State Space Visualization

Jan Friso Groote and Frank van Ham

Department of Computer Science, Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB, Eindhoven, The Netherlands
{jfg,fvham}@win.tue.nl

Abstract. Insight in the global structure of a state space is of great help in the analysis of the underlying process. We advocate the use of visualization for this purpose and present a new method to visualize the structure of very large state spaces. The method uses a clustering method to obtain a simplified representation, which is used as a backbone for the display of the entire state space. With this visualization we are able to answer questions about the global structure of a state space that cannot easily be answered by conventional methods. We show this by presenting a number of visualizations of real-world protocols.

1 Introduction

In the last decade, advances in computer hard- and software have made it possible to effectively analyze the behavior of complex software systems by means of state transition systems. Techniques based on explicit state enumeration can now deal with systems consisting of billions of states and have reached a scale at which they become sufficiently effective to assist in the design and testing of real world software systems. However, they also confront us with state transition graphs [1] of enormous dimensions, of which the internal structure is generally a mystery.

The most common approach to obtain insight in the structure of large state spaces is by abstracting from actions and/or state information, and by reducing the state space modulo a suitable equivalence. Suitable equivalences are trace, weak, branching or strong bisimulation for example. Although this approach maintains the behavioral aspect of state spaces, it destroys their structure. Typical questions that cannot be answered by using such techniques are:

- How many states are in each phase of a protocol?
- How many independent parts does the state space have, i.e. parts without a path between them? This question might be relevant to determine the effectiveness of different testing approaches.
- Are there hot-spots, i.e. groups of states that are visited relatively often when randomly traversing the state space? Are there parts of the state space that have extremely low probability to be visited?

On top of that, in many cases the reductionist approach mentioned above does not help in getting insight in the behavioral aspect of state spaces either. This is simply because even after abstraction, state spaces are often larger than a few hundred states, which is well above the limit of current graph drawing packages such as *dot* or *neato*

[4] where each state and each transition is explicitly drawn. The images they produce suffer from information overload, cluttering and generally take too long to generate. A recent method by [8] is able to layout graphs of millions of nodes in a matter of minutes, but is only effective for highly regular grid-like graphs. This leaves us with analysis techniques like simulation, testing, model checking and the verification of behavioral equivalences. Although such techniques are very suitable to answer all kinds of questions about state spaces, they do not provide insight in the structure of the state space itself.

In this paper we present a number of applications of a novel technique [6] that can effectively display state transition systems consisting of millions of states and clarify their structure in this way. Essentially, the scalability of the technique is limited by the capacities of the graphics hardware. With the fast development of graphics hardware, our visualization technique will soon be suitable for even larger state spaces. Besides being scalable, this technique is also computationally inexpensive and very predictable (that is, local changes in structure do usually not lead to global changes in visualization). A prototype version is freely available from [7].

Currently, input state spaces are generated on the basis of behavioral specifications in μCRL [3] from which state spaces in SVC-format [9] are produced. In these state spaces transitions are labeled with an action and states are labeled with a vector of data values. The basic idea underlying the visualization technique is to use a simplified representation in the form of a tree as a *backbone* for the entire structure. First the state space is layered using the distance of each node to the root. Then, the tree structure is obtained by clustering sets of states in each layer, such that for each set a unique path to the root is obtained. Each set of states is subsequently modeled using a disk shape in a three dimensional space. Finally, all disks are connected in a manner resembling cone trees [12], forming the shapes such as the ones in figures 2, 3 and 4. Note that visual cues such as interactive motion, colors, lighting and transparency all add strongly to the three dimensional perception of the shapes, the black-and-white still pictures in this print are by no means comparable to the onscreen images. After visualizing the state space as a tree shaped 3D-object we can use coloring to stress particular aspects of the state space. Typically, coloring can be induced by intrinsic properties such as the value of the transition label or statevector, or derived properties, such as the probability to visit a state during a random walk.

In the next section we elaborate on the clustering method mentioned above. Section 3 presents visualizations of real world examples, and we conclude in section 4.

2 The Visualization Algorithm

Since it is impossible for a human to cope with pictures containing hundreds of thousands of data elements, no matter what visualization method we use, it is unavoidable to bring a state space back to a manageable number of elements if we wish to visualize it. Where conventional methods generally reduce a state space to a smaller one based on behavioral information, we do not apply any reduction and use only structural information to cluster states together.

A state transition system can be defined as a 4-tuple (S, Act, T, s) where S is the set of states, Act is a set of possible actions, the relation $T \subseteq S \times Act \times S$ is a labeled transition relation and $s \in S$ is the systems initial state. Each state consists of a vector of parameter values. We can represent the corresponding state transition graph by a graph (V, E, s) where a node $x \in V$ represents a state and $a_{xy} \in E$ ($E \subseteq V \times V$) represents a *directed* edge (or *arc*) between the nodes x and y. An arc exists for every transition in the corresponding state transition system.

