GSK: Universally Accessible Graph SKetching

Suzanne Balik, Sean Mealin, Matthias Stallmann, and Robert Rodman
Dept. of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
{spbalik, spmealin, mfms, rodman} @ncsu.edu

ABSTRACT
Combinatorial graphs, often conveyed as node-link diagrams, figure prominently in Computer Science and other Science, Technology, Engineering, and Mathematics (STEM) disciplines. Unfortunately, they are most often inaccessible to blind students and professionals. This paper introduces GSK, a self-contained Graph SKetching tool that allows blind and sighted people to easily create, edit, and share graphs in real-time using interaction mechanisms (mouse, keyboard, monitor, screen reader) that are standard for them. GSK was successfully used by a blind Computer Science student and his sighted instructors to create and access graphs specific to his automata theory and operating systems courses. Our hope is that GSK will enable more blind STEM students and professionals to actively participate in their disciplines by providing them and their sighted colleagues with a cross-collaboration tool that allows them to share graphs just as easily as they share text and word processing documents.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces*;* K.4.2 [**Computers and Society**]: Social Issues – *assistive technologies for persons with disabilities;* K.3.2 [**Computers and Education**]: Computer and Information Science Education; G.2.2 [**Discrete Mathematics**]: Graph Theory .

General Terms
Human Factors.

Keywords
Universal Design, Accessibility, Computational Equivalence.

1. INTRODUCTION
We developed GSK, a self-contained Graph SKetching tool, to provide a universally accessible means of creating, editing, and sharing combinatorial graphs, also known as node-link diagrams. A combinatorial graph consists of nodes joined by edges. While a very simple structure, it figures prominently in Computer Science and other Science, Technology, Engineering, and Mathematics (STEM) disciplines. Entity relationship diagrams, finite-state machines, various Universal Modeling Language (UML) diagrams, resource-allocation graphs, chemical molecules, etc., are all represented using (combinatorial) graphs.[1] Unfortunately, these representations are generally not accessible to blind students and professionals forming a barrier between the blind user and active participation in STEM fields. Our goal is to help remove that barrier by providing blind people with the means to independently create and use graphs in a variety of disciplines for knowledge representation and problem solving, and to easily share them with sighted people.

SIGCSE'13, March 6–9, 2013, Denver, Colorado, USA.

2. DESIGN CONSIDERATIONS
The principles of *universal design* and *computational equivalence* as well as a desire to *level the playing field* for blind people guided our design of GSK.

2.1 Universal Design
Universal design principles espouse the development of products that are equitable in use without segregating users [3]. A number of applications have been developed *separately* and *specifically* to provide blind users with access to various types of graphs [1, 2, 4, 6, 7, 9]. However, separate is most often not equal and it is important that blind users be included in software applications intended for universal use. Work is now underway to include blind students in the use of JFLAP, a popular automata theory platform [5]. Our prior experience with making our graph-based automata theory application, ProofChecker, accessible to blind students without affecting the sighted interface [11] inspired us to do the same for generalized graph editing.

Text editors and word processing programs are useable by both sighted and blind people via interaction mechanisms that are standard for them (mouse, keyboard, monitor, screen reader). When a blind and a sighted person are working together on a text or word processing document, they have immediate access to changes made by the other. GSK is designed to provide the same type of experience for graph editing for both blind and sighted people in terms of ease of use and document sharing.

2.2 Computational Equivalence
In order to use graphs for problem solving, it is important that blind people have graph representations that are *computationally equivalent* and not simply *informationally equivalent* to those available to sighted people. Larkin and Simon distinguish between informational and computational equivalence as follows:

Two representations are informationally equivalent if all of the information in the one is also inferable from the other, and vice

[1] A recent issue of the Communications of the ACM (June 2011) contained over 20 graphs representing networks, structural equation analysis, page ranking, record assembly, and verification pipelines, among other things, within its 128 pages.

