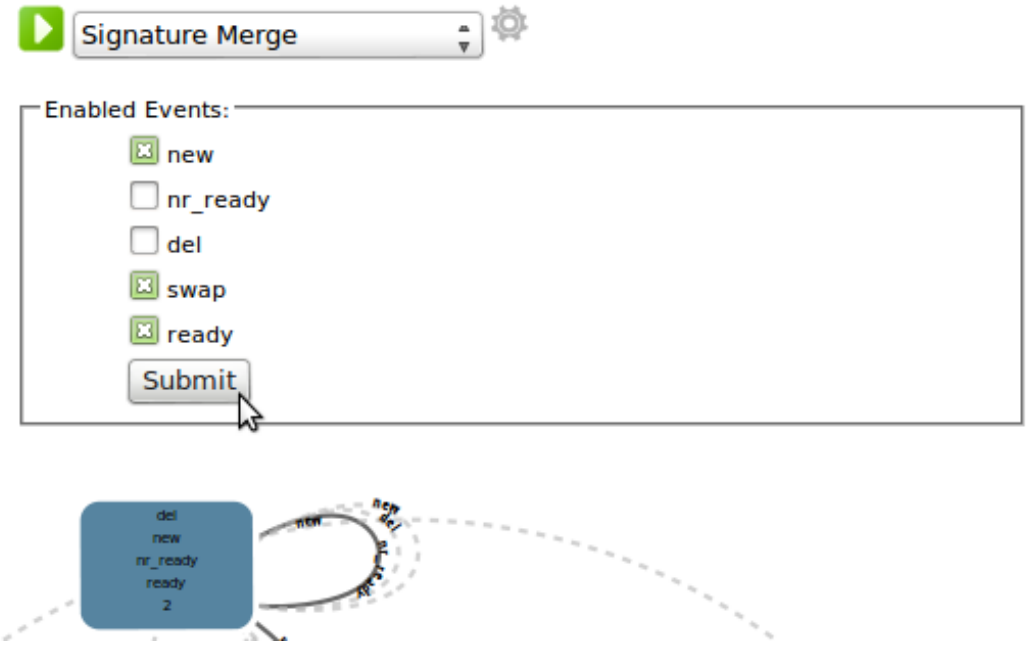


Figure 10: User interface to chose events for signature merge.

The decision to support GraphViz visualizations in the state space visualization in addition to D3 visualizations took place relatively late in the development process. Using the existing visualization framework, it was possible to implement this feature in a relatively short period of time. This shows that the visualization framework is relatively flexible and can be easily extended.

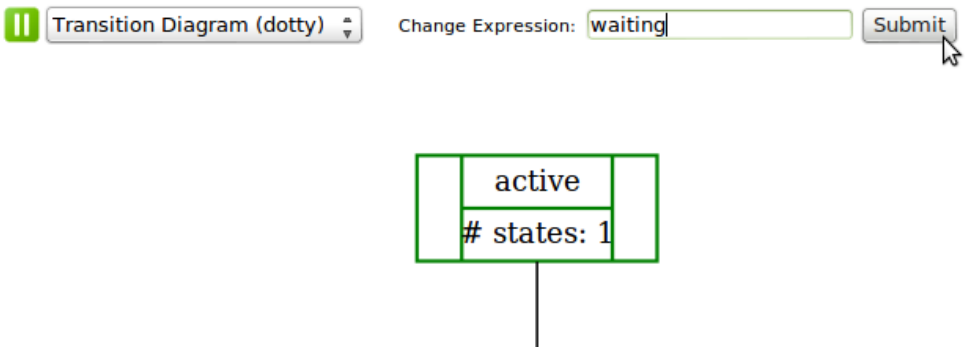


Figure 11: The user can input a new expression to recalculate the transition diagram.

