A Natural Interaction Reasoning System for Electronic Circuit Analysis in an Educational Setting

by

Chang She

B.S. Electrical Engineering and Computer Science
B.S. Political Science

Massachusetts Institute of Technology, 2005

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2006

Signature of Author: ______
ı Computeı Science
May 26, 2006

Certified by: ___
Randall Davis
Science and Engineering
pervisor

Accepted by: ______
. Smith
Chairman, Department Committee on Graduate Theses

Natural Interaction Reasoning System for Electronic Circuit Analysis in an Educational Setting

By

Chang She

Submitted to the Department of Electrical Engineering and Computer Science
On May 26, 2006
in partial fulfillment of the requirements for the
Degree of Master of Engineering in
Electrical Engineering and Computer Science

Abstract

This thesis presents a sketch-based interaction system that can be used to illustrate the process of reasoning about an electrical circuit in an educational setting. Recognition of hand-drawn shapes is accomplished in a two stage process where strokes are first processed into primitives like lines or ellipses, then combined into the appropriate circuit device symbols using a shape description language called LADDER. The circuit is then solved by a constraint-propagation reasoning component. The solution is shown to the user along with the justifications that support each deduction. The level of detail and the speed of the solution playback can be customized to tailor to a student's particular learning pace. A small user study was conducted to test the performance of the recognition component, which revealed several recognition problems common to almost all of the users' experiences with the system. Suggestions for dealing with these problems are also presented

Thesis Supervisor: Randall Davis

Title: Professor of Computer Science and Engineering,
Director of Research, MIT CSAIL

Acknowledgements

First and foremost I am deeply beholden to my advisor Randy Davis for being such a wonderful mentor. This thesis would never have been possible without all of his guidance and support. I would also like to thank Professor Gerald Sussman and Chris Hanson for helping me understand and use the circuit reasoning system they developed.

I am very grateful also for the amazing colleagues in my research group. In particular, I would like to thank Tracy Hammond for all of her assistance on the LADDER framework and Tom Ouyang for many enlightening discussions as well as help with handwriting recognition.

Finally, I am forever indebted to my parents Youtian and Aiqin for their understanding, encouragement, and endless patience when I needed them the most.

Table of Contents

1. Introduction

This thesis presents a sketch-based interaction system that can be used to illustrate the process of reasoning about an electrical circuit. The illustration of complex processes has always been a very important task in education. In physics, illustration of the interaction between objects helps to show the underlying physical forces at work. In genetics, diagrams are used to illustrate the sequence of gene expression and protein activity in complicated genetic pathways. And in electrical engineering, the circuit diagram is the most important tool both in illustrating the underlying electromagnetic forces at work and to show the many layers of abstraction that make analysis of complex devices possible at all.

Unfortunately, in almost all instances these illustrations are static and their explanatory power may be limited without accompanying verbal explanations. Although it is possible to animate diagrams with tools such as PowerPoint or software specialized for the particular domain of interest, these tools often come with several disadvantages. Presentation software with animation tools like PowerPoint typically require each example to be animated separately. CAD-like tools such as SPICE (for electronic circuits) have difficult-to-use interfaces with either drag-and-drop methods or even textual descriptions. Moreover, tools like SPICE or Matlab uses matrix methods to solve circuits – a process that is very non-intuitive and often non-reproducible by humans.

This thesis presents a system designed to overcome these deficiencies in an educational setting for the circuit domain. A more natural user interface recognizes hand-drawn sketches of circuit devices. Recognition occurs online so that the user receives feedback while drawing. After the circuit has been recognized, a limited constraint-propagation reasoning system is used to solve the circuit. Constraint-propagation style reasoning is much more comprehensible than matrix methods because fewer variables are used at each step of reasoning and because the justification for each computed value can be made available. At the time of writing, the reasoning system is limited to parts of the circuit that can be solved without resorting to a system of simultaneous equations. Finally, the deduced values along with their justifications are played back to the user. In this manner, useful examples can be illustrated and explained in an unsupervised setting.

Take the following interaction for example: Figure 1.1 shows a hand-drawn circuit diagram recognized by the system consisting of two voltage sources, three resistors, and a transistor. The voltage source on the left has a DC voltage of 15 volts while the voltage source on the right has a DC voltage of 5 volts. The three resistors are also labeled with their respective resistance values. The parameters for the transistor have not been specified by the user (as a reminder to the reader, we have added to the figure labels for the 3 terminals of the transistor – base, collector, and emitter).

Once the user presses "start", the circuit is given to the reasoning system and the deduction process begins. First, the ground is inferred to have a potential of 0 (by definition) and the plus terminals of the two voltage sources are shown to have values of 15 volts and 5 volts, as specified by the voltage source parameters. The system also displays the default parameters for a transistor (beta = 100 and IO = -10^{12} amperes).

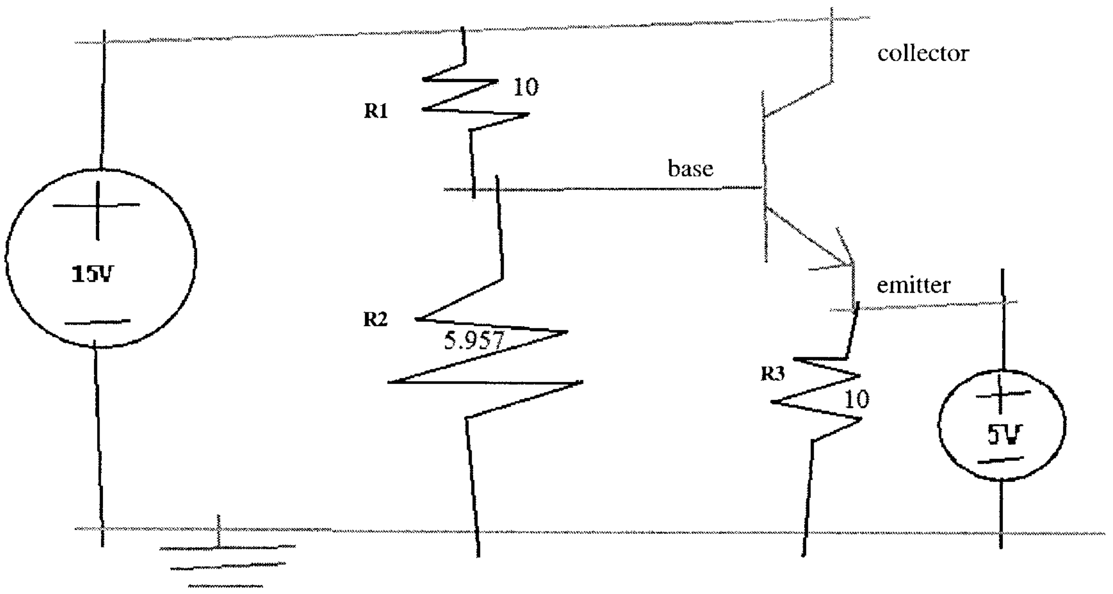


Figure 1.1. A hand-drawn circuit diagram consisting of two voltage sources, three resistors, and a transistor. The voltage source on the left has a DC voltage of 15 volts while the voltage source on the right has a DC voltage of 5 volts. The resistors R1, R2, and R3 are labeled with their respective resistance values. The names of the 3 terminals for the transistor have been added as a reminder to the reader, but the transistor's parameters have not been supplied by the user.

Now the system analyzes the transistor. Because the values of the parameters have not been specified by the user in this example, the system operates on a set of default assumptions that is common when first analyzing a transistor. It assumes that there is no base current. It also assumes that the transistor operates in the amplifying region, meaning that there is a positive current flowing from the collector down to the emitter. Lastly, it assumes that the base-emitter voltage is 0.6 volts. Based on these assumptions, the system can now deduce the collector current (0.5 A) and the potential of the base (5.6 V).

Next, the system turns its attention to the resistor R3. The resistance parameter specified in the beginning, combined with the deduced voltage difference across R3, enables the system to apply Ohm's law in order to compute the current flowing through R3. Once the value for a variable (in this case the current through R3) has been deduced, the system shows the justification for the deduction, followed by the deduced value. First, the name of the law ("Ohm") used in the deduction is displayed. Then the body of the Law ($V = I * R$) is shown in prefix notation. Next, the system fills in the values for all other variables in the equation (voltage = 5 V and resistance = 10 Ohms). Finally, the system shows that the current flowing down through R3 is 0.5 amperes (Figure 1.2). From there, Ohm's law is applied once again to obtain the current flowing through resistor R1. Finally, because the system also knows the resistance and the voltage across R2, it applies Ohm's law one last time to deduce the current flowing through R2. At this point, the system stops because all node potentials and branch currents in the circuit have been determined.

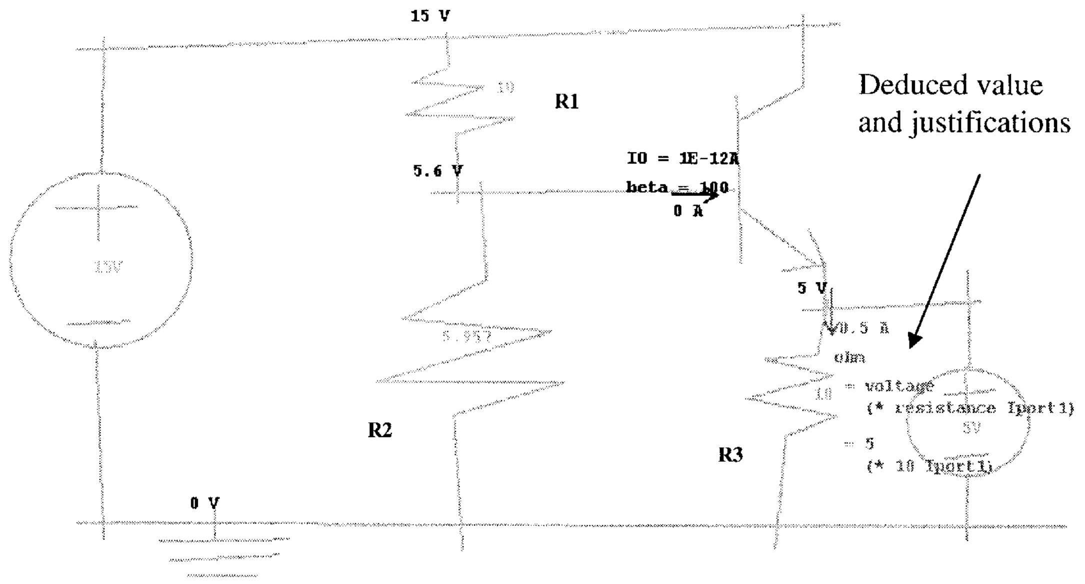


Figure 1.2. Knowing the voltage difference and the resistance of R3 enables the system to deduce the value of the current flowing through it. The system displays the name of the relation (ohm) first. Then, the body of the relation (V = I*R) is shown. Next the system fills in the values of the other variables (5 = 10 * Iport1). Finally, the system displays the deduced value, 0.5 amperes.