The first step in the reduction process is to assign each individual node a non-negative layer or rank. The most common method is to assign each node x a rank $Rank(x)$ depending on its distance from the start node, taking edge direction into account. But other ranking algorithms are also possible, such as using a similar layering that ignores edge direction. In the remainder of this paper we refer to arcs running from a low ranked node to a higher ranked node as arcs pointing downward and draw these accordingly. In our ranking all connections between consecutive ranks point downward, but undesirable long up-arcs spanning more than one rank (we call these *backpointers*) are frequent. These backpointers tend to spoil the resulting visualization, so in our application we can interactively choose to display or hide them.

In a second step we cluster nodes based on an equivalence relation. To facilitate the definition of this relation we slightly modify our edge set E by removing all backpointers and reversing the direction of all other up-arcs (if any). In our resulting modified edge set E' all arcs are now either down arcs or arcs connecting equally ranked nodes. Let $D(x)$ be the set of all nodes that can be reached from node x via zero or more arcs in E'. We now define two nodes x and y to be equivalent iff there exists a sequence $x=z_1,z_2,...,z_N=y$ of nodes with equal rank, such that for all $1 \le i < N$ it holds that $D(z_i) \cap D(z_{i+1}) \neq \varnothing$ (see Fig. 1).

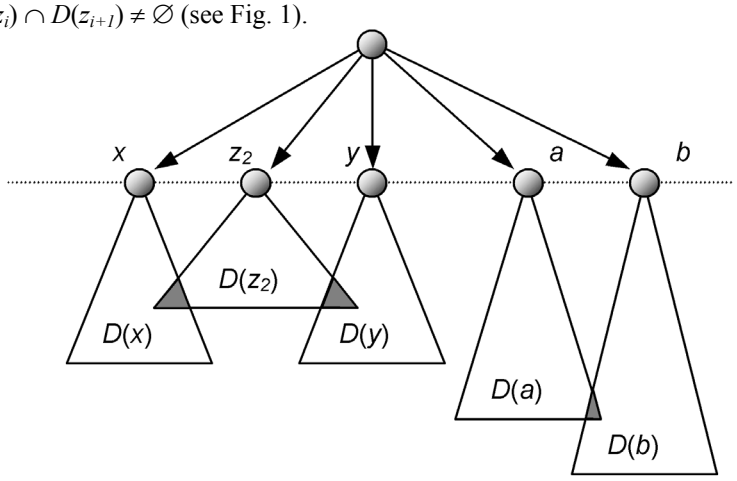


Fig. 1. Nodes x and y are equivalent, whereas nodes x and a are not.

We use the resulting equivalence classes as clusters, and define a relation between clusters by extending the concept of rank from nodes to clusters, that is, the rank of a cluster C is equal to the rank of any node in C. We now define a cluster C_1 to be an ancestor of a cluster C_2 iff $Rank(C_1) = Rank(C_2) - 1$ and there exists an arc in E' con-

necting a node in C_1 with a node in C_2. A cluster C_2 is defined to be a descendant of C_1 iff C_1 is an ancestor of C_2. The resulting cluster structure forms a tree with the cluster containing the startnode at the root.

We visualize this backbone structure by representing each cluster by a horizontal disc, with the disc diameter proportional to the number of nodes in the cluster. Descendant clusters are positioned in a symmetrical manner below their parent cluster. This results in a three dimensional structure of discs, with clusters having the same rank being positioned in the same horizontal plane. To improve the display we connect clusters in consecutive ranks by drawing a knotted cone between them and slightly rotating specific branches of clusters outward. The actual states in a cluster are visualized as small spheres and distributed over the disc edge. The exact procedure is explained in full detail in [6].

3 Examples

This section contains some examples of situations in which a visualization of a finite automaton provides useful info, which would be very hard to extract using conventional methods. The first example consists of two simple, similar automatae, and is meant to illustrate the algorithm outlined above. The second and third are examples of real-world industrial protocols.

3.1 The Alternating Bit Protocol and PAR Protocol.

The Alternating Bit Protocol (ABP) is a communication protocol concerned with data transmission over an unreliable channel [2]. In this section we present visualizations of the ABP and one of its variations, the Positive Acknowledgement with Retransmission (PAR) Protocol [11]. Fig. 2 shows different visualizations of models of both protocols using two different data elements in the transmission. Individual states are depicted as spheres, transitions between states are depicted as lines, while backpointers are depicted as arcs. A knotted cone between a cluster and its multiple descendants is rendered more transparent and shows up lighter than the other cones. The start node is always located at the top of the visualization.

All visualizations depicted in Fig. 2 show a high degree of symmetry in the vertical axis, since the behavior for both data elements is identical. Note that the visualization in Fig. 2a consists of more states than the one in Fig. 2b, even though they both describe the exact same protocol. This is due to the fact that the software used to extract the state space from its formal description suffers from state duplication. This is especially obvious in the bottom part of the left visualization, which duplicates itself. This can be remedied by applying a strong bisimulation reduction on the left state space, which gives us the state space in Fig. 2b. This visualization clearly displays regularities in the alternating bit protocol. The behavior in the top half is identical to the behavior in the bottom part, with the distinguishing factor being the control bit sent with the data. The four protrusions on both sides of the visualization represent errors in the transmission of the datum (1^{st} and 3^{rd} from the top) or errors in the acknowledgement of a transmission (2^{nd} and 4^{th} from the top).