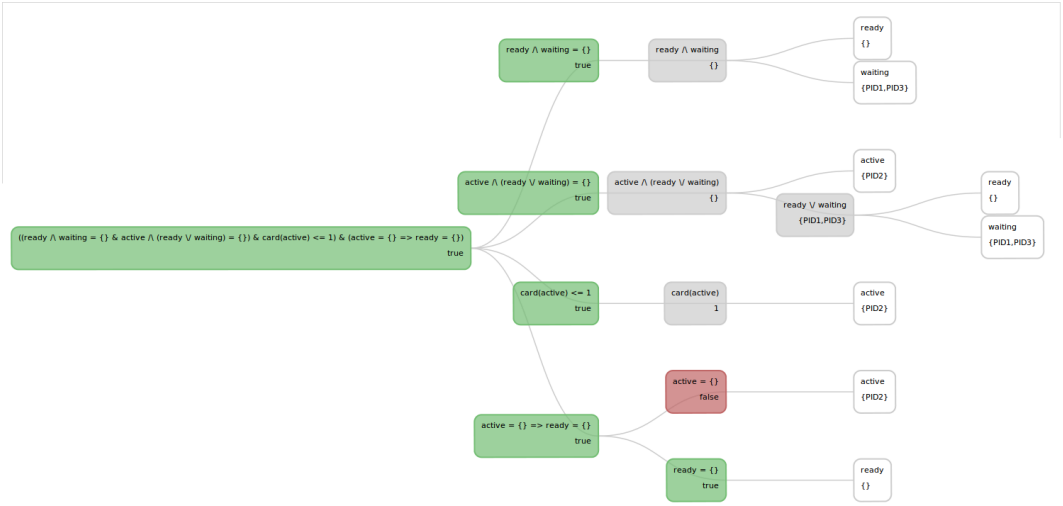


Figure 12: Visualization of the invariant of the Scheduler model

5 Future Work

One of the main features we implemented in this work was the creation of a visualization framework. Each visualization has its own unique elements, but the shared elements have been encapsulated so that they can be reused in other visualizations. This is true both on the server and the client side. The basic elements of a ProB visualization have been extracted into a JavaScript library. For instance, developers can now use this library to create a basic canvas which can be zoomed and panned. We have also implemented basic servlets that listen for animation and state space changes. These can easily be extended to create new servlets.

We are currently developing a worksheet element for the ProB 2.0 Plugin. This will serve as documentation and a means to run groovy scripts within the ProB application. In the future, the visualizations created in the scope of this work will be embedded within this worksheet feature.

It will also be necessary to implement other data visualizations in ProB 2.0. For instance, there is currently no graphical represenation of a state. This visualization is available in