The system can also be used to guide students through circuit synthesis problems. In synthesis problems, students are often asked to determine the appropriate device parameters given a set of assumptions about the circuit. Suppose we are given the same circuit as shown in Figure 1.1, except this time the resistance for R2 was not given. We are asked to determine what the resistance value must be in order for the default transistor assumptions to hold true.

The analysis starts in exactly the same manner as the previous example. However, once the system finishes analyzing all other parts of the circuit, it cannot use Ohm's law to deduce the current flowing through R2 because the resistance value was not given. Instead, the system uses the assumption about the transistor to help with the reasoning process. Because the transistor is assumed to have no current flowing through its base terminal, the current flowing through R1 must equal the current flowing through R2. The system then shows that 0.94 amperes is flowing through R2. Finally, the current and voltage across R2 enables the system to apply Ohm's law to determine the missing resistance value that would be consistent with the transistor assumptions (Figure 1.3).

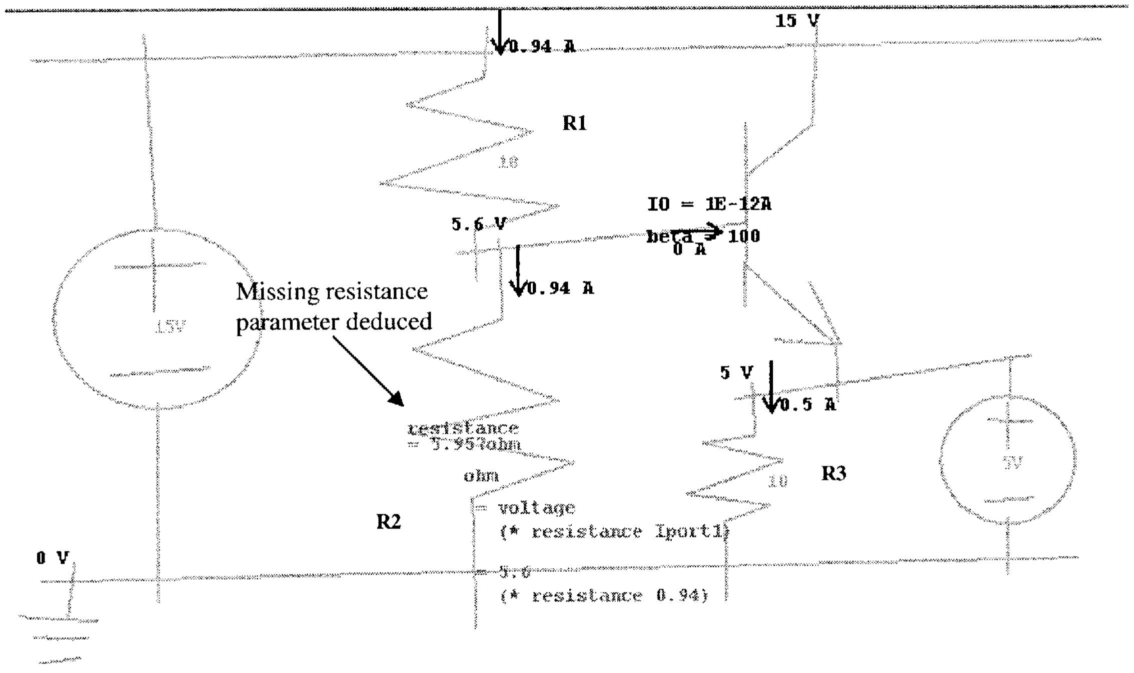


Figure 1.3. When a parameter is missing but can be deduced, the system will also show the value. In this example, while the resistance for R2 was not given, there is only one value that it can have given the assumptions that the system has made about the transistor (no base current, positive current from the collector to the emitter, and 0.6 volts across the base to the emitter).

It may surprise the student in an introductory circuit theory course that many expert deductions can be made without resorting to setting up any systems of simultaneous equations. However, in the above example, we saw that by using a simple set of assumptions about circuit devices, the system was able to use single step deductions to solve seemingly complex circuits. Moreover, showing the appropriate justifications for each deduction helps the user to understand how the deduction was made. This may help students gain a firmer grasp of circuit concepts such as Ohm's law or the implications of operating assumptions about a device. This type of learning would not be possible if the circuits were solved using matrix methods. Even though KVL, KCL, and device laws are still used to create the matrix equations, traditional matrix solution techniques such as LU decomposition or QR factorization do not preserve the configuration of the matrix. As a result, it is unclear how one would create and maintain a mapping between rows or columns of entries in the solution matrix and a particular device or device law.

2. Methods

The system presented in this thesis is an integration of a sketch recognition component with a constraint-propagation style circuit reasoning component. The input is given by the user in the form of hand-drawn circuit diagrams using devices like a Tablet PC. The strokes on the drawing surface are then broken down and classified as primitive shapes like lines, ellipses, and polygons. Groups of primitive shapes that satisfy the appropriate geometric constraints are then combined to form the corresponding circuit device symbols. The recognition process occurs in tandem to user input so that the recognition feedback for the user is almost immediate.

Once the circuit is complete, the system translates the graphical representation of the circuit into the representation required by the reasoning system. The circuit is then fed to the reasoning component, deriving values for the unsolved node potentials, branch voltages, and device parameters. Lastly, the derived values are shown to the user along with the justifications for the deductions (Figure 2.1.).

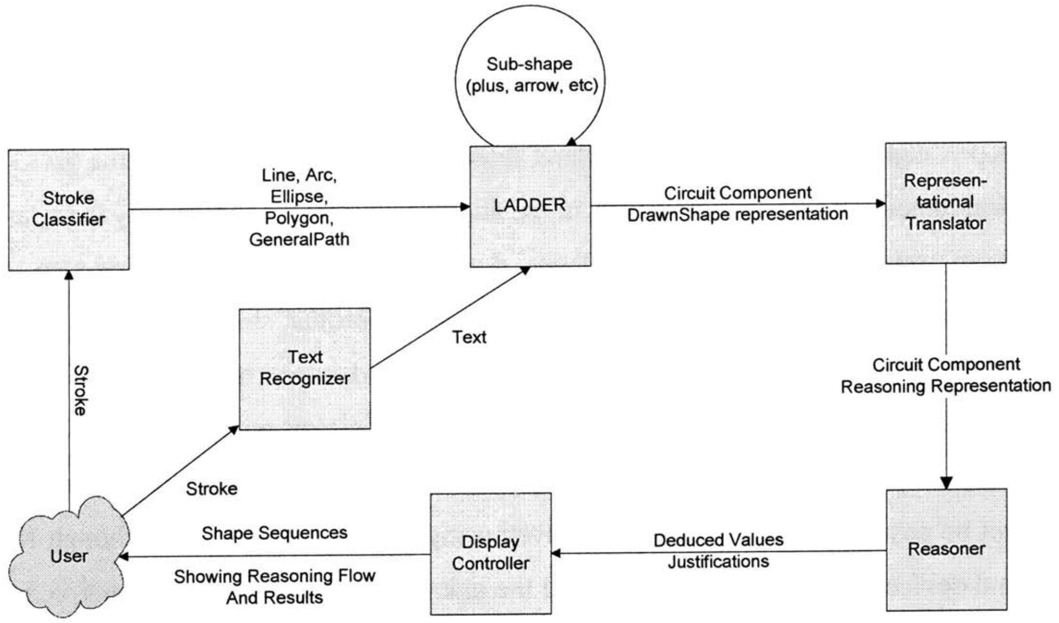

Sub-shape (plus, arrow, etc)
Stroke Classifier
Line, Arc, Ellipse, Polygon, GeneralPath
LADDER
Circuit Component DrawnShape representation
Represen-tational Translator
Text
Text Recognizer
Stroke
Circuit Component Reasoning Representation
Stroke
Shape Sequences
User
Showing Reasoning Flow And Results
Display Controller
Deduced Values Justifications
Reasoner

Figure 2.1. Input is given by the user in the form of raw strokes, possibly drawn using the stylus on a Tablet PC. The strokes are recognized and combined into the appropriate circuit devices. Text input is handled separately from the stroke classifier. Then the system translates the circuit into the circuit language used by the reasoning component. The reasoning component deduces values for the unsolved parts of the circuit and the system displays those values along with the appropriate justifications to the user.

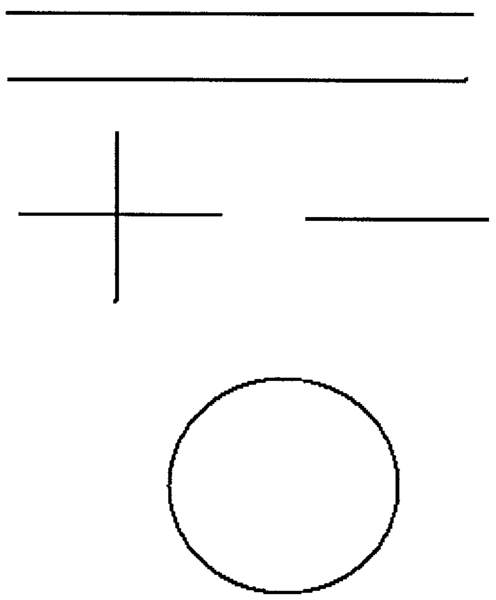
2.1. Recognition of Circuit Devices

Recognition begins as soon as the user starts drawing the circuit. The basic unit for recognition is a stroke, defined as everything drawn by the user between pen-down and pen-up. The process that takes raw strokes to recognized circuit devices is a two stage approach in which strokes are first classified as primitive shapes, and the primitive shapes are then hierarchically combined into increasingly complex shapes.