*versa. Each could be constructed from the information in the other. Two representations are computationally equivalent if they are informationally equivalent and, in addition, any inference that can be drawn **easily and quickly** from the information given explicitly in the one can be drawn easily and quickly from the information given explicitly in the other, and vice versa.* [8]

Oftentimes graphs are provided to blind people in the form of tables which are informationally equivalent, but not generally computationally equivalent to the node-link diagrams typically used by sighted people. As one blind student studying finite-state machines (FSMs) succinctly put it, he was "stuck in table mode" while his sighted peers had the advantage of easily moving from state to state of an FSM by following the edges (transitions) of a node-link diagram representation.

To further illustrate this point, consider the following example from automata theory [11]. A deterministic finite automaton (DFA) is a form of directed graph that is used to determine whether or not a string (sequence of characters) is a member of a regular language. A DFA is said to "accept" or "reject" a string in terms of membership in its language. Table 1 and Figure 1 below provide *informationally equivalent* representations of the same DFA, i.e., each could be created from the information provided in the other.

Table 1. Transition table representation of a DFA [11]

δ	a	b
q_0	q_2	q_3
*q_1	q_0	q_1
q_2	q_0	q_1
q_3	q_2	q_3

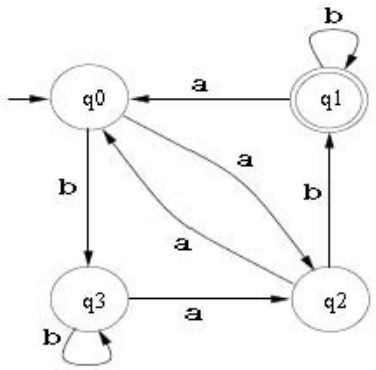

Figure 1. Node-link diagram representation of a DFA [11]

In this example, the alphabet Σ = {a, b} and the set of states Q = {q_0, q_1, q_2, q_3}. The first state in the transition table, q_0, is the start state and is designated as such by an incoming arrow in the node-link diagram. The accepting state, q_1 is preceded by an asterisk (*) in the table and denoted by a double circle in the diagram. This DFA may be used to tell whether or not a string consisting of 0 or more a's and b's has an odd number of a's and ends with b.

For example, determining whether the string *abbabab* is accepted by the DFA above involves moving from state to state while reading the current input symbol as illustrated by the following sequence of configurations:

(q_0, abbabab) ⊢ (q_2, bbabab) ⊢ (q_1, babab) ⊢ (q_1, abab) ⊢

(q_0, bab) ⊢ (q_3, ab) ⊢ (q_2, b) ⊢ (q_1, ε)

Because q_1, the last state reached in the computation, is an accepting state, the string is indeed in the language of the DFA.

Imagine a sighted person carrying out the above computation by moving a finger back and forth and up and down the rows of the transition table, which is analogous to the way a blind person would access an electronic form of the table. Using a table, the time to determine the inclusion of a string *w* in the language of a DFA is proportional to the number of states times the size of its alphabet times the length of the string, (|Q| |Σ| |*w*|). On the other hand a sighted person moving a finger from state to state of a node-link diagram following the transition corresponding to the current input symbol could complete the computation in time proportional to the size of the alphabet times the length of the string, (|Σ| |*w*|). For a DFA with a large number of states, the time savings would be considerable. Thus, the two representations are *not computationally equivalent*.

Our design of GSK aims to relieve blind students and professionals of the tedium of using tables to solve graph-related problems and instead provides them with the computational advantage of being able to easily traverse the edges of a graph and otherwise efficiently create, edit, and access graphs.

2.3 Leveling the Playing Field

Besides providing blind people with computationally equivalent graph representations, we would like to help enable them to function as independently and efficiently as possible in academic and professional settings. All of the graphs shown in Figures 2, 7, 8, and 10 were drawn independently by a blind student, who is also the second author of this paper. Using GSK, he was able to communicate graphs with his automata theory instructors, create and include graphs in documents for homework assignments, and quickly render the graphs necessary to solve problems on an operating system quiz. Our hope is that GSK will help to level the playing field for more blind students and professionals allowing them to fully participate in their disciplines alongside their sighted peers.