The PAR protocol globally resembles the ABP, but differs in the way errors are handled. Where the ABP assumes non-lossy channels, the communication channels in PAR can loose data. To cope with this additional problem a timer is integrated in the protocol, which generates a time-out on loss or corruption of a message. This explains why the 1st and 3rd protrusions are much smaller, since PAR deals with errors in transmission more efficiently than ABP: On loss or corruption a timeout is delivered directly to the sender, which can then immediately retransmit. The ABP has to use the acknowledgement channel to communicate this. The handling of errors in acknowledgement transmission (2nd and 4th protrusion) is very similar, except for the fact that PAR has an additional possibility of message loss, accounting for the wider section in the visualization, and the fact that errors while resending the datum can be dealt with quicker thanks to the timeout (accounting for the shorter loop inside the PAR error handling).

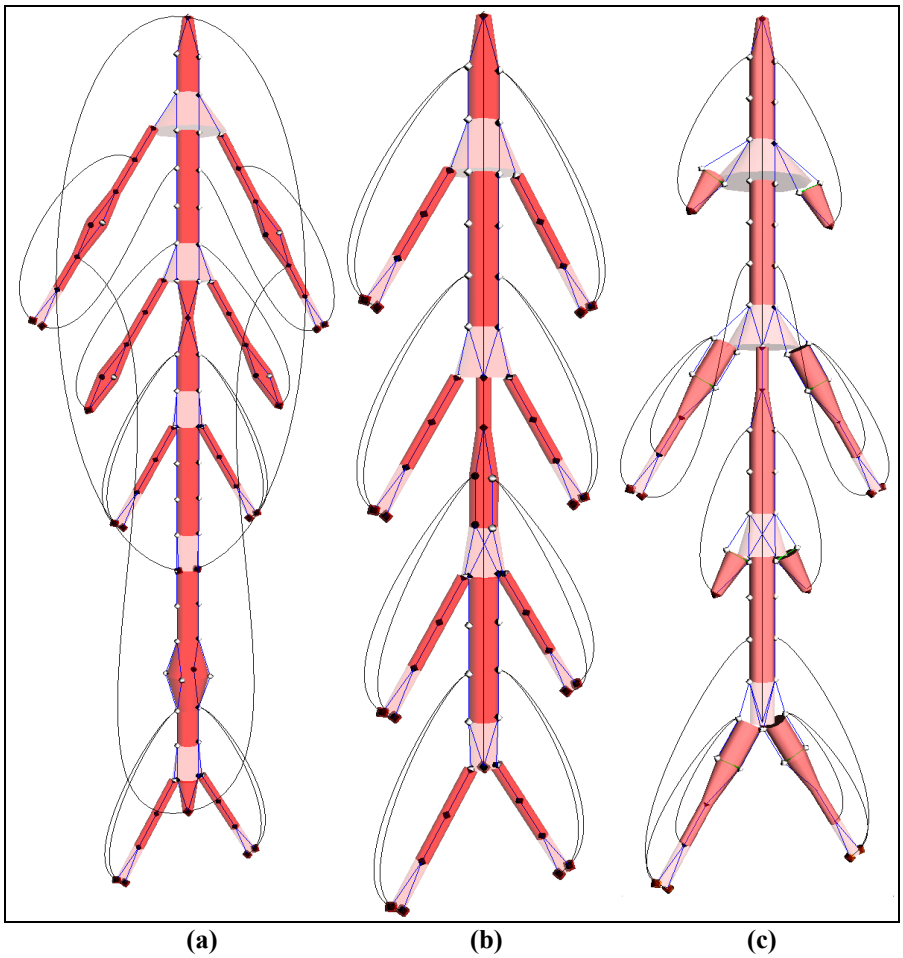


Fig. 2. Communication Protocols: ABP (*a*), ABP reduced modulo strong bisimulation (*b*) and PAR reduced modulo strong bisimulation (*c*)

3.2 A Modular Jack System

This section deals with the communication protocol for a modular jack system. The protocol regulates the communication between an expandable set of industrial jack platforms, allowing the entire set of jacks to be operated from the controls of any one jack. All jacks are communicating via a ring-shaped shared bus. The protocol was formally modeled and analyzed in [5] and the exact formulation of the protocol that we visualize is distributed with the μCRL toolset [3]. Fig. 3 shows the visualization of the protocol when it has to synchronize two separate jack platforms. One of the immediate features is the existence of two separate but symmetric legs. Although this feature is very apparent from the picture, the researchers involved only realized this when observing the picture. Generally, the protocol for *k* jacks has *k* independent legs.