The first stage of recognition is classification of strokes into primitive shapes. The stroke classifier is based on Sezgin and Davis (2003, 2001). Raw strokes given by the user are first given to the corner detector. Sezgin's corner detection algorithm is an improvement over many existing algorithms because it combines results from stroke speed and curvature data to detect the corners in a stroke. Once corner detection is complete, a stroke fit is generated for the primitive shapes of line, ellipse, arc, polygon, and general path. A polygon is a combination of lines while a general path is a combination of lines, arcs, and polygons. The fit with the minimum least squared error is then selected as the final classification for the stroke.

In addition to the graphical portions of a circuit, device parameters also need to be specified. Parameters are specified by the user via text labels. Text input is not directly sketched on the drawing surface and is not processed by the stroke classifier. Instead, text can be given by the user either by typing or by using the text recognizer supplied in the Microsoft Tablet PC Recognizer Pack.

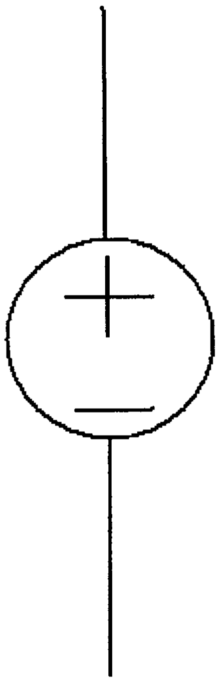
Once the primitive shapes have been generated, the recognition processes advances to the second stage where groups of simpler shapes that satisfy appropriate constraints are combined into more complex ones. This stage of the recognition process occurs under the LADDER shape description framework designed by Hammond and Davis (2004, 2001). LADDER is a shape description language that allows shapes to be recognized by describing the components of a shape and the relationship between the components. The LADDER language also allows recognized shapes to be components in more complex shapes. For example, a voltage source is described as a combination of lines, a plus sign, and an ellipse (Figure 2.1.1). A plus sign, because it is not a primitive shape, must also be described, in this case as a horizontal line intersecting a vertical line at its center.



```
(define voltage-source
  ...
  (components
    (Line top)
    (Line bottom)
    (Plus pl)
    (Line minus)
    (Ellipse circle)
  ...
)
```

Figure 2.1.1. In order to recognize a shape using LADDER, the components of the shape must be specified. In this example, a voltage source is made of 3 lines, a plus sign, and a circle.

A shape description cannot stop at listing its components, or otherwise the lines and the ellipse shown in Figure 2.1.1 would be considered a voltage source. We need to specify the relationship between the component shapes. For example, the plus and minus signs must be inside the ellipse, while the lines representing the terminal wires of the voltage source should intersect the circle at one end, be parallel to each other, and be perpendicular to the minus sign (Figure 2.1.2.). Note that constraints are transitive. By specifying that the lines "top" and "bottom" should be parallel and that "top" should be perpendicular to the minus sign, the constraint that "bottom" should also be perpendicular to the minus sign no longer needs to be explicitly stated. Specifying redundant constraints does not affect the correctness of the system, but does slow down the recognition process.



```
(define voltage
  ...
  (components
   (Line top)
   (Line bottom)
   (Plus pl)
   (Line minus)
   (Ellipse circ))
  (constraints
   (contains circ pl)
   (contains circ minus)
   (perpendicular minus top)
   (parallel top bottom)
   (intersects circ top.p1)
   (intersects circ bottom.p1)
   (oppositeSide minus bottom.p1 plus.center)
   (oppositeSide minus top.p1 bottom.p1))
```

Figure 2.1.2. Appropriate constraints must be specified in addition to the list of components

Constraints can also be added for purposes of convenience. In Figure 2.1.2., the second-to-last constraint specified is (oppositeSide minus bottom.p1 plus.center). This constraint means that the endpoint "p1" of the line called "bottom" and the center point of the plus sign should be on opposite sides of the minus sign. This constraint does not actually help the system recognize a voltage source. The voltage source would look exactly the same if we were to switch the lines "top" and "bottom". Instead, the purpose of this constraint is to help the system identify which line ("top" or "bottom") corresponds to which terminal of the voltage source. Once this is done, devices connected to the lines "top" and "bottom" can be matched with the correct terminals so that the reasoning component does not need to perform this check explicitly. Convenience constraints like the one just described are included in other directional components as well (e.g., current source).

One may be tempted to specify orientation-dependent location constraints like "above" and "below" (e.g., "top" is above "bottom"), doing so would make the system reject true positives in different orientations. Figure 2.1.3 shows examples of true positives in different orientations that would be rejected by using orientation-dependent constraints.

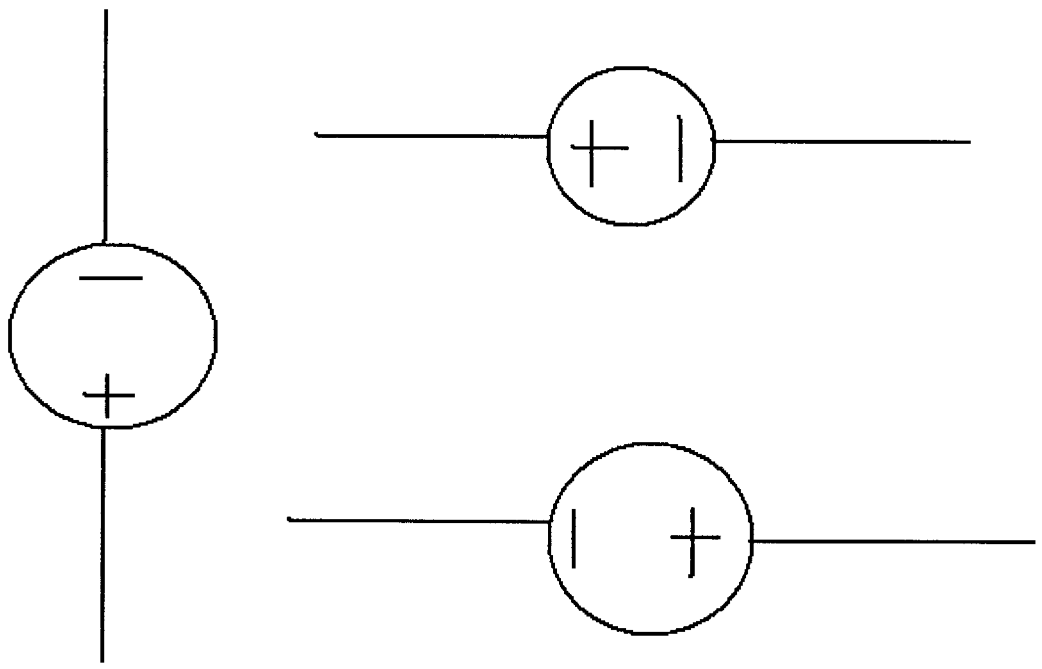


Figure 2.1.3. If orientation-dependent location constraints like "above" and "below" are used, these valid examples of voltage sources would be rejected.

Finally, the recognition of a wire is an interesting problem that must be handled by taking advantage of context. Seen in isolation, there is no way for the system to tell whether a line that has just been drawn is part of a circuit device or a wire. Knowledge of the context can help resolve this issue. In the electrical engineering domain, wires are used to connect circuit devices. Therefore, for a line to be considered a wire, it must be connected to at least two other shapes. Moreover, the shapes that the wires are connected to must be shapes that have been recognized to be complete circuit devices. For example, Figure 2.1.4.a shows a line intersecting a voltage source. The line could be part of a capacitor (Figure 2.1.4.b) or it could be a wire (Figure 2.1.4.c). Therefore, the system waits until there is additional input before deciding whether to classify the line as a wire.

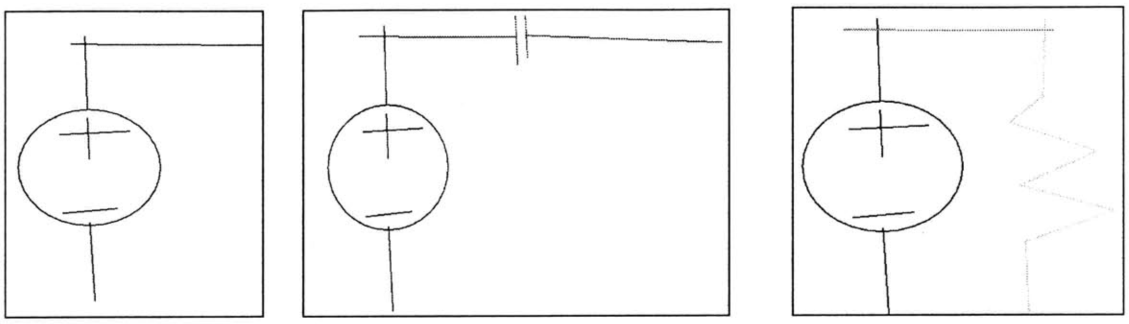

(a) (b) (c)

Figure 2.1.4. In (a) voltage source is shown to intersect a line at its positive terminal. At this point, the system does not yet no whether it is a wire. (b) shows that the user intended for a line to be part of a capacitor, and thus is not considered a wire, while (c) shows that after the user draws a resistor connected to the line, the line changes colors to show that it was recognized as a wire.

2.2. Representational Translation

Once the circuit devices are recognized correctly, we would like to analyze the circuit. However, before handing off the circuit for analysis, the system must act as a translator between the recognizer and the reasoner because the two components use very different representations for circuit devices. This difference in representation arises primarily because the two types of circuit descriptions contain fundamentally different information. The domain knowledge needed for the display and recognition of circuits is mainly geometric: to recognize circuits, the system needs to know information about how the device is drawn, what sub-shapes the device consists of, and what constraints must these sub-shapes satisfy in order for them to be classified as a complete circuit device.

By contrast, the representation used in the circuit reasoning component contains information about how the device behaves rather than what it looks like. This information includes relations between the parameters and terminals, the different models under which the circuit can be analyzed, and also the various sets of assumptions under which the circuit operates. Figure 2.2.1 shows a graphical description of a voltage source followed by a behavioral description of the voltage source. The behavioral description contains nothing about how the shape is drawn while the graphical description contains nothing about how the parameters and the terminals are related to each other.