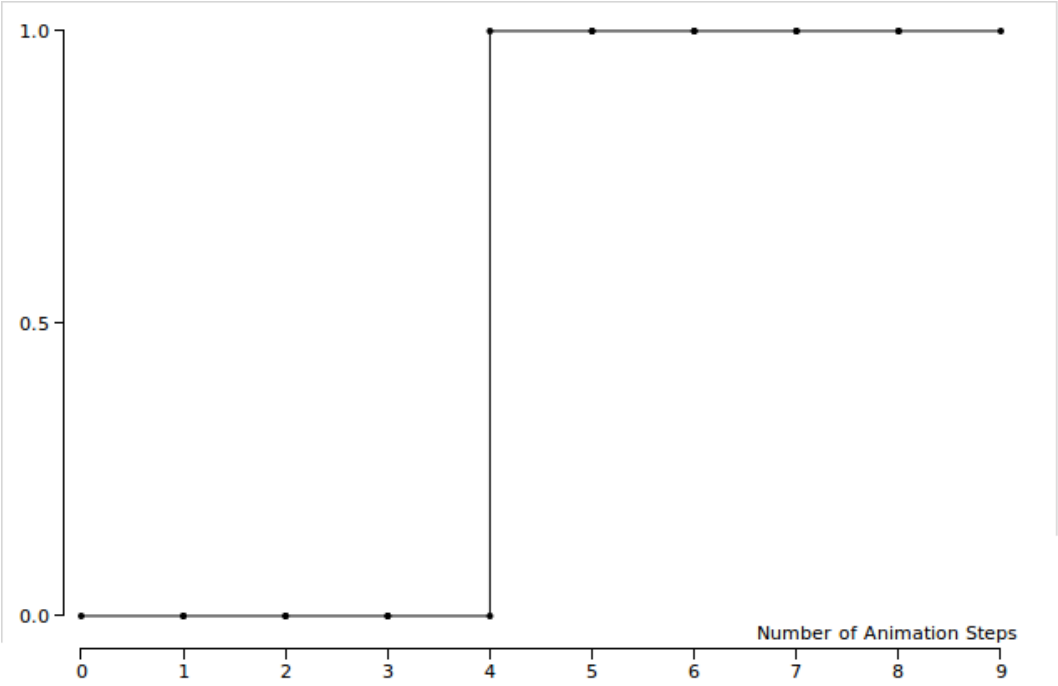


Figure 13: Visualization of value of *card(active)* from the Scheduler model over the course of an animation.

the ProB Tcl/Tk application, but it still needs to be implemented within ProB 2.0.

The state space visualization and the visualization of a formula over time are not language specific. It is possible to use the visualizations for any specification language that ProB supports. For the visualization of the breakdown of a formula, however, this is not the case. The current implementation supports only Classical B and Event-B formulas. In the future, work could be done to support formulas from other formalisms.

Current visualizations will also need to be maintained and updated to add functionality. For instance, it might be possible to integrate the proposed state visualization with the existing state space visualization. Then, when a user would select a state, a window would pop up displaying the state visualization for that particular state.

References

[AAH05] ABRIAL, J.R. ; ABRIAL, J.R. ; HOARE, A.: *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005. – ISBN 9780521021753

[BH07] BUTLER, Michael ; HALLERSTEDE, Stefan: The Rodin Formal Modelling Tool. In: *BCS-FACS Christmas 2007 Meeting*, 2007

[BOH11] BOSTOCK, Michael ; OGIEVETSKY, Vadim ; HEER, Jeffrey: D3: Data-Driven Documents. In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011). `http://vis.stanford.edu/papers/d3`

[Bos12] BOSTOCK, Michael: *D3 Wiki*. `https://github.com/mbostock/d3/wiki`. Version: 2012

[Dwy09] DWYER, Tim: Scalable, versatile and simple constrained graph layout. In: *Proceedings of the 11th Eurographics / IEEE - VGTC conference on Visualization*. Eurographics Association (EuroVis'09), 991–1006

[Jak03] JAKOBSEN, Thomas: *Advanced Character Physics*. 2003

[LB03] LEUSCHEL, Michael ; BUTLER, Michael: ProB: A Model Checker for B. In: KEIJIRO, Araki (Hrsg.) ; GNESI, Stefania (Hrsg.) ; DINO, Mandrio (Hrsg.): *FME* Bd. 2805, Springer-Verlag, 2003. – ISBN 3–540–40828–2, S. 855–874

[LB05] LEUSCHEL, Michael ; BUTLER, Michael: Automatic Refinement Checking for B. In: LAU, Kung-Kiu (Hrsg.) ; BANACH, Richard (Hrsg.): *Proceedings ICFEM* Bd. 3785, Springer-Verlag, May 2005, S. 345–359

[LB08] LEUSCHEL, Michael ; BUTLER, Michael: ProB: An Automated Analysis Toolset for the B Method. In: *Software Tools for Technology Transfer (STTT)* 10 (2008), Nr. 2, S. 185–203

[LSBL08] LEUSCHEL, Michael ; SAMIA, Mireille ; BENDISPOSTO, Jens ; LUO, Li: Easy Graphical Animation and Formula Viewing for Teaching B. In: ATTIOGBÉ, C. (Hrsg.) ; HABRIAS, H. (Hrsg.): *The B Method: from Research to Teaching*, Lina, 2008, S. 17–32