Another feature that is hard to extract using conventional analysis is the fact that the protocol clearly starts with an initialization phase, the end of which is marked by a large number of returning backpointers. (See arrows – Fig. 3) Looking at the formal description the initialization phase assigns a consecutive global numbering to all jacks, with each jack having equal opportunity to become the first. This also explains why the two sections are symmetrical, after all the observable behavior of the entire setup should be insensitive to the numbering scheme used. To illustrate this in Fig. 3, we marked those states in the system in which a jack has been numbered 1 (dark area in the right leg). This implies that in the remaining states this jack has been numbered 0, and that, during the initialization both jacks are numbered 0.

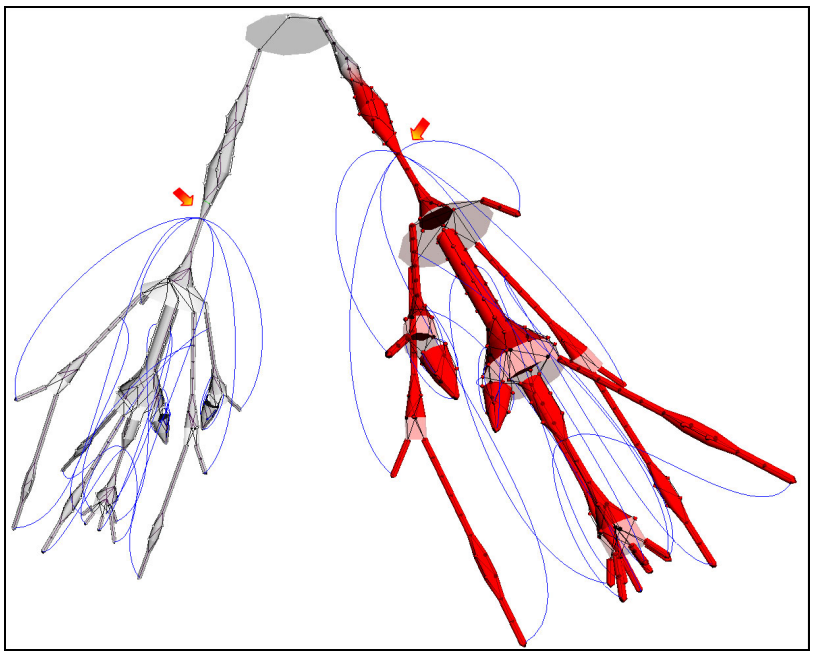


Fig. 3. Modular jack communication protocol for 2 jacks.

Another interesting observation is that the general behavior of a system of two platforms is visually very similar to the behavior of a system of three platforms, consisting of approximately 3,000 states. Fig. 4 shows such a behavior with backpointers hidden. Notice the three identical sections and how different subsections of the graph visually correspond to sections of the two-platform graph. The fact that similar graphs give similar pictures is a major advantage of this method, allowing the application of insight gained in simplified versions of behavior to more complex behaviors. In Fig. 4 we also depicted the full state spaces of five and seven communicating jack platforms, the latter consisting of over 1 million states.

This visualization also shows a bug in the protocol: the small encircled section that splits off during the initialization phase of the protocol has no returning backpointers, which means that states at the end of this appendix are deadlock states. This was not the case in the two-jack version. Note that these features can also easily be spotted by conventional analysis (actually, in this case they have been, see [5]), nevertheless a picture in which you can actually point out the bug is a great asset in communication.