Graphical Description:

Name: voltage source

Components:
  Line port1
  Line port2
  Plus pl
  Line minus
  Ellipse circ
  Text voltage

Constraints:
  (contains circ plus)
  (contains circ minus)
  ...

Behavioral Description:

Name: voltage source

Terminals: port1, port2

Parameters: voltage

Relations:
  KVL: Voltage = Potential(port1) - Potential(port2)

  KCL: Current(port1) = - Current(port2)

Figure 2.2.1. A comparison between a graphical description of a voltage source and a behavioral description of a voltage source. The graphical description does not include KVL and KCL relations while the behavioral description contains none of the component shapes and constraints they have to satisfy.

Perhaps the most important difference is the type of connectivity information used by the recognition versus the reasoning components. The recognition component accepts geometric connectivity, which is the way that users naturally express connections between devices. In Figure 2.1.4.c, the wire connecting the terminals of the voltage source and the resistor expresses the fact that those two terminals are connected.

On the other hand, the reasoning system uses topological connectivity. The reasoning component is most concerned with the notion that connected terminals are equipotential and that current can flow from one terminal to the other. By contrast, it does not care at all about the graphical locations of these connections.

To make the user always draw topological connections is very unnatural. Without using wires, the user would be required to draw in a way such that if two terminals were connected, then the lines that represent their terminals would have to terminate at a common point. Figure 2.2.2 shows an example of how unnatural it would be to require users to draw topological connections. With simple circuit diagrams, this requirement would be merely annoying. With complex circuits, a diagram would be extremely difficult to create.

Forcing the reasoning component to include the notion of a wire is also awkward. While it is natural for humans to think about wires when drawing or physically constructing a circuit, the notion of wires is unnecessary for the analysis of a circuit. Adding wires to the reasoning component can be achieved by adding a new device called, for example, a "wire-segment". The wire-segment will have two equipotential terminals. It would have another relation setting the sum of current flowing into its two terminals equal to zero. Based on this description, for a circuit node that contains N terminals, N-1 wire-segments need to be created between distinct pairs of terminals. In Figure 2.2.2.a, this means that 3 wire-segments are required to represent each of the 2 wires in the circuit. This description of a wire is equivalent to the notion of a circuit node. While it would make the translation process easier, it would introduce delays into the reasoning process. Because the reasoning process is occurring at the same time as the results are being shown to the user, introducing delays into the reasoning process may negatively affect the user experience.

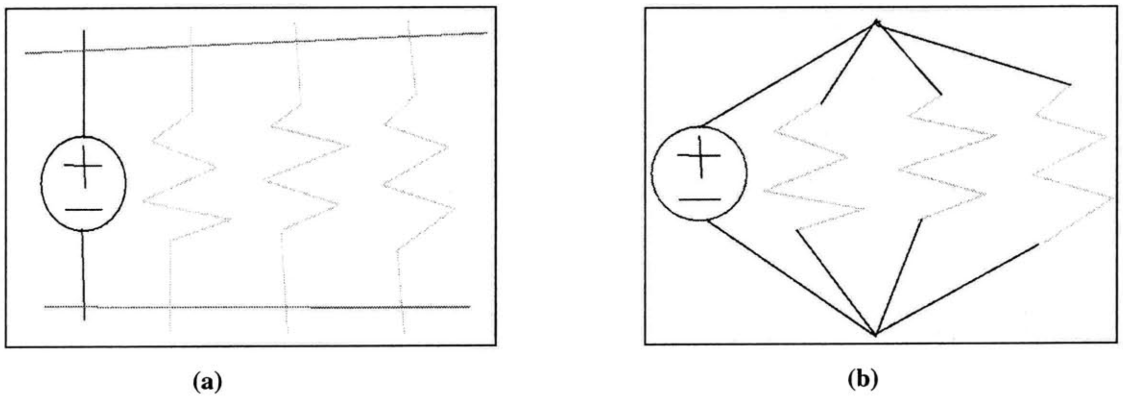


Figure 2.2.2. In (a) and (b), two equivalent parallel resistor networks are shown. (a) shows how a human naturally sketches using the concept of wires to express connectivity. (b) shows how the user would be required to draw if only topological connections were allowed.

The translation from the graphical description to the behavioral description of a circuit is accomplished in two steps. First, a mapping is created between the shapes that represent a device and their reasoning counterparts of the same name. The text components of each device are mapped to the appropriate parameters. For a voltage source, the text component named "voltage" is mapped to the parameter named "voltage". Resistances, capacitances, and inductances are handled in the same manner. A mapping between the device terminals and their graphical representations are also created. In Figure 2.2.1 for example, port1 and port2 of a voltage source corresponds to the lines of the same names respectively.

Once this mapping is created, the system proceeds to step two of the translation process, which creates the topological connections between devices. The system first creates a list of all terminals of all devices in the circuit. It picks the first terminal in the list and retrieves its graphical representation. Next, the line that corresponds to the terminal is tested for geometric intersection with the lines that represent all the other terminals. Those that are connected will be placed into one single circuit node. Then the system recursively performs this connectivity check, starting with the terminals that were connected to the initial terminal. In order to prevent loops, the terminals that have been connected previously are not checked again. This recursive process finds all terminals reachable via terminal-to-terminal or terminal-to-wire-to-terminal connections. All terminals found are topologically connected to a single circuit node and are removed from the list of terminals to check. Then, the system creates a new circuit node and repeats the same process until no terminals remain.

While creating the topological connectivity information, the graphical location of intersection between pairs of device terminals is saved. This information is used to aid the playback of deduced values and their justifications.

2.3. Constraint-Propagation Circuit Reasoning

After the circuit has been translated into the appropriate representation, it is given to the reasoning component. The circuit reasoner used in this thesis was built by Hanson and Sussman (2003) and uses constraint satisfaction to solve the circuit, instead of the matrix methods that are more commonly seen in circuit simulation systems like Matlab. As stated in Stallman (1977), the constraint propagation style of reasoning can make deductions based on limited information in a way similar to the behavior of expert circuit analyzers. This reasoning process also generates justifications for each deduction and these justifications can be examined by the user to help learn the appropriate relations and assumptions that come with each circuit device.

The reasoning component accepts input in the form of circuit device descriptions in the circuit language specified by Hanson and Sussman (2003). This circuit language description contains information such as device terminals, parameters, and relations. The relations are then used to create the constraints that must be satisfied by the various terminals and parameters in the circuit.

For example, let us examine a resistor. The resistor obeys Ohm's law, meaning that the voltage difference between its terminals is equal to the product of the resistance and the current flowing through it (commonly expressed as $V = IR$). This constraint contains three slots (V, I, and R) that can be filled with values. If all slots except one have been filled, the value for the remaining slot is computed. Whenever a slot is filled in with a numerical value, this value is then given to all other constraints that use the same slot. This is the propagation step. This process of constraint solving and propagation is repeated until no more deductions can be made.

In addition to relations like Ohm's Law and Kirchoff's Laws, the circuit language also contains the concept of groups of mutually exclusive relations. This is used by more complex devices to express notions like operating region or different parameter assumptions. For example, a transistor is said either to be beta-infinite or beta-finite. Under the beta-infinite condition, there is no current flowing through the base terminal. Under the beta-finite condition, the base current is equal to the collector current divided by a factor of 100.

One very important issue to realize is that this process of constraint propagation is not a complete inference system. In particular, it cannot solve circuits that require simultaneous equations. This greatly limits the types of circuits the system can solve. While local constraint propagation enables to system to solve complex circuits like the one presented in Figures 1.1, it does not allow the system to solve a simple voltage divider resistor network (Figure 2.3.1). In the example shown, solving one resistor would enable the system to solve the other resistor. However, because both resistors have two empty slots (current and voltage difference) in their Ohm's Law constraints, the system would need to simultaneously solve two equations (one for each resistor) in order to deduce the current flowing through the circuit and the potential between the two resistors.

A possible way to work around the lack of a complete inference system is to use the notion of equivalent devices. For example, we could define a new device called a "series-resistor-network". The device would have an indefinite number of parts, all of which are required to be resistors. The device would have two terminals and obey KVL, KCL, and Ohm's Law. Its resistance value would be equal to the sum of the resistance values of all of its resistor parts. In effect, we have combined the resistors in the network into a single equivalent resistor. If the terminal potentials and all of the resistance values are known, the Ohm's Law constraint for the series-resistor-network device only needs a single step deduction to compute the current that is flowing through it. Once the current has been calculated, the node voltages along the resistor network can be computed by the Ohm's Law constraints for each of its resistor parts. In this manner, the circuit in Figure 2.3.1 can be solved without a complete inference system. Similar new macro-devices (e.g., "parallel-resistor-network") can also be defined to handle additional circuit types. This idea of using equivalent devices is discussed in more detail in Sussman (1977).

The new devices for series and parallel resistor networks are more than just a clever hack to fill the gap for the reasoning component. In fact, students in introductory circuit theory classes are often taught to analyze resistor networks in exactly the same way. Therefore, the proposed new circuit devices would actually have significant educational value in addition to enabling the reasoner to solve a larger class of circuits. Unfortunately, these additions to the system could not be accomplished because there is currently no construct in the circuit language that allows for the definition of a device with an indefinite number of parts.