[LT05] LEUSCHEL, Michael ; TURNER, Edd: Visualising Larger State Spaces in ProB. In: TREHARNE, Helen (Hrsg.) ; KING, Steve (Hrsg.) ; HENSON, Martin (Hrsg.) ; SCHNEIDER, Steve (Hrsg.): *ZB* Bd. 3455, Springer-Verlag, November 2005. – ISBN 3–540–25559–1, S. 6–23

[RJT81] REINGOLD, Edward M. ; JOHN ; TILFORD, S.: Tidier drawing of trees. In: *IEEE Trans. Software Eng* (1981)

[Sch01] SCHNEIDER, S.: *The B-Method: An Introduction*. Palgrave Macmillan Limited, 2001 (Cornerstones of Computing Series). – ISBN 9780333792841

[Zak11] ZAKAI, Alon: Emscripten: an LLVM-to-JavaScript compiler. In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM (SPLASH '11). – ISBN 978–1–4503–0942–4, 301–312

List of Figures

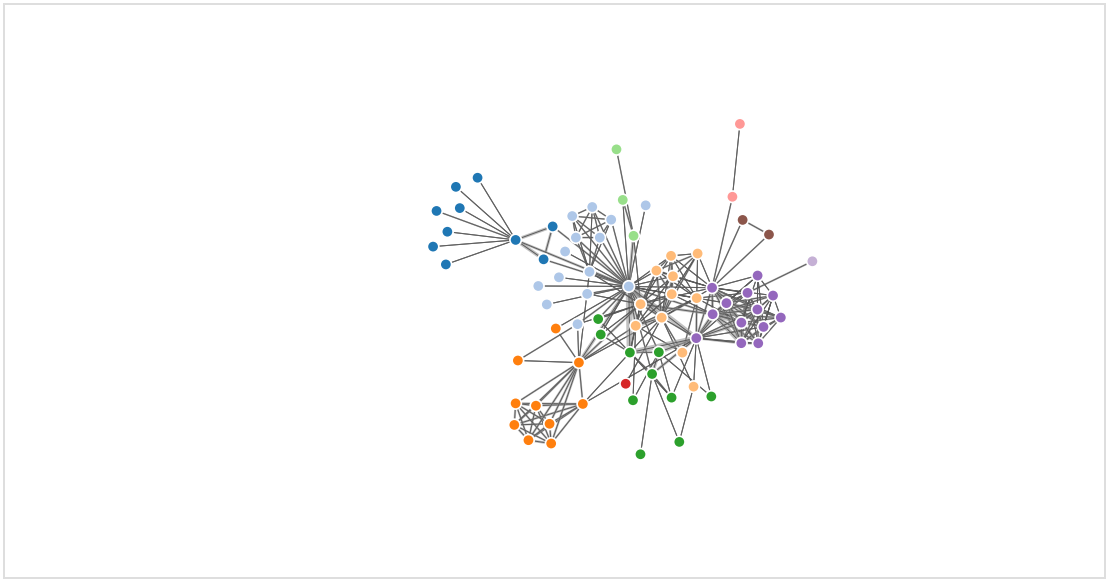
Listings

A Force Directed Graph

mbostock's block #4062045

Force-Directed Graph

May 8, 2013



This simple force-directed graph shows character co-occurence in *Les Misérables*. A physical simulation of charged particles and springs places related characters in closer proximity, while unrelated characters are farther apart. Layout algorithm inspired by Tim Dwyer and Thomas Jakobsen. Data based on character coappearence in Victor Hugo's *Les Misérables*, compiled by Donald Knuth.