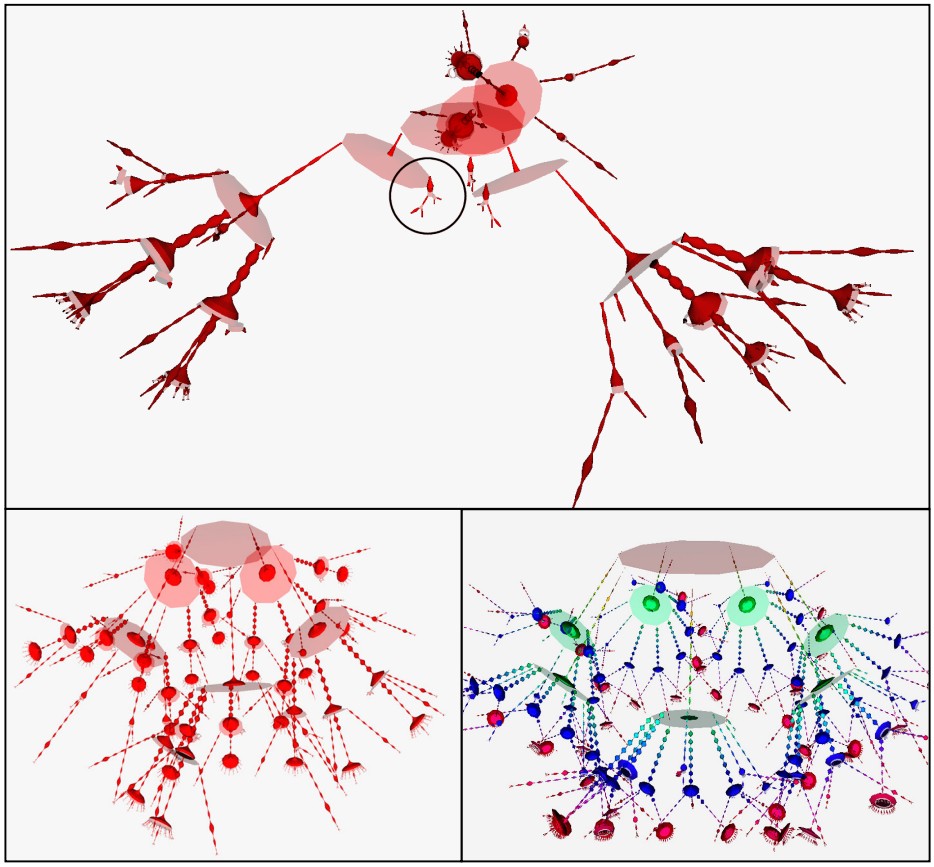


Fig. 4. Protocols for a setup of 3 jacks (2860 states), 5 jacks (70,926 states) and 7 jacks (1,025,844 states) respectively. Backpointers are hidden.

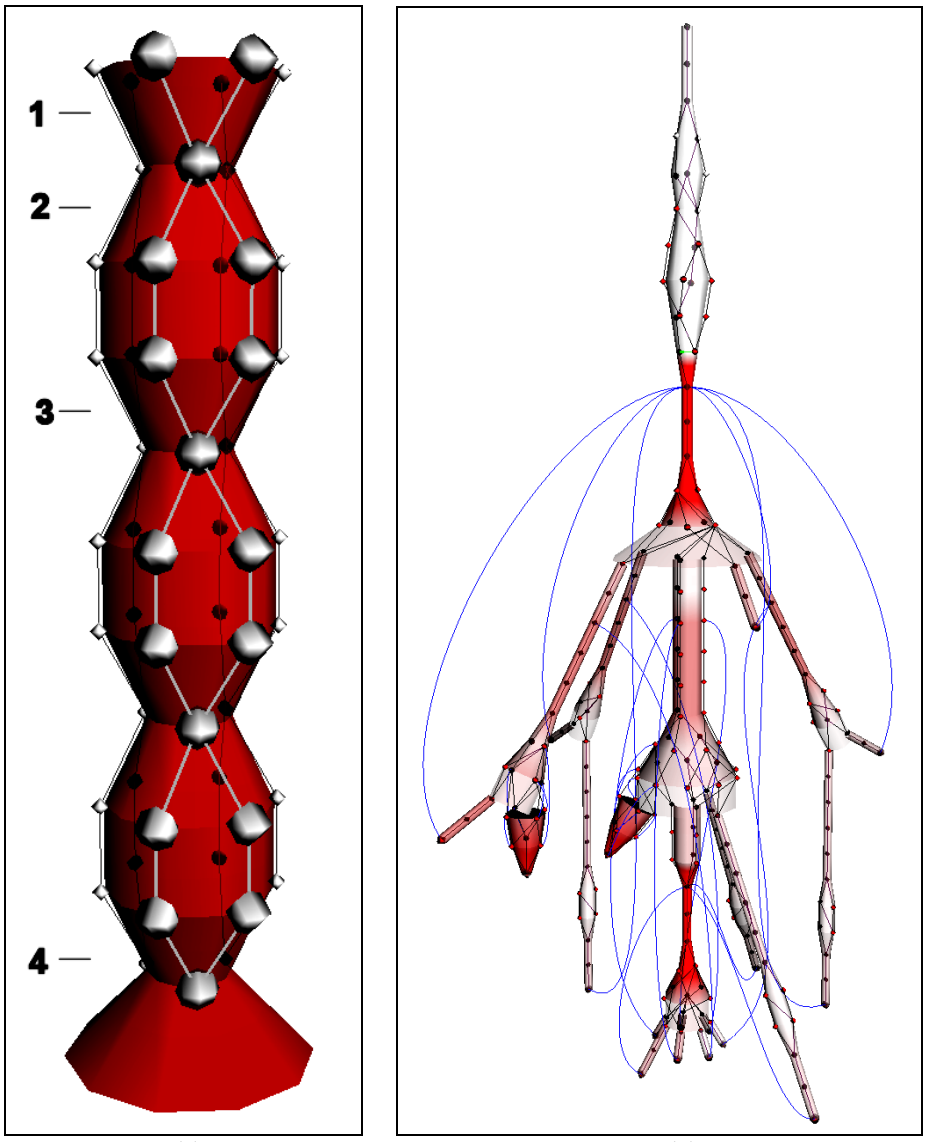


Fig. 5. Detail section of three-jack protocol (*a*) and stochastic analysis of two-jack protocol (*b*).