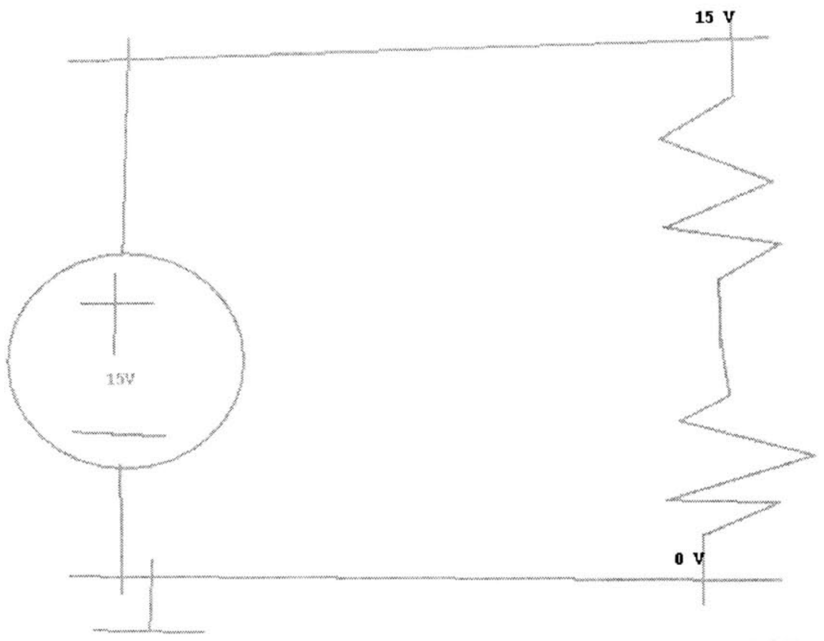


Figure 2.3.1. The reasoning component used in this thesis is not a complete inference system. It cannot solve circuits that require simultaneous equations. In this series resistor network, each of the two resistors has two empty slots (current and voltage difference). Thus, we would need to solve a system of two equations in order to deduce the current flowing through the circuit and the potential between the two resistors.

2.4. Reasoning Playback

Once the reasoning process begins, results are shown to the user after each reasoning step. For each newly deduced value, the system queries the reasoner for the relation that justifies the deduction. Figure 2.4.1 shows an example of the playback sequence. First, the name of the relation is shown to the user. Next, the body of the relation is shown. The relation is displayed in prefix notation with the mathematical operators (e.g., =, +, *) shown before the arguments. This is a consequence of the fact that the reasoning component expresses its constraints in prefix notation. Third, the system queries the reasoner for the values of the supporting variables in the relation and displays the relation again, but with all the supporting variable values filled in. Then, the system shows the deduced value. Finally, the justification disappears, leaving only the deduced value as the system proceeds to the next deduction. The system repeats this process of one-step propagation and deduction playback until no more deductions can be made.

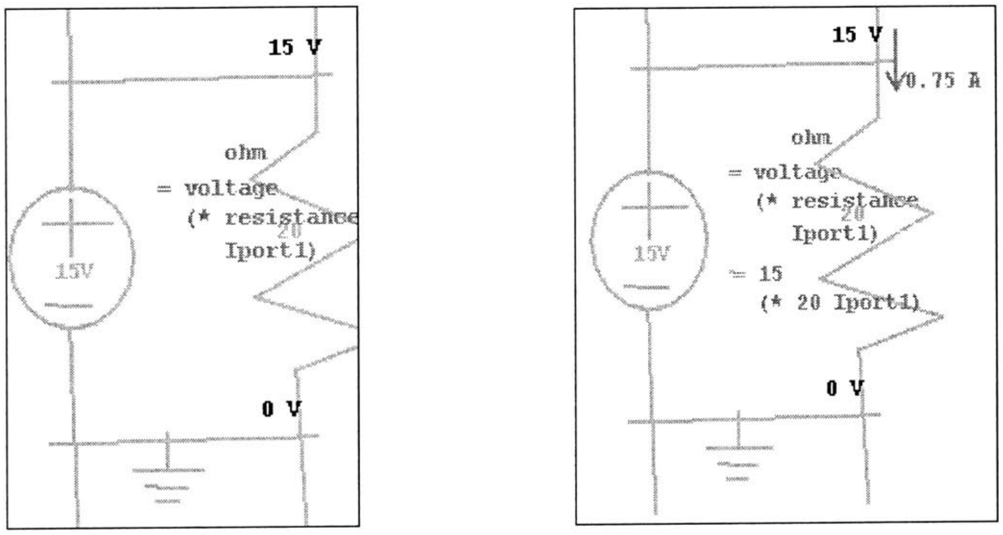


Figure 2.4.1. A partial sequence of the reasoning playback. First, the name of the law (Ohm) used in the deduction is shown. Then the body of the relation is displayed (= Voltage (* resistance Iport1)). Next, the values for all variables except for the variable being deduced are shown. Last, the deduced value is shown (the current for the resistor is shown to have a value of 0.75 amperes downwards). After the sequence for one deduction is complete, the justification disappears, leaving only the deduced value (the values 15 V and 0 V are the remains of previous deductions).

Prior to starting the playback sequence, the user can change the speed at which each step is shown and can choose whether to show the justification for each device type and relation. During the playback sequence, the user can pause as well as rewind the playback. These options allow the system to be tailored to the individual learning needs of the user.

3. Results

The performance of the recognition component was tested in a pilot study. An overview of the results is given here. A more detailed discussion of the study, including the experimental setup and analysis of the results, is contained in sections 3.1 – 3.4. The results of the study showed that the recognizer in general was not sensitive enough. In other words, many shapes that should have been valid circuit symbols were rejected by the recognizer. Recognition errors were evenly split between stroke level problems and constraint specification problems. A few false positives were generated due to under-specification of constraints for certain shapes.

There were two main stroke level problems. One problem was that, due to a low sampling rate, some strokes contained too few points and were not classified correctly. The other problem was that short line segments were often not classified correctly. Constraint specification problems mainly concerned shapes with repeated parts like a resistor or ground symbol. The variety with which users tended to draw these shapes showed that the descriptions for these shapes were over-constrained.

Several deficiencies of the LADDER framework and the stroke processor were revealed. First, shapes with repeated motifs like resistors turned out to be difficult to describe. Second, symbols like the one for an inductor presented additional challenges because its sub-structures are not easily segmented using the corner detection algorithm in the stroke classifier. And third, specifying relationships between shapes (instead of between components) was awkward.

A test of the reasoning component was not conducted. To gauge the effectiveness of the reasoning component, a larger user study should be conducted to test whether students in a circuit theory class using the system are able to gain a better understanding of the material. Experimental issues for this possible study are discussed in section 5.

3.1. Experimental Setup

Subjects for the pilot study were asked to reproduce circuit diagrams that were shown to them. The performance of the recognition component of the system was measured in relation to several key metrics. The experiment was designed to measure the frequency of false positives and false negatives, whether errors occurred mainly at the stroke level or the constraint specification level (defined later), whether the method for parameter specification was intuitive, and how much effect the input media had on recognition performance.

The study consisted of 10 users divided evenly into two groups. The first group sketched the circuit diagrams on a 40” plasma display via an interactive overlay on top of the display. The second group sketched the circuit diagrams on a Tablet PC with a 12” screen. The purpose of this separation was to understand the effect of the input device on the usability of the system’s user interface.

Each subject was asked to reproduce 15 circuit diagrams on the computer. Each diagram was drawn from sample problems used in an introductory circuit theory course. The subject was told to draw at a reasonable speed using the same degree of care as they would for an important homework assignment. This request was made as the study was not designed to robustly measure how well the system handles noisy input.

For each circuit diagram, the user was first shown the sample drawing and then asked to reproduce it on the computer. At all times, examples of how individual circuit devices should be sketched were available to the user for reference. The actual diagram they were asked to sketch was not available for reference while they were drawing. This requirement was made so that the subject would not copy the sample drawings stoke for stroke. However, if the subject forgot what the diagram looked like, he or she was allowed to stop drawing and refer to the diagram before starting again.

After the user completed a drawing, he or she was asked to specify parameters for some or all of the circuit devices. The parameters were described to the user verbally (e.g., the voltage source has a strength of 10 volts). The subject was then asked to double click on where they wished to place the parameter and write the value into the text recognition popup window. This part of the study was designed to test whether the system's method for parameter specification was intuitive. For voltage and current sources, the first text input placed within the ellipse component of the shape was considered to be the value of the voltage and current parameters, respectively. For devices like resistors, inductors, and capacitors, the parameters were required to be within a 50-pixel distance from the center of the respective components.

The user had access to an "undo" button that would erase their last stroke. The number of times that the user used the undo button was counted in order to control for the differing level of patience for each user. Feedback to the user was limited for the purposes of the study. While stroke level recognition results were shown to the user, the color changes that reflected shape recognition was not shown. This was designed to eliminate the effect of each user's ability to interpret color changes.

Out of the 15 diagrams each user was asked to draw, three from the first set of five were identical to three from the last set of five. This was designed to test whether users were able to adapt to the system over the course of the study. Without the color changes that reflect shape recognition, this part of the experiment was essentially measuring whether the user could observe errors in stroke classification and adapt accordingly (e.g., by drawing more carefully or by breaking down strokes).

Once the user completed the drawings, the data was recorded for several metrics used to gauge the performance of the system. First, the number of circuit devices correctly recognized was measured as a percentage of the total number of circuit devices in each diagram. In this metric, ground symbols were considered a circuit device but wires were not. The reason why wires were not included is because correct recognition of wires depends almost entirely upon correct recognition of the circuit devices the wire intersects. Thus, including wires in this first metric would double the effect of the intersecting devices on the recognition score. The total number of correctly recognized devices is very useful for showing how natural the sketch-based interface was overall. However, additional metrics were needed because this measure does not give information indicating the type of errors that occurred the most or suggest how to improve the system to correct these errors.

After the number of correctly recognized devices was recorded, the incorrectly recognized shapes were analyzed and placed into one of two categories. The first category consists of devices not correctly recognized because of stroke level errors. Figure 3.1.1.a shows a drawing intended to be a capacitor that was not recognized because one of its lines was classified as an arc. The second category consists of devices not correctly recognized because of a constraint level error. A constraint level error is defined as a false negative where all component primitives of the shape were correctly classified, but the constraints were not satisfied. Figure 3.1.1.b shows another intended capacitor that was not correctly recognized. While all 4 of its component lines were correctly recognized, the two halves of the capacitor were drawn too far apart to satisfy the constraints specified in the capacitor's LADDER description.