Open in a new window.

index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

.node {
  stroke: #fff;
  stroke-width: 1.5px;
}

.link {
  stroke: #999;
  stroke-opacity: .6;
}

</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script>

var width = 960,
    height = 500;
```

```
var color = d3.scale.category20();

var force = d3.layout.force()
    .charge(-120)
    .linkDistance(30)
    .size([width, height]);

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

d3.json("miserables.json", function(error, graph) {
  force
      .nodes(graph.nodes)
      .links(graph.links)
      .start();

  var link = svg.selectAll(".link")
      .data(graph.links)
    .enter().append("line")
      .attr("class", "link")
      .style("stroke-width", function(d) { return Math.sqrt(d.value); });

  var node = svg.selectAll(".node")
      .data(graph.nodes)
    .enter().append("circle")
      .attr("class", "node")
      .attr("r", 5)
      .style("fill", function(d) { return color(d.group); })
      .call(force.drag);

  node.append("title")
      .text(function(d) { return d.name; });

  force.on("tick", function() {
    link.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    node.attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; });
  });
});

</script>
```

miserables.json

```
{
  "nodes":[
    {"name":"Myriel","group":1},
    {"name":"Napoleon","group":1},
    {"name":"Mlle.Baptistine","group":1},
    {"name":"Mme.Magloire","group":1},
    {"name":"CountessdeLo","group":1},
    {"name":"Geborand","group":1},
    {"name":"Champtercier","group":1},
    {"name":"Cravatte","group":1},
    {"name":"Count","group":1},
    {"name":"OldMan","group":1},
    {"name":"Labarre","group":2},
    {"name":"Valjean","group":2},
    {"name":"Marguerite","group":3},
    {"name":"Mme.deR","group":2},
```

```
    {"name":"Isabeau","group":2},
    {"name":"Gervais","group":2},
    {"name":"Tholomyes","group":3},
    {"name":"Listolier","group":3},
    {"name":"Fameuil","group":3},
    {"name":"Blacheville","group":3},
    {"name":"Favourite","group":3},
    {"name":"Dahlia","group":3},
    {"name":"Zephine","group":3},
    {"name":"Fantine","group":3},
    {"name":"Mme.Thenardier","group":4},
    {"name":"Thenardier","group":4},
    {"name":"Cosette","group":5},
    {"name":"Javert","group":4},
    {"name":"Fauchelevent","group":0},
    {"name":"Bamatabois","group":2},
    {"name":"Perpetue","group":3},
    {"name":"Simplice","group":2},
    {"name":"Scaufflaire","group":2},
    {"name":"Woman1","group":2},
    {"name":"Judge","group":2},
    {"name":"Champmathieu","group":2},
    {"name":"Brevet","group":2},
    {"name":"Chenildieu","group":2},
    {"name":"Cochepaille","group":2},
    {"name":"Pontmercy","group":4},
    {"name":"Boulatruelle","group":6},
    {"name":"Eponine","group":4},
    {"name":"Anzelma","group":4},
    {"name":"Woman2","group":5},
    {"name":"MotherInnocent","group":0},
    {"name":"Gribier","group":0},
    {"name":"Jondrette","group":7},
    {"name":"Mme.Burgon","group":7},
    {"name":"Gavroche","group":8},
    {"name":"Gillenormand","group":5},
    {"name":"Magnon","group":5},
    {"name":"Mlle.Gillenormand","group":5},
    {"name":"Mme.Pontmercy","group":5},
    {"name":"Mlle.Vaubois","group":5},
    {"name":"Lt.Gillenormand","group":5},
    {"name":"Marius","group":8},
    {"name":"BaronessT","group":5},
    {"name":"Mabeuf","group":8},
    {"name":"Enjolras","group":8},
    {"name":"Combeferre","group":8},
    {"name":"Prouvaire","group":8},
    {"name":"Feuilly","group":8},
    {"name":"Courfeyrac","group":8},
    {"name":"Bahorel","group":8},
    {"name":"Bossuet","group":8},
    {"name":"Joly","group":8},
    {"name":"Grantaire","group":8},
    {"name":"MotherPlutarch","group":9},
    {"name":"Gueulemer","group":4},
    {"name":"Babet","group":4},
    {"name":"Claquesous","group":4},
    {"name":"Montparnasse","group":4},
    {"name":"Toussaint","group":5},
    {"name":"Child1","group":10},
    {"name":"Child2","group":10},
    {"name":"Brujon","group":4},
    {"name":"Mme.Hucheloup","group":8}
  ],
  "links":[
    {"source":1,"target":0,"value":1},
    {"source":2,"target":0,"value":8},
    {"source":3,"target":0,"value":10},
```