We can also use this visualization method to inspect sections of the protocol in detail. Fig. 5a shows a section that deals with the bus communication of the three-jack system. When one of the jacks is instructed to perform a common action (i.e. move all jack platforms up synchronously), all jacks are first brought into standby mode. From the top two states (which are actually bisimilar) the first jack broadcasts a "*ready for standby*" message to all other jacks (*1*). These messages can be received by the other jacks in any order in which explains the branching at (*2*). In the two (bisimilar) top

states at (*3*) both other jacks have received the message. The next jack in line broadcasts its "ready for standby" message at (*3*) and this process repeats itself for every jack in the system. Finally, at (*4*) the originating jack broadcasts a "*all move up*" message to the other jacks. In the five and seven jack systems in Fig. 4 this typical communication pattern is also abundant, which leads us to estimate that in these graphs at least 50% of the states is related to communication activities.

Another interesting feature is the ability to perform stochastic analysis on different automata. Assuming each transition has an equal probability of occurring, we can gain insight on which states in the process have a relatively high probability of being visited by simulating a random walk. We can then visualize this information by coloring clusters based on these probabilities. Fig. 5b shows such a visualization, based on one part of the two leg jack protocol. In this case high probability (dark) sections are located directly behind incoming backpointers and in the two small extrusions on the left of the picture, due to the large number of short cycles there. These types of pictures are even more useful if measured real-world probability data is used. This would make it possible to make a clear distinction between states that are in those parts of the automaton that deal with, for example, error handling and states that are part of the main body of the process.

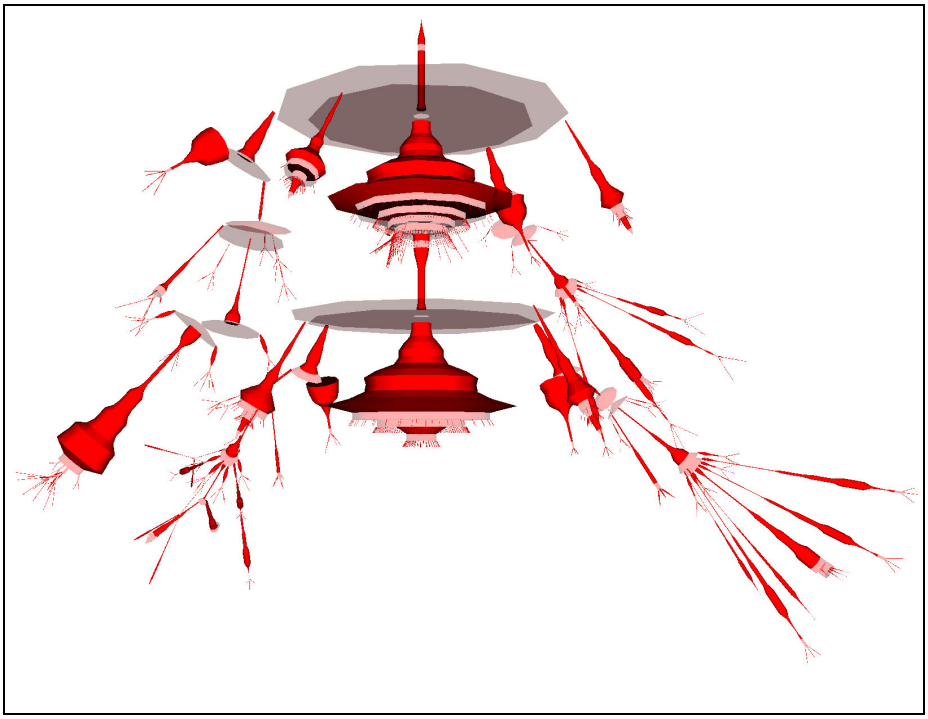


Fig. 6. Visualization of Link layer of IEEE 1394 (25,898 states).

3.3 The Link Layer of IEEE 1394

The final example deals with the link layer of the IEEE 1394-1995 (also known as FireWire or I. Link) communication protocol. FireWire is a widely used high-speed serial protocol. Some effort was put into formally analyzing FireWire, resulting in a formal model of the link layer, which provides an interface between the high-level transaction layer and the physical layer [10]. Based on this model, the state transition system of two FireWire nodes sharing a common databus was generated, in which we abstracted from the data being transferred. We ended up with a state space of approximately 25,000 nodes, which is shown in Fig. 6.

One of the most notable features of this visualization is the similarity between the two 'bulks' at the top and bottom. These are communication phases, since the number of states locally expands and then converges again. Both are separated by a relatively thin funnel, consisting of a comparably small number of states. This type of information is not easily extracted by conventional methods, yet can be very useful for testing purposes. During a test run it makes sense to flag states in the funnel to see if these are reached, which indicates the lower bulk is being tested too. Also, testing in general lends itself well for visualization, since it probably would be desirable to run test traces that cover as much of the state space as possible. A visualization technique such as this one could make it much easier to see what parts of the state space are already covered by tests, for example by rendering a trace in the actual visualization (Fig. 7).