3. OVERVIEW OF GSK

GSK is a Java application built using standard Swing components that leverages the Java Accessibility API. It was developed and tested on a Windows platform using the JAWS screen reader, which requires the installation of the Java Access Bridge. The bridge provides a communication mechanism between the screen reader and the Java Virtual Machine (JVM). Being written in Java, our hope is that GSK would port well to other systems and use by other screen readers.

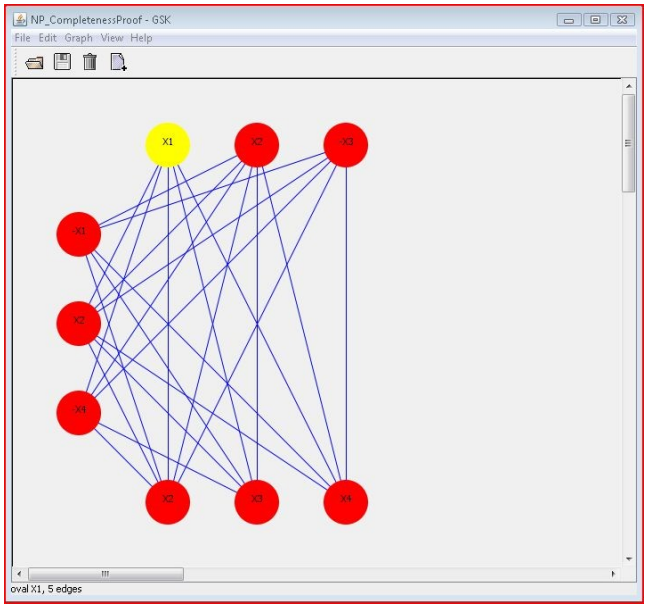

Figure 2. Connection View of an NP-Completeness Graph

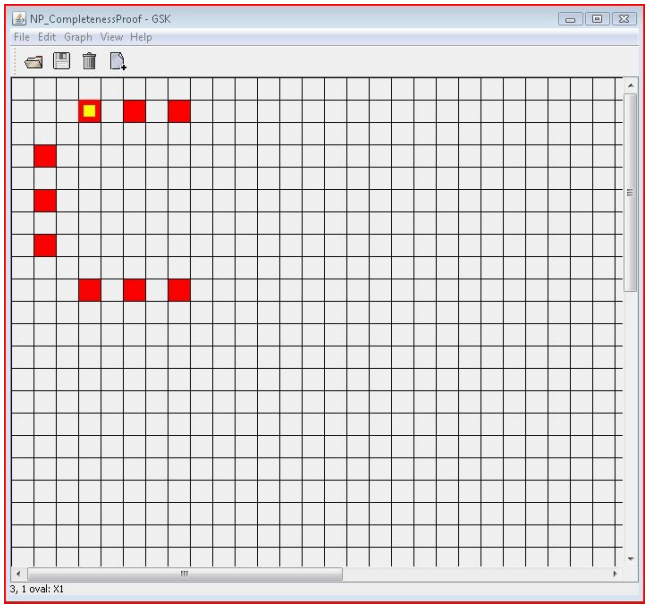

Figure 3. Grid View of Graph in Figure 2

GSK was developed as a collaboration between the sighted first author and the aforementioned blind second author, which was itself an exercise in universal accessibility. We found that the Eclipse platform using SVN for version control worked well for both of us. The ability provided by Eclipse to specify the indentation level of our source code and correct the indentation when necessary was particularly helpful. We used Microsoft Word to prepare our initial design documents and a raised print drawing kit whenever we needed to share a drawing of the interface, control flow, etc. Eventually, we were able to use GSK instead of the drawing kit to create and share simple flowcharts that we used in the design of subsequent functionality.