```
    {"source":3,"target":2,"value":6},
    {"source":4,"target":0,"value":1},
    {"source":5,"target":0,"value":1},
    {"source":6,"target":0,"value":1},
    {"source":7,"target":0,"value":1},
    {"source":8,"target":0,"value":2},
    {"source":9,"target":0,"value":1},
    {"source":11,"target":10,"value":1},
    {"source":11,"target":3,"value":3},
    {"source":11,"target":2,"value":3},
    {"source":11,"target":0,"value":5},
    {"source":12,"target":11,"value":1},
    {"source":13,"target":11,"value":1},
    {"source":14,"target":11,"value":1},
    {"source":15,"target":11,"value":1},
    {"source":17,"target":16,"value":4},
    {"source":18,"target":16,"value":4},
    {"source":18,"target":17,"value":4},
    {"source":19,"target":16,"value":4},
    {"source":19,"target":17,"value":4},
    {"source":19,"target":18,"value":4},
    {"source":20,"target":16,"value":3},
    {"source":20,"target":17,"value":3},
    {"source":20,"target":18,"value":3},
    {"source":20,"target":19,"value":4},
    {"source":21,"target":16,"value":3},
    {"source":21,"target":17,"value":3},
    {"source":21,"target":18,"value":3},
    {"source":21,"target":19,"value":3},
    {"source":21,"target":20,"value":5},
    {"source":22,"target":16,"value":3},
    {"source":22,"target":17,"value":3},
    {"source":22,"target":18,"value":3},
    {"source":22,"target":19,"value":3},
    {"source":22,"target":20,"value":4},
    {"source":22,"target":21,"value":4},
    {"source":23,"target":16,"value":3},
    {"source":23,"target":17,"value":3},
    {"source":23,"target":18,"value":3},
    {"source":23,"target":19,"value":3},
    {"source":23,"target":20,"value":4},
    {"source":23,"target":21,"value":4},
    {"source":23,"target":22,"value":4},
    {"source":23,"target":12,"value":2},
    {"source":23,"target":11,"value":9},
    {"source":24,"target":23,"value":2},
    {"source":24,"target":11,"value":7},
    {"source":25,"target":24,"value":13},
    {"source":25,"target":23,"value":1},
    {"source":25,"target":11,"value":12},
    {"source":26,"target":24,"value":4},
    {"source":26,"target":11,"value":31},
    {"source":26,"target":16,"value":1},
    {"source":26,"target":25,"value":1},
    {"source":27,"target":11,"value":17},
    {"source":27,"target":23,"value":5},
    {"source":27,"target":25,"value":5},
    {"source":27,"target":24,"value":1},
    {"source":27,"target":26,"value":1},
    {"source":28,"target":11,"value":8},
    {"source":28,"target":27,"value":1},
    {"source":29,"target":23,"value":1},
    {"source":29,"target":27,"value":1},
    {"source":29,"target":11,"value":2},
    {"source":30,"target":23,"value":1},
    {"source":31,"target":30,"value":2},
    {"source":31,"target":11,"value":3},
    {"source":31,"target":23,"value":2},
```