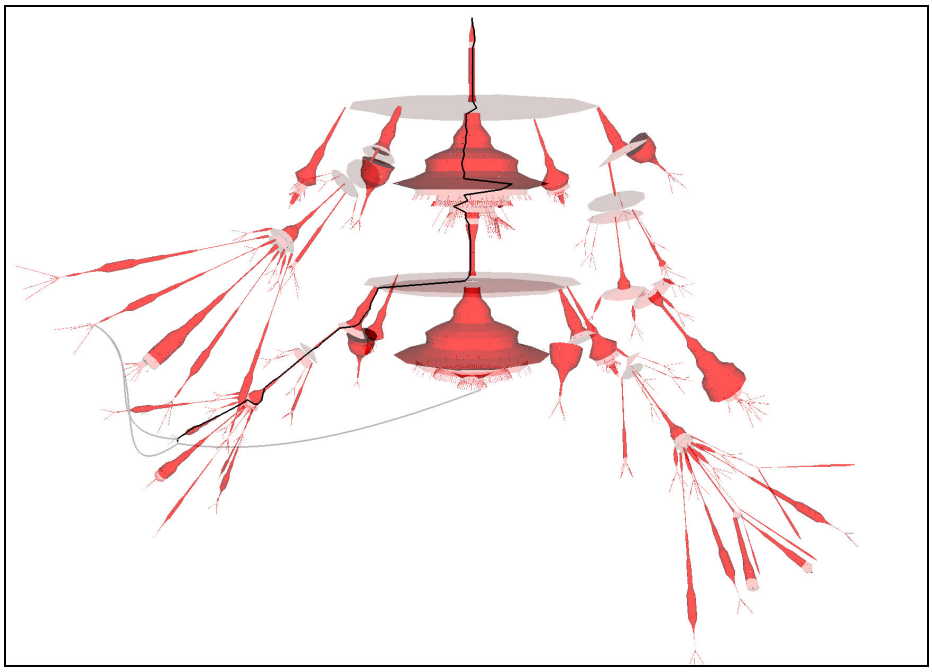
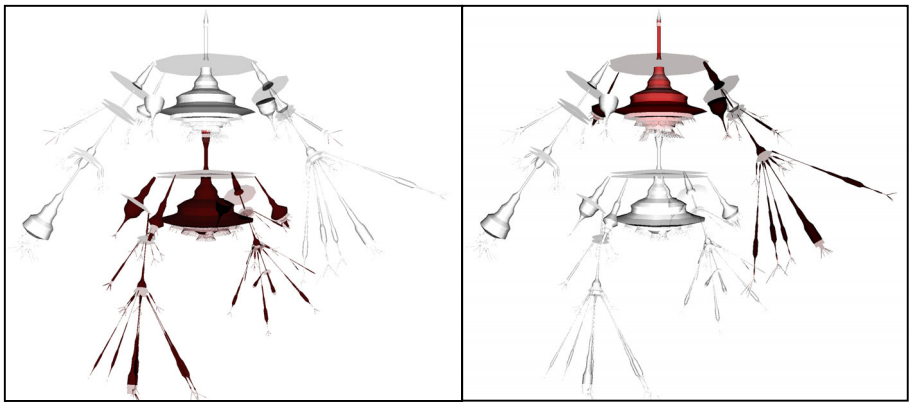


Fig. 7. Rendering a test trace in a visualization.

Another question that comes to mind is why we are seeing two similar bulks. Although the general behavior for both is almost identical, their state vectors differ. If we could determine a value of a state variable that is unique for the bottom bulk, we could pin a semantic meaning to that specific part of the graph. An effective way to accomplish this is to select a part of the visualization and subsequently correlating states in this part with the set of states with a particular value for a parameter. If we do this for all possible combinations of state parameters and values we end up with a list of correlation factors, of which the ones closest to 1 indicate the best match. Applying this method on the nodes in the bottom part gives us one parameter/value combination that is an almost perfect match. These states are marked in figure 8a.

Looking back at the formal specification we observed that the parameter in question was a Boolean array *b* of length two that kept track of which nodes have already requested use of the bus during a so called *fairness interval*. A fairness interval is a fixed amount of time during which all connected nodes should have had the opportunity to use the shared bus if they needed to. All nodes in the marked (darker) set in Fig. 8a have a value of [*true, true*] for *b* indicating that in the bottom bulk both nodes have already requested the bus during the current fairness interval. From this information we can deduce that in the top part a single node has requested use. To illustrate this figure 8b shows the collection of states where *b* = [*false,true*].



(a) **(b)**

Fig. 8. States where state variable *b* = [*true,true*] (*a*) and states where *b* = [*false,true*] (*b*)

Although we expect the top bulk to be symmetrical, in fact it is not. The fan-like marked section in figure 8b for example is not repeated in the top section. There are subtle differences in symmetry, which means that the simulated behavior of the system differs slightly based on which one of the two nodes requests the bus first in the upper bulk of the protocol. Whether this difference lies in the protocol itself or in the formal description of the protocol turns out to be a question that neither we, nor the author of [10] have yet been able to answer.