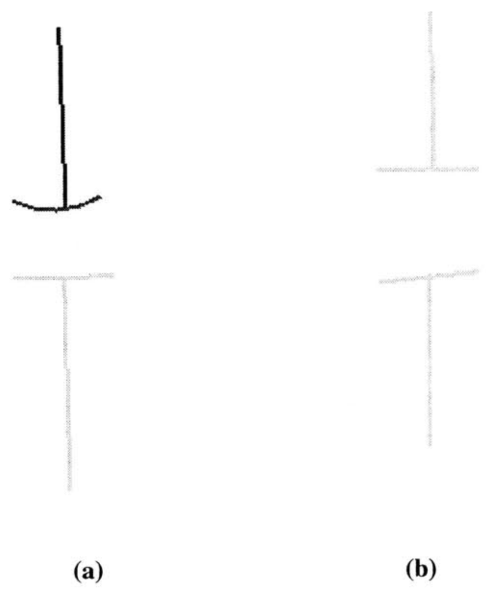

(a) (b)

Figure 3.1.1. (a) shows a drawing intended to be a capacitor that was not recognized because one of its lines was classified as an arc. The bottom half is yellow showing that the recognizer correctly recognized that half of the capacitor. However, the top half was not correctly recognized as the line and the arc are shown in their original colors. In (b), both halves of the capacitor were correctly recognized, however, they were placed too far apart to be considered a capacitor by the system. This comparison illustrates the difference between a stroke level error and a constraint level error.

This separation of error types was useful for separating the performance of the stroke classifier from that of the LADDER framework so that future work can better focus on the parts of the system that would make the most impact on performance. However, because the two parts of the recognizer work in sequence, stroke level errors may be masking constraint level problems. For example, if the arc in Figure 3.1.1.a was corrected, it is still not clear whether the two halves of the capacitor are close enough to be correctly recognized. Thus, the estimation of constraint level errors in this study is most likely an underestimate of the true measure of the constraint level errors of the system.

The measurement of constraint level errors also brought out the question of threshold values for constraints. For example, the capacitor in 3.1.1.b was not correctly recognized because the two halves were placed too far apart to meet the constraint which specified that the joints (i.e., where the two lines of each half meet) of the two halves must be "near" to each other. However, we should ask the question of how near shapes must be to be considered "near" to each other. How do we decide the threshold between what is considered near and what is not? While an optimal threshold value can always be empirically determined, it turns out that a single static threshold value may not be good enough. Instead, threshold values may be best determined dynamically using contextual information. Section 3.4 discusses this issue in greater detail.

3.2. Experimental Results and Discussion

In general, experimental results showed that the system failed to recognize many of the circuit symbols drawn by the users. Out of the 100 possible circuit devices in all 15 diagrams, an average of 47% were recognized correctly for each user. The group averages were significantly different from each other. While the Tablet PC group averaged 53%, the interactive overlay group averaged 41% (Table 3.2.1). Three common recognition problems were identified. Constraints for resistors were overly restrictive and rejected many good instances of resistors. Devices connected by a single straight line were not recognized correctly. Finally, shorter line segments were often classified as arcs by the stroke processor. These and other common problems are discussed separately in sections 3.3 and 3.4.

The slow speed of the online recognition process was a source of many complaints from the users. Because roughly half of the shapes were not recognized, many primitive shapes remained on the drawing surface and were checked for shape constraint satisfaction whenever a new stroke was drawn. For the more complex diagrams, users often had to wait 5 to 10 seconds before the recognition process was complete.

Users in the overlay group complained about the difficulty of drawing on the interactive display unit. Because there was a significant distance between the image and where the pen made physical contact with the overlay, users often could not see exactly where they were drawing. The same effect was much weaker for the Table PC simply because the pen was much closer to the image. Moreover, while users of the Tablet PC could move the mouse pointer by hovering the pen over the drawing surface, the mouse pointer on the overlay did not move until a mouse button press was registered. This meant that users in the overlay group did not really know where the stroke would start before he or she started drawing. These drawing difficulties may be the primary cause of the difference in the average recognition scores between the two user groups.

In addition to verbal feedback from the user, the number of undos used also suggested that the overlay was more difficult to draw on. While the overlay group used the undo button an average of 3.51 times per circuit diagram, the tablet group only used it an average of 0.96 times per diagram (Table 3.2.1).

Overall, there were slightly more recognition errors in the constraint category than in the stroke category. The division of errors was roughly even for the overlay group. For the tablet group, the amount of constraint errors (as a percentage of total errors) was significantly higher than the amount of stroke level errors. For the constraint level, the error rate of the overlay group was slightly higher than that of the tablet group. However, there was a much larger difference between the two groups at the stroke level (Table 3.2.1). This would seem to suggest that, even with the increased use of the undo button, users in the overlay group still found it harder to draw the correct strokes than the tablet users.

	Overlay	Tablet	Mean	Difference
Total devices drawn	100	100	100	0
Average # of Devices recognized	41	53	47	-12
Average # of Stroke level errors	30	20.3	25.15	9.7
Average # of Constraint level errors	29	25.7	27.35	3.3
Stroke error / total error	51%	44%	47.5%	N/A
Constraint error /total error	49%	56%	52.5%	N/A
Undos per diagram	3.51	0.96	2.24	2.55

Table 3.2.1. The overlay group performed substantially worse than the tablet group. The average number of devices recognized was higher in the tablet group by 12%. While the difference in constraint level errors between the two groups was small, the difference in stroke level errors between the two groups was very large. The overlay group also tended to use the undo button much more than the tablet group.

Parameter placement also revealed differences between the two user groups. Users in the overlay group were very consistent in whether they placed the parameter on top of the circuit device or on nearby blank space. This seemed to suggest that users can be divided into two groups, one that naturally preferred to place parameters on top of devices and one that preferred to place parameters next to devices. However, the tablet group showed a rather different pattern. While some tablet users consistently placed parameters on top of devices, not a single user consistently placed them next to the devices. This difference was most likely caused by the tablet's smaller screen size. When there is little space between devices, even if the user would prefer to place the parameter next to device, he or she may be forced to place it on top of the device instead.

The results for the three pairs of identical drawings were also analyzed and compared across the two groups. While the tablet PC group showed an average of 17% improvement in correctly recognized symbols, the overlay group only showed a 1% improvement. Table 3.2.2 shows the average number of additional components recognized for each of the three pairs as well as the percent improvement in recognition for each user group. One possible explanation for this difference was that users in the overlay group were frustrated with the user experience and drew less carefully for the last few diagrams.

	Pair 1	Pair 2	Pair 3	% improvement
Overlay Group	0.00	-0.40	0.60	0.01
Tablet Group	0.33	2.00	0.33	0.17

Table 3.2.2. The overlay group showed a significant decrease in the average number of symbols correctly recognized while the tablet group showed a significant increase. One possible explanation for this difference was that users were simply too frustrated with the overlay's user experience.

3.3. Stroke Level Problems

One common stroke level error was that short line segments were often classified as arcs. The root of this stroke processing problem lies in the fact that due to the sampling rate of the input interface, a short line segment may consist of only 3-5 points. If these points were not exactly collinear, what looks more like a straight line to the user may produce a smaller error for an arc fit in the stroke classifier.

One solution to this problem would be to increase the sampling rate of the drawing surface. However, if that is not possible, the problem may also be corrected by using contextual information in the shape recognizer. For example, Figure 3.3.2 shows a shape intended to be a voltage source that was not recognized because the line that was supposed to be the minus sign was classified as an arc. Conceivably, the shape recognizer could see that all other components and constraints specified in the shape definition have been satisfied and decide to look at what would happen if the arc was changed to a line. The system would see that doing so would result in a correctly recognized shape, and therefore decide to go ahead and complete the correction.

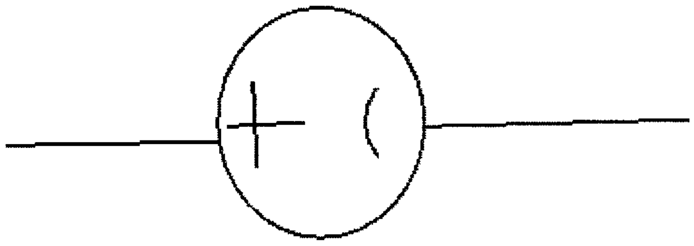
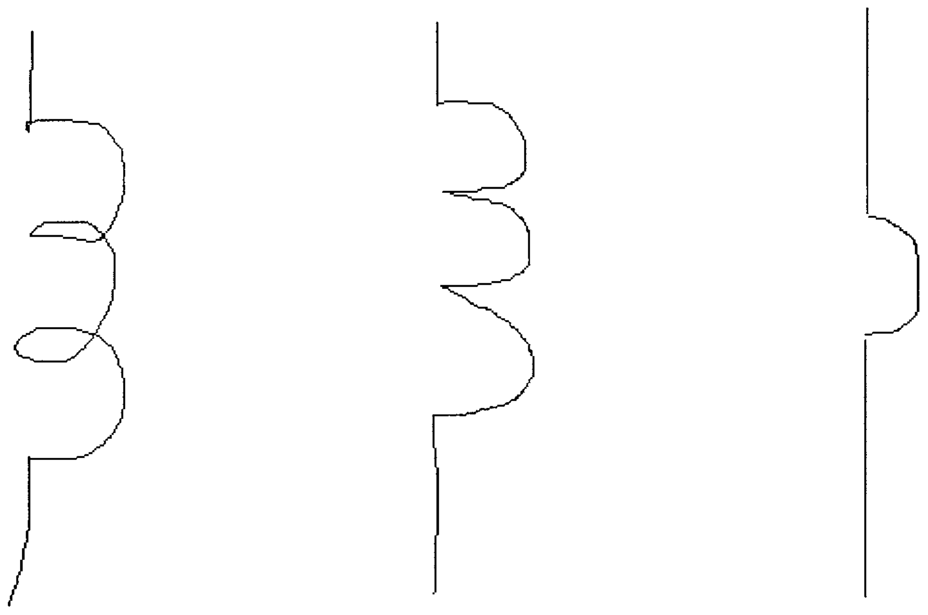


Figure 3.3.1. The intended voltage source was not correctly recognized because the minus sign designating its negative terminal was incorrectly classified as an arc. To fix this problem, the recognizer may use contextual information to decide whether to correct a stroke.

Segmenting the sub-strokes of an inductor also poses a difficult problem for the stroke classifier. Because the classifier relies on corner-detection to break down complex strokes into sub-strokes, shapes without well-defined corners in its substructure (like the inductor shown in Figure 3.3.1.a) could not be properly handled. In order to correctly recognize the coils for an inductor, a different method for stroke processing that does not depend solely on corner-detection needs to be used in order to process the corner-less spiral structures.