```
    {"source":31,"target":27,"value":1},
    {"source":32,"target":11,"value":1},
    {"source":33,"target":11,"value":2},
    {"source":33,"target":27,"value":1},
    {"source":34,"target":11,"value":3},
    {"source":34,"target":29,"value":2},
    {"source":35,"target":11,"value":3},
    {"source":35,"target":34,"value":3},
    {"source":35,"target":29,"value":2},
    {"source":36,"target":34,"value":2},
    {"source":36,"target":35,"value":2},
    {"source":36,"target":11,"value":2},
    {"source":36,"target":29,"value":1},
    {"source":37,"target":34,"value":2},
    {"source":37,"target":35,"value":2},
    {"source":37,"target":36,"value":2},
    {"source":37,"target":11,"value":2},
    {"source":37,"target":29,"value":1},
    {"source":38,"target":34,"value":2},
    {"source":38,"target":35,"value":2},
    {"source":38,"target":36,"value":2},
    {"source":38,"target":37,"value":2},
    {"source":38,"target":11,"value":2},
    {"source":38,"target":29,"value":1},
    {"source":39,"target":25,"value":1},
    {"source":40,"target":25,"value":1},
    {"source":41,"target":24,"value":2},
    {"source":41,"target":25,"value":3},
    {"source":42,"target":41,"value":2},
    {"source":42,"target":25,"value":2},
    {"source":42,"target":24,"value":1},
    {"source":43,"target":11,"value":3},
    {"source":43,"target":26,"value":1},
    {"source":43,"target":27,"value":1},
    {"source":44,"target":28,"value":3},
    {"source":44,"target":11,"value":1},
    {"source":45,"target":28,"value":2},
    {"source":47,"target":46,"value":1},
    {"source":48,"target":47,"value":2},
    {"source":48,"target":25,"value":1},
    {"source":48,"target":27,"value":1},
    {"source":48,"target":11,"value":1},
    {"source":49,"target":26,"value":3},
    {"source":49,"target":11,"value":2},
    {"source":50,"target":49,"value":1},
    {"source":50,"target":24,"value":1},
    {"source":51,"target":49,"value":9},
    {"source":51,"target":26,"value":2},
    {"source":51,"target":11,"value":2},
    {"source":52,"target":51,"value":1},
    {"source":52,"target":39,"value":1},
    {"source":53,"target":51,"value":1},
    {"source":54,"target":51,"value":2},
    {"source":54,"target":49,"value":1},
    {"source":54,"target":26,"value":1},
    {"source":55,"target":51,"value":6},
    {"source":55,"target":49,"value":12},
    {"source":55,"target":39,"value":1},
    {"source":55,"target":54,"value":1},
    {"source":55,"target":26,"value":21},
    {"source":55,"target":11,"value":19},
    {"source":55,"target":16,"value":1},
    {"source":55,"target":25,"value":2},
    {"source":55,"target":41,"value":5},
    {"source":55,"target":48,"value":4},
    {"source":56,"target":49,"value":1},
    {"source":56,"target":55,"value":1},
    {"source":57,"target":55,"value":1},
```