3.1 Universal Accessibility Features

GSK is intended to appeal to both sighted and blind users. In addition to providing a point-and-click interface, GSK is fully *keyboard accessible* which is crucial for blind users and in fact required by Sec. 508 accessibility guidelines [10]. This helps make GSK useable by a wide group of users including those with mobility or other impairments.

Providing *programmatic focus* and an effective *navigation scheme* is also crucial for blind users. While sighted users can easily rest their gaze on any item on the screen, blind users must use the keyboard to navigate from item to item. When using GSK to examine a graph, the node or edge with focus is designated as the current or selected node/edge. Whenever focus is changed to a different node or edge, its name is announced by the screen reader in use by the blind user. The currently selected node or edge is also highlighted thus providing a visual focus indicator, especially important for sighted users navigating via the keyboard.

In order for blind users to create and share graphs with sighted people, they must be able to position the nodes in such a way that they are visually viewable. GSK facilitates this by providing blind users (and other users) with the means to specify the *spatial layout* of a graph.

3.2 The GSK Interface

The GSK interface shown in Figures 2 and 3 above provides two different views of the same graph, *Connection View* and *Grid View*. Connection View displays a graph as a conventional node-link diagram. Grid View allows blind (and other) users to spatially lay out the nodes of a graph, but does not display its edges. Using the View Menu, the user may easily switch between them depending on which is more appropriate for the task at hand.

In addition to the large central panel where the graph is displayed, the interface contains a menu bar with File, Edit, Graph, View, and Help menus as well as a tool bar with Open, Save, Remove, and Add Edge buttons. A status bar at the bottom provides information about the currently selected node/edge or, in the absence of a graph, the name of the view. When a screen reader is in use, the status is announced whenever it changes thus allowing a blind user to track focus changes. Appropriate mnemonics and keyboard shortcuts are also provided thus allowing for multiple means of accessing each function of the interface. Graphs may be saved in .gsk format and/or exported as PNG images.

3.2.1 Connection View

Connection View provides the typical point-and-click interface and node-link diagram representation of a graph to which sighted users are accustomed. It is also useful for blind users in that it allows them to navigate a graph via the keyboard. When working together, changes to a graph that are immediately visible to a sighted user are announced to a blind user via the screen reader.

Creating, Editing, and Removing Nodes

In Connection View, a new node is created by double-clicking on an empty space on the graph panel. The Node Properties dialog box shown in Figure 4 is then displayed with default values for the node name, X and Y Locations in Grid View, shape, height, width, and color, any of which may be changed, if desired. Currently available node shapes include general *oval* and *rectangle* shapes as well as shapes pertaining to automata theory – *accepting state*, *accepting/start state*, *start state*, and *state*. We plan to add shapes for other domains such as flow charts, UML

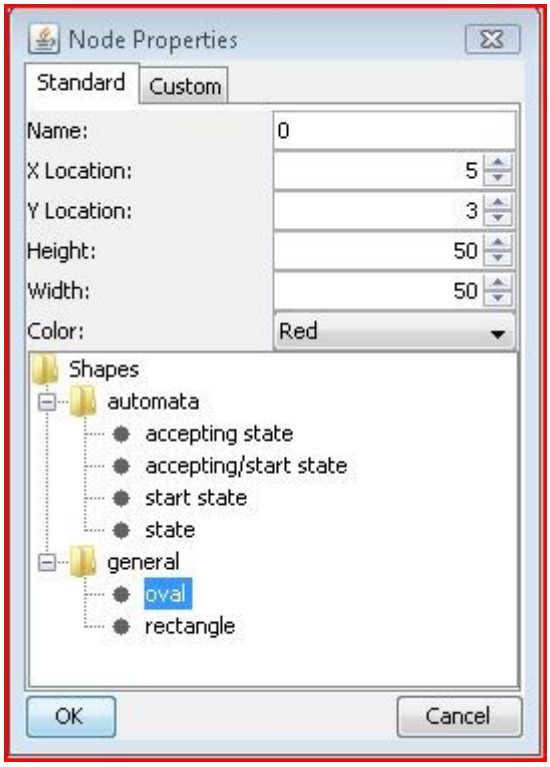

Figure 4. Node Properties Dialog Box

diagrams, etc. and have designed GSK to make this extension relatively straightforward.