Another commonly accepted way to draw inductors does have well defined corners (Figure 3.3.1.b). However, if the entire inductor were to be drawn in a single stroke (as users tend to do), the stroke processor would not be able to consistently segment out all of the arcs in the inductor. While the stroke processor can easily separate out the two lines from the rest of the inductor, it would then break down the arcs into a combination of smaller lines and arcs. Thus, the user must draw each arc of the inductor with separate strokes. To minimize the inconvenience imposed by this restriction, an alternate representation of an inductor was used (Figure 3.3.1.c). The number of arcs in an inductor was reduced to one so that it took 3 strokes to draw as opposed to 5 or 6.



(a) (b) (c)

Figure 3.3.2. (a) and (b) are commonly accepted alternatives to drawing an inductor. The example shown in (a) cannot be handled by the stroke process because it does not have corners which the stroke processor use to break down complex strokes. On the other hand, (b) does have well defined corners. However, the stroke recognizer cannot break down the arcs between the corners consistently. If (b) were to be drawn in one stroke, the recognizer would most likely break down each arc into smaller arcs and lines. Therefore, the strokes in an inductor must be drawn separately. (c) is the representation of an inductor used in this thesis in order to reduce the annoyance of having to draw each stroke separately.

3.4. Constraint Level Problems

One common constraint level problem in the user study was the fact that all devices were specified with their terminal wires. For example, the inductor was defined to be a line (port1) attached to an arc attached to another line (port2). This meant that connected devices cannot be recognized unless the user used two strokes to draw the straight line between them (Figure 3.4.1). During the user study, the requirement for the user to break down what should intuitively have been contiguous straight lines made the user experience much less natural than it would otherwise have been. Even though each user was reminded to break down strokes between connected devices, there was an average of 12.535 such connections across the 15 circuit diagrams that were not broken by the user.

Eliminating this unnecessary constraint would make the user experience much smoother and may result in a significant gain in recognition performance. To eliminate this requirement, the terminal lines should be defined as contextual shapes connected to the device rather than part of the device itself.

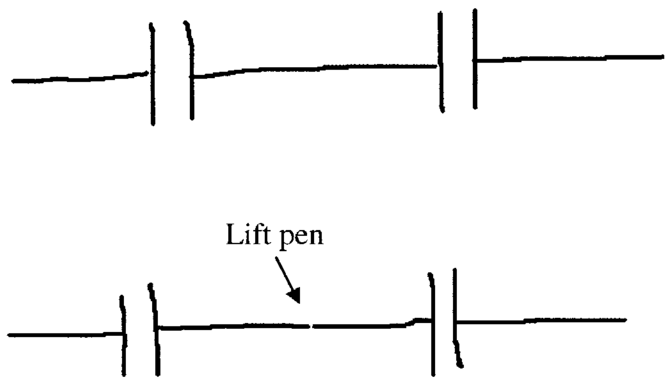


Figure 3.4.1. The two series capacitors at the top would not be recognized correctly. One would be recognized as a capacitor and the other one would be missing a line component. Only if the line between the capacitors were drawn with two separate strokes could both capacitors be recognized correctly.

Another common problem was that resistors were often not correctly recognized because the constraints that describe the resistor were too restrictive. In the system, the LADDER description for a resistor was defined to have an indefinite number of alternating parallel lines. While the indefinite number of lines made the shape description more adaptable to user drawing habits, the constraint that alternating lines must be parallel seemed to be too restrictive. Many shapes that clearly should have been considered valid resistors were not correctly recognized because some of its alternating lines were not parallel. In Figure 3.4.2 for example, a resistor was not recognized because lines A, B, and C were not parallel to each other. Instead, a resistor should have been specified so that consecutive lines had to have acute angles. This would have been a much looser constraint and most likely would have substantially reduced the number of false negatives.

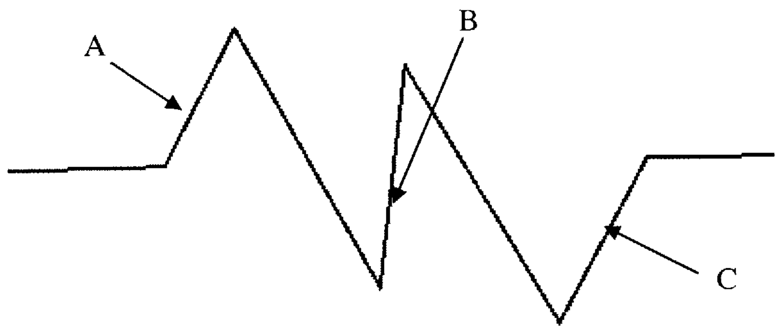


Figure 3.4.2. A shape that should have been a valid resistor was not correctly recognized because the lines A, B, and C were not considered parallel to each other.

Specifying an indefinite number of lines for a resistor was also awkward under the LADDER framework. In order to achieve this flexibility, recursive definitions for the body of the resistor (the section between its two terminal lines) had to be made. The first definition (resistor body type A) was specified to have 4 lines. A second definition (type B) was specified to have an instance of type A plus an additional line. Type B was then defined to be a sub-type of A. These two type definitions create a looping hierarchy that enables the system to accept an arbitrary number of lines as the resistor's main body.

For example, suppose we first drew 4 lines that were recognized as a resistor body type A. Next, we draw an additional line. This meets the definition for resistor body B. However, because B is a sub-type of A, the newly acquired instance for B is also considered an instance for A. Therefore, when we draw a sixth line, we once again satisfy the definition for resistor body B. While it was possible to specify the notion of an undetermined number of connected lines using this recursive technique, it is generally desirable to not have to do so via such a convoluted work-around. What is really needed is a mechanism in LADDER to describe a list of indefinitely repeated sub-shapes.

Describing the relationships between shapes (as opposed to between components of a shape) was similarly awkward. In the circuit domain, the concept of a wire must use contextual shapes to determine whether a line should be considered a wire. However, the LADDER framework does not have a natural way to specify the relationship between a wire and the contextual shapes it must intersect. A work-around exists in LADDER in the form of contextual components. A contextual component is a component of the shape that can be shared between shapes. Thus, a wire was defined to be a line that intersects two circuit devices (or wires). The problem with specifying the contextual shapes to be a (special) component of the wire is that many behaviors of a regular component are imposed onto the contextual component unless explicitly stated otherwise. For example, if one clicks on a wire to select it, the contextual devices it is connected to would also be selected. An additional check must be explicitly created so that contextual shapes are not selected along with the normal components. Other editing and display behaviors like deletion and coloring also face the same problem of an ambiguous delineation between something that is a part of a shape and something that is related but not truly a part of the shape.

Finally, the difficulty of threshold definition in determining the satisfaction of constraints among the components of a shape was made more apparent by the user study. For example, in the current implementation of LADDER, anything within 5 pixels of a point is considered to intersect that point. The problem with defining this static threshold is that it cannot take into account differences in the size of shapes drawn. This turned out to be an issue for drawing complex diagrams where each device needed to be small. Figure 3.4.3 shows two shapes that would have met the definition for half of a capacitor (defined to be two lines where line 1 is the perpendicular bisector of line 2 and an endpoint of line 1 touches line 2). In both shapes, an endpoint of line 1 is within the threshold for intersection with line 2. However, while the shape on the left could very well be part of a capacitor, it is fairly clear that the shape on the right is actually a plus sign.

This particular difficulty in LADDER was overcome by adding a new constraint to the definition of a capacitor. The new constraint was that the two lines making up half of the capacitor cannot cross at their centers. This kept the left shape in 3.4.3 valid, but made the system reject the plus sign on the right. However, LADDER should not make it necessary for its users to specify such a constraint because a constraint like this contains knowledge about the implementation details of LADDER instead of fundamental geometric information about how the shape should be drawn.

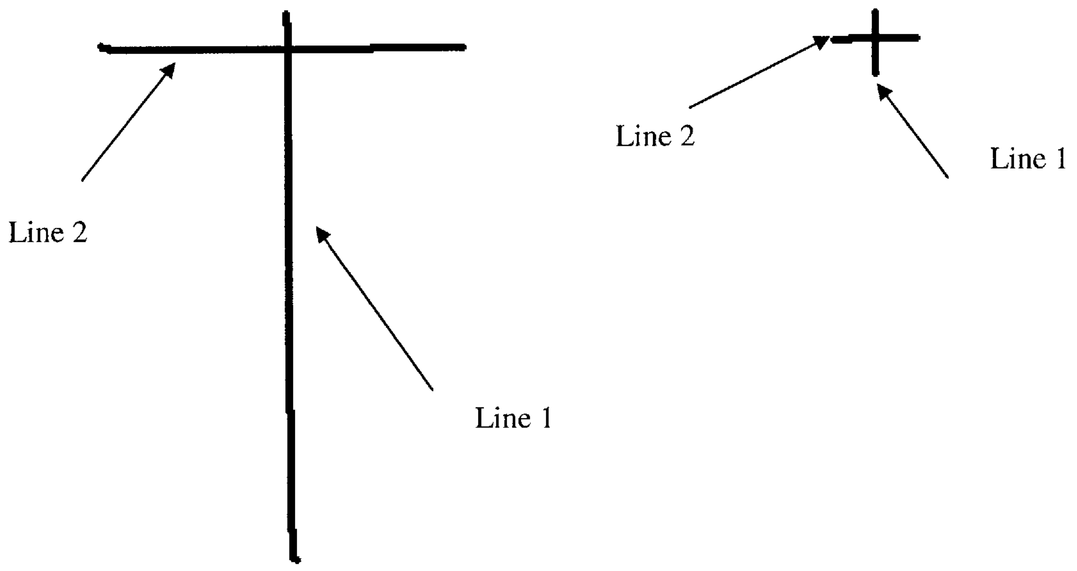


Figure 3.4.3. In both shapes, an endpoint of line 1 is within the threshold for intersection with line 2. However, while the shape on the left could very well be part of a capacitor, it is fairly clear that the shape on the right is actually a plus sign.

4. Related Work

As a synthesis of sketch-recognition and circuit reasoning, this thesis draws upon previous work in the two respective areas. Sketch-recognition systems have been around for a long time. Masayuki (1984) designed a system to recognized logic gates; Mahoney and Fromherz (2002) built a system to recognize stick figures; Hammond (2001) designed TAHUTI to handle UML diagrams. In the electronic circuit domain, Gennaria et al. (2005) created a system that uses ink density and a statistical classifier to identify circuit devices. Domain knowledge about circuits is used to help eliminate candidates that do not meet common circuit device constraints (e.g., a resistor should have 2 connectors while a transistor should have 3).