4 Conclusions

In this paper we have made a case for the use of a new visualization technique to gain more understanding of large state spaces. Its main advantages over existing contemporary visualization techniques are:

- *High scalability*: by not displaying each individual state and transition we can effectively visualize a number of nodes that is at least two orders of magnitude larger than the best conventional techniques. On a medium desktop PC with a high-end graphics card we were able to display visualizations of graphs consisting of millions of nodes. At the same time, the user is still able to interactively zoom in on parts of interest and view individual transitions if needed.
- *Predictability*: In contrast to some other popular graph layout approaches (for example force directed methods or simulated annealing) this method is highly predictable. As a consequence, similar input graphs also lead to similar looking visualizations.
- *Speed*: The performance of the method is currently limited by available memory and the speed of the graphics card. The algorithm that generates the global layouts has a time complexity that is linear in both the number of states and the number of edges. This currently allows us to preprocess a graph of 300K states for visualization in approximately 16 seconds. The visualization itself is instantaneous.
- *Interactivity*: Since it is impossible to display all information related to a state transition graph in a single picture, interaction is critical. In our application the user is enabled to zoom into subsections of interest, color parts of the system based on state vector values or transition label values, perform stochastic analysis and much more.

This does not mean however, that this technique is the ultimate answer to every question one might have for a specific state transition graph. As with any visualization technique it is simply meant as a support tool, to be used complementary to other techniques. Nevertheless, having a tool that is capable of producing a meaningful image of very large state spaces greatly enhances general understanding of these state spaces.

Unfortunately the method also has its weak points. Firstly, it does not deal well with large graphs that are highly connected. In this case nodes tend to clutter together in a very small number of clusters, which gives no information on the internal structure of the graph. Secondly, the layout algorithm used to position individual nodes in a cluster does not always produce an optimal layout. Finally, we found that researchers that we confronted with visualization of their own state spaces generally had a hard time interpreting the pictures, because in most cases they had no reference as to what the state space they were analyzing should look like. The option to mark sections of the visualization based on a specific state parameter or transition label proved a great help in relating the original specification to the shape of the visualization.

Future work on this method will focus on integrating state reduction techniques such as abstraction and bisimulation into the visualization. One can think of the possibility of interactively collapsing similar looking subsections or a visualization of a large state space morphing into a smaller version. We also plan to look into recur-

sively applying this method to more complex sections of the graph, which may split large clusters into a number of smaller ones.

It is needless to say that we are enthusiastic about the techniques we offer. We have not seen any technique that even resembles what we do, and we sincerely believe that this visualization technique is a great step forward in understanding the global structure of state-transition diagrams. Actually, by looking at and interacting with the visualization tool we became aware of many properties of state spaces that, although sometimes obvious and sometimes more obscure, we had not realized until we saw them.

Acknowledgements

Thanks go to Jack van Wijk and Huub van de Wetering, for their valuable discussions ,comments and programming work. The authors would also like to thank the Netherlands Organisation for Scientific Research (NWO) for their financial support under grant 612.000.101.

References

1. A. Arnold, *Finite Transition Systems*, Prentice Hall, 1994.
2. J. Baeten and P. Weyland. *Process Algebra*. Volume 18 of Cambridge Tracts in Theoretical Computer Science. Cambridge Univ. Press, Cambridge, 1991.
3. S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. van Langevelde, B. Lisser, and J.C. van de Pol, μCRL: A Toolset for Analysing Algebraic Specifications. *Computer Aided Verification (CAV 2001)*, pp. 250-254, LNCS 2102.
4. E.R. Gansner and S.C. North. An Open Graph Visualization System and Its Applications to Software Engineering. *Software – Practice and Experience* 30(11), pp 1203-1233, 2000.
5. J.F. Groote, J. Pang and A.G. Wouters. A Balancing Act: Analysing a Distributed Lift System. Technical Report, Dept. of Software Eng., CWI, Amsterdam, The Netherlands (to appear).
6. F. van Ham, H. van de Wetering and J.J. van Wijk. Visualization of State Transition Graphs. *Proc. IEEE Conf. Information Visualization '01*, pp 59-66, 2001.
7. F. van Ham, Interactive Visualization of State Transition Systems, Project website at http://www.win.tue.nl/~fvham/fsm/.
8. Y. Koren, L. Carmel and D. Harel. ACE : A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs. Technical Report MCS01-17, The Weizmann Institute Of Science, 2001. Available from http://www.wisdom.weizmann.ac.il/reports.html.
9. I.A. van Langevelde. A compact file format for labeled transition systems. Technical report SEN-R0102, CWI, Amsterdam, The Netherlands, 2001.
10. S.P. Luttik. Description and Formal Specification of the Link Layer of P1394. Technical Report SEN-R9706, CWI, Amsterdam, The Netherlands, 1997.
11. S. Mauw and G.J. Veltink (eds). *Algebraic Specifications of Communication Protocols*. Volume 36 of Cambridge Tracts in Theoretical Computer Science. Cambridge Univ. Press, Cambride, 1993.
12. G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Human Factors in Computing Systems, CHI '91 Conf. Proc.*, pp. 189-194, 1991.