The properties of an existing node may be edited by double-clicking on the node or by selecting the node and pressing Enter, both of which bring up the Node Properties dialog box. A node may be moved by changing its X and/or Y position or by dragging it to a different position. A node may be removed by selecting it and pressing the Delete key, selecting Remove from the Graph menu, or pushing the Remove button on the toolbar. Whenever a node is removed, its edges are removed as well.

Adding, Editing, and Removing Edges

An edge may be added to the selected node by pushing the Add Edge button on the toolbar, selecting Add Edge... from the Graph menu, or entering CTRL+E. The Edge Properties dialog box shown in Figure 5 is then displayed with default values for the edge name, endpoints, color, and direction. The direction choices are Undirected, Incoming, Outgoing, and Bidirected.

The properties of an existing edge may be edited by double-clicking on the edge or by selecting the edge and pressing Enter, both of which bring up the Edge Properties dialog box. An edge may be removed by selecting it and pressing the Delete key, selecting Remove from the Graph menu, or pushing the Remove button on the toolbar.

Graph Navigation

A node is selected by clicking on it, selecting Jump to Node... from the Graph menu, or by entering CTRL+J. The latter two options bring up the Jump to Node dialog shown in Figure 6 which allows the user to choose a node from a list. The left/right arrow keys are used to move between and select the edge(s) of the

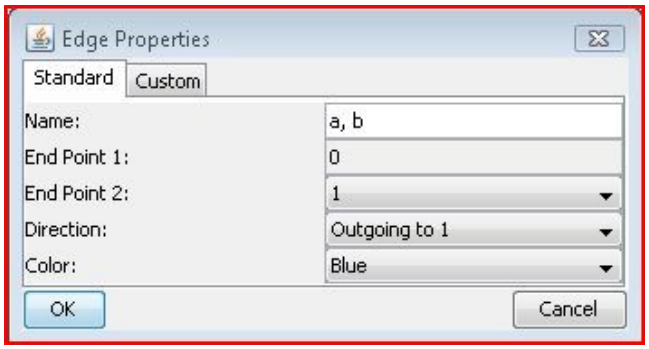

Figure 5. Edge Properties Dialog Box

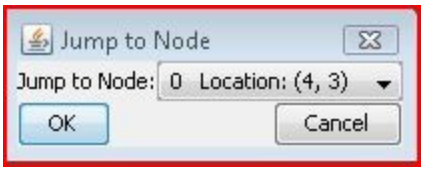

Figure 6. Jump to Node Dialog Box

node. Pressing up arrow when an edge is selected, moves focus to (selects) the opposite endpoint; pressing Escape returns focus back to the original endpoint. Each time focus changes to a node or edge, information about the selected node/edge is displayed in the status bar and announced by the screen reader, if one is in use.

3.2.2 Grid View

Grid View provides blind (and other) users with the ability to spatially lay out a graph. In this view, the graph panel is divided into a grid of squares, each one of which corresponds to a potential or actual node position in Connection View. The squares are referenced by their X, Y Location in the grid with the X values starting at 0 and running horizontally across the grid and the Y values starting at 0 and running vertically down the grid. The grid squares are much smaller than the default size of nodes in Connection View and are intended to show the location of nodes relative to each other. Figures 2 and 3 provide a side by side comparison of the same graph displayed in both views.

A grid square may be selected by clicking on it or navigating to it with the up/down/left/right arrow keys. If the square contains a node, it may also be selected using the Jump to Node action. The X, Y Location for the selected square is displayed in the status bar. If the square contains a node, the node shape and name is displayed as well. For example, in a graph used for automata theory, information about a start state named q0 with 6, 5 as its X, Y Location would be displayed as "6, 5 start state: q0." Whenever focus changes to a new square, its status bar information is announced by the screen reader, if one is in use.