```
    {"source":57,"target":41,"value":1},
    {"source":57,"target":48,"value":1},
    {"source":58,"target":55,"value":7},
    {"source":58,"target":48,"value":7},
    {"source":58,"target":27,"value":6},
    {"source":58,"target":57,"value":1},
    {"source":58,"target":11,"value":4},
    {"source":59,"target":58,"value":15},
    {"source":59,"target":55,"value":5},
    {"source":59,"target":48,"value":6},
    {"source":59,"target":57,"value":2},
    {"source":60,"target":48,"value":1},
    {"source":60,"target":58,"value":4},
    {"source":60,"target":59,"value":2},
    {"source":61,"target":48,"value":2},
    {"source":61,"target":58,"value":6},
    {"source":61,"target":60,"value":2},
    {"source":61,"target":59,"value":5},
    {"source":61,"target":57,"value":1},
    {"source":61,"target":55,"value":1},
    {"source":62,"target":55,"value":9},
    {"source":62,"target":58,"value":17},
    {"source":62,"target":59,"value":13},
    {"source":62,"target":48,"value":7},
    {"source":62,"target":57,"value":2},
    {"source":62,"target":41,"value":1},
    {"source":62,"target":61,"value":6},
    {"source":62,"target":60,"value":3},
    {"source":63,"target":59,"value":5},
    {"source":63,"target":48,"value":5},
    {"source":63,"target":62,"value":6},
    {"source":63,"target":57,"value":2},
    {"source":63,"target":58,"value":4},
    {"source":63,"target":61,"value":3},
    {"source":63,"target":60,"value":2},
    {"source":63,"target":55,"value":1},
    {"source":64,"target":55,"value":5},
    {"source":64,"target":62,"value":12},
    {"source":64,"target":48,"value":5},
    {"source":64,"target":63,"value":4},
    {"source":64,"target":58,"value":10},
    {"source":64,"target":61,"value":6},
    {"source":64,"target":60,"value":2},
    {"source":64,"target":59,"value":9},
    {"source":64,"target":57,"value":1},
    {"source":64,"target":11,"value":1},
    {"source":65,"target":63,"value":5},
    {"source":65,"target":64,"value":7},
    {"source":65,"target":48,"value":3},
    {"source":65,"target":62,"value":5},
    {"source":65,"target":58,"value":5},
    {"source":65,"target":61,"value":5},
    {"source":65,"target":60,"value":2},
    {"source":65,"target":59,"value":5},
    {"source":65,"target":57,"value":1},
    {"source":65,"target":55,"value":2},
    {"source":66,"target":64,"value":3},
    {"source":66,"target":58,"value":3},
    {"source":66,"target":59,"value":1},
    {"source":66,"target":62,"value":2},
    {"source":66,"target":65,"value":2},
    {"source":66,"target":48,"value":1},
    {"source":66,"target":63,"value":1},
    {"source":66,"target":61,"value":1},
    {"source":66,"target":60,"value":1},
    {"source":67,"target":57,"value":3},
    {"source":68,"target":25,"value":5},
    {"source":68,"target":11,"value":1},
```

```
    {"source":68,"target":24,"value":1},
    {"source":68,"target":27,"value":1},
    {"source":68,"target":48,"value":1},
    {"source":68,"target":41,"value":1},
    {"source":69,"target":25,"value":6},
    {"source":69,"target":68,"value":6},
    {"source":69,"target":11,"value":1},
    {"source":69,"target":24,"value":1},
    {"source":69,"target":27,"value":2},
    {"source":69,"target":48,"value":1},
    {"source":69,"target":41,"value":1},
    {"source":70,"target":25,"value":4},
    {"source":70,"target":69,"value":4},
    {"source":70,"target":68,"value":4},
    {"source":70,"target":11,"value":1},
    {"source":70,"target":24,"value":1},
    {"source":70,"target":27,"value":1},
    {"source":70,"target":41,"value":1},
    {"source":70,"target":58,"value":1},
    {"source":71,"target":27,"value":1},
    {"source":71,"target":69,"value":2},
    {"source":71,"target":68,"value":2},
    {"source":71,"target":70,"value":2},
    {"source":71,"target":11,"value":1},
    {"source":71,"target":48,"value":1},
    {"source":71,"target":41,"value":1},
    {"source":71,"target":25,"value":1},
    {"source":72,"target":26,"value":2},
    {"source":72,"target":27,"value":1},
    {"source":72,"target":11,"value":1},
    {"source":73,"target":48,"value":2},
    {"source":74,"target":48,"value":2},
    {"source":74,"target":73,"value":3},
    {"source":75,"target":69,"value":3},
    {"source":75,"target":68,"value":3},
    {"source":75,"target":25,"value":3},
    {"source":75,"target":48,"value":1},
    {"source":75,"target":41,"value":1},
    {"source":75,"target":70,"value":1},
    {"source":75,"target":71,"value":1},
    {"source":76,"target":64,"value":1},
    {"source":76,"target":65,"value":1},
    {"source":76,"target":66,"value":1},
    {"source":76,"target":63,"value":1},
    {"source":76,"target":62,"value":1},
    {"source":76,"target":48,"value":1},
    {"source":76,"target":58,"value":1}
  ]
}
```

B Collapsible Tree Layout