Creating, Editing, and Removing Nodes

In Grid View, a new node is created by double-clicking on an empty square or by pressing Enter when focus is on an empty square. This brings up the Node Properties dialog box as previously described which allows for changing the default values of the node, such as name, X and Y Locations, shape, and color.

The properties of an existing node may be edited by double-clicking on the square containing the node or by pressing Enter when focus is on the node's square, both of which bring up the Node Properties dialog box. A node may be moved by changing

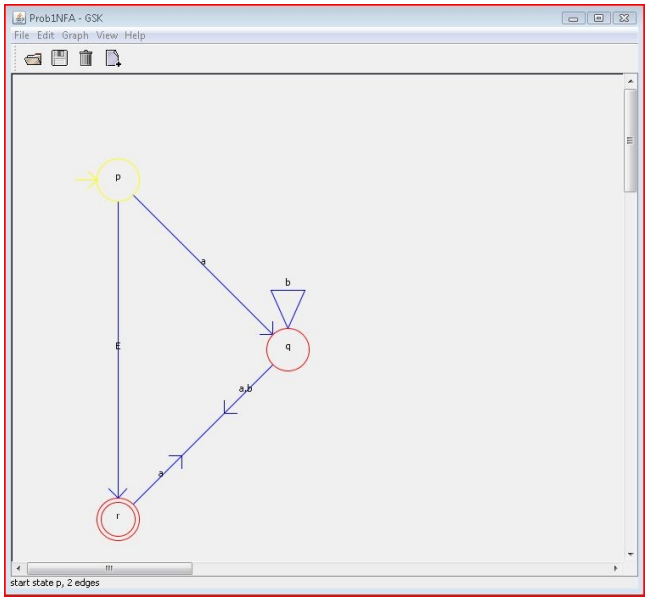

Figure 7. Nondeterministic Finite Automaton (NFA)

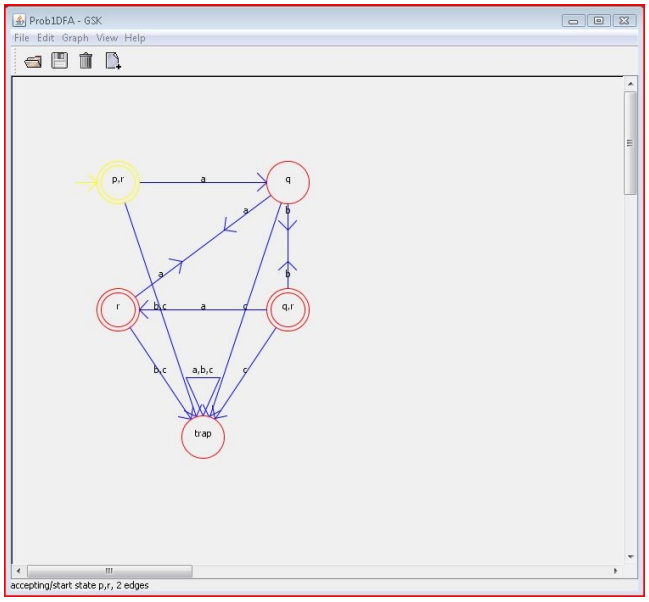

Figure 8. Deterministic Finite Automaton (DFA)

its X and/or Y Location via the Node Properties dialog or by cutting and pasting it in a different square using the standard CTRL+X, CTRL+V sequence. A node may be removed by selecting the square containing the node and pressing the Delete key, selecting Remove from the Graph menu, or pushing the Remove button on the toolbar. Whenever a node is removed, all of its edges are removed as well.

Adding, Editing, and Removing Edges

When in Grid View, an edge may be added to the selected node in the same way as previously described in Section 3.2.1. However, editing and removing existing edges requires the use of Connection View.

4. GSK IN PRACTICE

The second author, who is a blind third year Computer Science undergraduate student, successfully used GSK for assignments, tests, and quizzes in his Fall 2011 Automata Theory and Operating Systems courses. In addition to making it possible for him to create graphs to include in his course work, GSK provided him with an important problem-solving tool. It also functioned as a means for cross communication between him and the teaching staff in terms of sharing graphs. The student's use of GSK in both courses is described below.

4.1 Automata Theory Course

Graphs were used extensively throughout the Automata Theory course during the Graph Theory review, as part of NP-completeness proofs, and to represent automata – Deterministic and Nondeterministic Finite Automata (DFAs / NFAs), Pushdown Automata (PDAs), and Turing Machines. Examples of the use of GSK by the teaching staff and the second author for each of these topics are provided below.

4.1.1 Graph Theory

GSK was used by the teaching staff to render undirected graphs in the lecture notes, quizzes, and homework assignments and communicate them to the student. He found Connection View particularly helpful when examining the graphs to answer quiz and homework questions about Eulerian and Hamiltonian paths, cliques, independent sets, etc. He felt that these tasks would have been much more difficult had he been restricted to examining the graphs in table form.

4.1.2 NP-Completeness Proofs

As part of an NP-Completeness proof, the student was required to create a graph representation of the Boolean expression $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$. He used GSK to create this graph as shown in Figure 2, which he then exported as a PNG image and included in his LaTeX homework document. Using Grid View as shown in Figure 3 to lay out the graph was very helpful.

4.1.3 Automata

Besides using GSK to render the various types of automata, the student used it to convert an NFA to an equivalent DFA as shown in Figures 7 and 8. To determine whether a DFA was equivalent to a given regular expression, he generated strings using the regular expression and then used Connection View to check if the strings were accepted by the DFA. Again, he felt that using GSK was much more efficient than having to rely on transition table representations of the automata and that it provided him with computationally equivalent graph representations to those available to sighted students.

4.2 Operating Systems Course

The Operating Systems course only made use of one type of graph – a system resource-allocation graph. The student used GSK to create modified resource allocation graphs to help him detect deadlock and other resource contention problems and determine how they might be resolved. For a quiz, he was given a textual description of the resource-allocation graph in Figure 9. He was able to render it using GSK as shown in Figure 10 and solve the related problems using only 5 minutes more than his sighted peers, who were provided with the node-link diagram version of the graph.

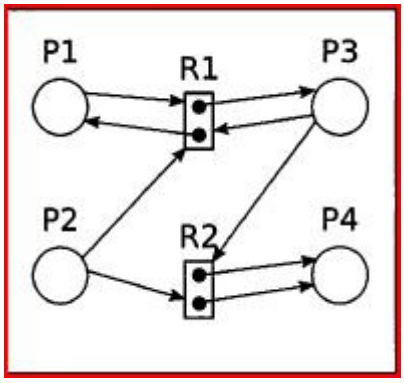

Figure 9. Resource-allocation Graph

Image courtesy Dr. David Sturgill

5. CONCLUSION AND FUTURE WORK

We have demonstrated the usefulness of GSK for a blind student in two CS courses – Automata Theory and Operating Systems; the student regrets not having had such a tool for his prior Discrete Mathematics and Data Structures courses for which it would have been invaluable. His sighted Automata Theory teaching assistant found creating graphs with GSK to be quite simple and straightforward, and expressed interest in using it to create homework solution sets. Thus, GSK appears to have universal appeal.

We plan to carry out controlled user studies with more blind participants to help determine whether the interface and paradigm are intuitive, as well as how close GSK comes to providing blind and sighted people with computationally equivalent access to graphs. We will continue to improve GSK, incorporating feedback from the user studies as well as our own ideas, in hopes of providing blind students and professionals with greater access to graphs in a universally accessible and appealing way. Our ultimate goal is to provide a graph sketching tool that is beneficial for all users regardless of their means of computer interaction.

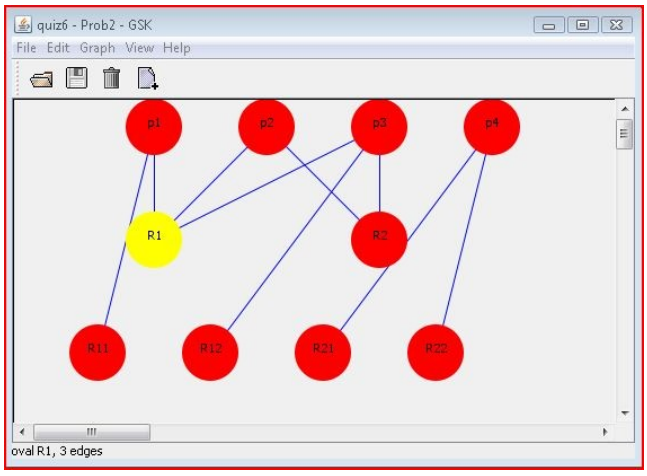

Figure 10. Resource-allocation Graph

6. ACKNOWLEDGMENTS

We would like to thank Dr. Richard Ladner for the AccessComputing mini-grant that supported this work via NSF Award #CNS-0837508.

7. REFERENCES

[1] Blenkhorn, P. and Evans, D. G. 1998. Using speech and touch to enable blind people to access schematic diagrams. *J. Netw. Comput. Appl.* 21, 1 (January 1998), 17-29. DOI=10.1006/jnca.1998.0060 http://dx.doi.org/10.1006/jnca.1998.0060

[2] Brown, A., Pettifer, S. and Stevens, R. 2003. Evaluation of a non-visual molecule browser. In *Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility* (Assets '04). ACM, New York, NY, USA, 40-47. DOI=10.1145/1028630.1028639 http://doi.acm.org/10.1145/1028630.1028639

[3] Burgstahler, S., and Cory, R. (2008). *Universal design in higher education: From principles to practice*. Cambridge, Mass: Harvard Education Press.

[4] Calder, M., Cohen, R. F., Lanzoni, J. and Xu, Y. 2006. PLUMB: an interface for users who are blind to display, create, and modify graphs. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility* (Assets '06). ACM, New York, NY, USA, 263-264. DOI=10.1145/1168987.1169046 http://doi.acm.org/10.1145/1168987.1169046

[5] Crescenzi, P., Rossi, L. and Apollaro, G. 2012. Making Turing machines accessible to blind students. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (SIGCSE '12). ACM, New York, NY, USA, 167-172. DOI=10.1145/2157136.2157190 http://doi.acm.org/10.1145/2157136.2157190

[6] Kennel, A. R. 1996. Audiograf: a diagram-reader for the blind. In *Proceedings of the second annual ACM conference on Assistive technologies* (Assets '96). ACM, New York, NY, USA, 51-56. DOI=10.1145/228347.228357 http://doi.acm.org/10.1145/228347.228357

[7] King, A., Blenkhorn, P., Crombie, D., Dijkstra, S., Evans, D.G., and Wood, J. Presenting UML Software Engineering Diagrams to Blind People. In *Proceedings of ICCHP*. 2004, 522-529.

[8] Larkin, J. H. and Simon, H. A. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science* 11, 1 (January - March 1987), 65-100.

[9] Miller, D. 2009. *Can we work together?* Ph.D. Thesis, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina. Retrieved November 9, 2012 from http://search.lib.unc.edu/search?R=UNCb5970444

[10] Sec. 508 Electronic and Information Technology Accessibility Standards. Retrieved November 9, 2012 from http://www.access-board.gov/sec508/standards.htm

[11] Stallmann, M. F., Balik, S. P., Rodman, R. D., Bahram, S., Grace, M. C. and High, S. D. 2007. ProofChecker: an accessible environment for automata theory correctness proofs. *SIGCSE Bull.* 39, 3 (June 2007), 48-52. DOI=10.1145/1269900.1268801 http://doi.acm.org/10.1145/1269900.1268801