However, these systems all have a common problem – while they can involve domains with a deep and rich body of knowledge, these systems are very difficult for domain experts to build or to improve upon. The reason for this difficulty is because these systems do not have a clean separation between the basic stroke classification and the recognition of the domain shapes. Stroke classification requires signal processing knowledge and statistical expertise that is often totally unrelated to the domain knowledge that the system is built for.

By contrast, LADDER (Hammond 2004, 2001) enables a much cleaner separation between stroke processing knowledge and domain knowledge because it clearly divides the information from the user input into two categories; one for recognizing primitive shapes and a second one for recognizing complex domain shapes. Once the domain expert learns the LADDER language, he/she can easily specify how the domain shapes are drawn while a signal processing expert can improve upon the stroke classifier to aid in the recognition of the primitive shapes (e.g., lines, ellipses, arcs).

Automated electronic circuit analysis systems have an even longer history. Stallman and Sussman (1977) designed one of the first systems for constraint-propagation style circuit reasoning. De Kleer and Sussman (1978) built a system for automated circuit synthesis with the same style of reasoning. Constraint-satisfaction has a great advantage over matrix methods because its process is much more comprehensible to humans. Whenever a new value is deduced, the reasoner can generate the proper justifications for it, based on the constraints that were involved in the deduction.

Hanson and Sussman (2003) improved upon previous work by building a more complete circuit language that can be used to specify circuit devices. The user can combine device definitions to create more complex macro-devices – much like LADDER enables users to define shape hierarchies to create more complex domain shapes.

Finally, several systems that integrate sketch interfaces with a reasoning system have been designed. Alvarado and Davis (2001) built ASSIST to handle hand-drawn diagrams of physical objects and to simulate their motions. Liwicki and Knipping (2005) developed a system to handle digital logic circuits.

However, the analog circuit domain is fundamentally different from physics or digital circuits. It is not possible to animate the action of electric circuits as is done for physical objects. Unlike the motion of a ball on a plane, showing the flow of electrons has very little meaning to the user. Analog circuits are also different from digital logic circuits because in digital circuits only the binary input and output values are relevant. By contrast, analysis of analog electronic circuits must often take into account different models and regions of operation. Therefore, simply showing a computed value for a part of the circuit is of limited use. Justifications are needed as well.

5. Conclusion

5.1. Future Work

While the statistical significance of the pilot study is limited, the results do suggest that the recognition component of the system is too slow and could not adequately handle the variation in how a user draws circuit devices. Future research should focus first on improving the recognition sensitivity as well as the speed of online recognition. One important question to answer is whether the online recognition process can be made fast enough to allow for a more natural user experience. The recognizer constructed for the LADDER framework is intended to be a domain-independent shape recognizer. It is reasonable to posit that heuristics are possible when specialized recognizers using the LADDER framework are created for a particular domain. For example, the recognizer could use the number of corners as a heuristic to aid the recognition of resistors. Because users tend to draw devices with a single stroke, the number of corners in one stroke should be much higher for a resistor than any other circuit device. On the other hand, if a segmented stroke contains a large number of arcs, then the recognizer should look to see if an inductor has just been drawn. It may be possible for this heuristic to be generalized so that stroke level information (e.g., number of corners and arcs within a single stroke) can be used to tell the recognizer which shape definitions to check first. Designing and implementing heuristics for the electronic circuits domain may also help create a generalized mechanism to add recognition heuristics for the LADDER framework.

Another interesting research task would be to create the necessary components to enable the system to better recognize resistors and inductors. A better way to recognize resistors needs new language constructs in LADDER to allow for the specification of indefinitely repeated motifs. Recognizing the inductor requires the stroke level processor to successfully segment the substructures in an inductor.

Defining relationships between shapes is also an important feature that needs to be developed in the LADDER framework. In the current incarnation of the LADDER system, one can specify "contextual components" to get around this problem. However, the context in which a shape is analyzed should not be considered components of the shape. In other words, there should be a clear delineation between what is and is not considered a part of the shape.

On the reasoning side, additional circuit language constructs that allow for devices to have an undetermined number of parts should be created in order to enable the definition of equivalent resistor networks. A more complete inference system should also be created to enable the reasoning component to analyze circuits that cannot be solved without a system of equations. Stallman (1977) provides an example of a complete inference system able to solve simultaneous equations using symbolic manipulations.

The system should also have a mechanism that allows the user to represent groups of circuit devices with a single shape. For example, Figure 5.1.1.a shows a circuit that is called the common-emitter amplifier, a classic example circuit in many circuit theory courses. When a student first learn about common-emitter amplifiers, it is useful to draw all of the devices that make up this amplifier so that its characteristics can be fully analyzed. However, a student who is analyzing how amplifiers are used in loudspeakers will only care about the amplifier's input-output relation. Thus, instead of requiring the user to draw all of the component devices, the system should allow the user to designate a new and much simpler shape (Figure 5.1.1.b) to represent the entire common-emitter amplifier. In this manner, as more complex devices are added to the repertoire of the student, the system can be used to create and illustrate the increasingly complex layers of abstraction needed for analysis. Sussman (1977) presents an example of a descriptive language that can be used to create macro-devices from a combination of simpler ones.

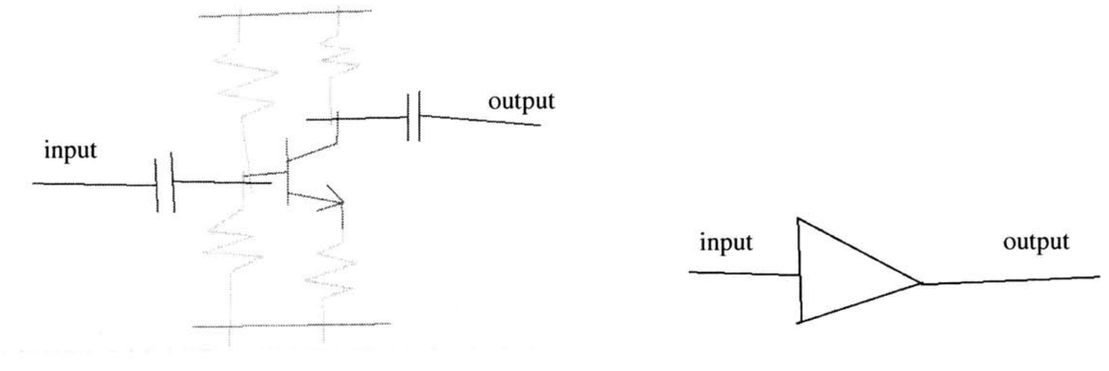


Figure 5.1.1. (a) shows a circuit called the common-emitter amplifier. However, if the student is using the amplifier as a component of a more complex device, only the input-output characteristics are relevant. As a result, the system should provide an easy way to allow the user to designate that the shape in (b) represents the entire group of circuit components shown in (a).

The issue of parameter specification should also be explored further. As circuit devices become increasingly complex, the number of parameters becomes more and more of a problem for the user. When multiple parameters exist for a given device, the user may be required to place multiple text inputs on or nearby the shape representing that device. The system should have an easy way for the user to designate which text input specifies the value for which parameter. This method of parameter specification should also allow the user to specify parameter values for complex devices without turning the drawing surface into a symphony of Greek letters.

To obtain a more rigorous measurement of the system's performance, larger user studies should be performed. First, as the recognition component is improved, a study with a much larger user base can give us more confidence about the sensitivity and specificity of the system. Moreover, it can also give us more accurate measurements of whether errors are caused by basic stroke processing or by incorrect shape descriptions and what the thresholds are when solving the constraints.

Additional usage scenarios should also be created to give users more options for interacting with the system. One important usage scenario would be to allow the user to specify guesses for the unsolved parts of the circuit. The system will then check whether the answer is correct and give immediate feedback to the user. This type of usage scenario would be an excellent tool in helping students complete practice problems in circuit analysis. Moreover, it would also enable the system to handle synthesis problems. Synthesis problems are fundamentally different from analysis problems because they are much less constrained. When designing circuits, there are often a large range of values that can be used for the parameters of each device. Therefore, the inference system cannot use the available constraints to deduce a unique answer. To give students guidance in synthesizing circuits, the system must also check for inconsistencies in parameter value combinations in addition to making deductions.

Once the recognition component is deemed to be sufficiently usable, a user study should be run to gauge the instructional value of the reasoning system. The user study should be set in a circuit theory classroom setting with one experimental subgroup that uses the circuit tutor as an instructional and supplemental learning tool. Students should be tested on their understanding of the material to see if there is any difference in performance between the experimental group and the rest of the class. Moreover, because the reasoning system provides its own guided flow of explanations, it may also be useful to measure the time savings for the instructor when preparing lesson plans. Of course, the time that the instructor spends learning how to use the software should be measured separately.

As with any experiment that involves humans, complete control is impossible to achieve. Several important issues need to be addressed in order to make this user study more effective. First, how can the software be available outside the classroom environment? While the system can still be useful even within the classroom, we suspect that its effect will be most prominent when used in an unsupervised setting. Second, if the software is available to students after class, how can control be maintained? Naturally students from the control group may be curious to use the software as well. Lastly, the system is only useful if the student uses it. However, one may reasonably guess that those who use the system most will be the students who are the most motivated. In general, these students can be expected to perform better than the rest of the class. Therefore, if we wanted to measure the effect of the system per amount of usage, we must separate out this self-selection effect from the effect of the system. All of these important issues must be considered when designing the user study.

5.2. Contributions

The system presented in this thesis integrates a sketch-recognition system with a constraint-propagation circuit solver to create a natural interaction circuit tutoring tool. The system provides several key improvements over the circuit illustration and tutoring systems available today. First, the user can sketch desired examples by hand instead of needing to learn how to use the drag-and-drop interfaces of CAD-like systems. Moreover, if sketch-based systems become widely available, it also means that the user no longer needs to learn a new way of creating diagrams whenever the software changes (for example, using Verilog versus SPICE).

Secondly, the construction of the recognition system was also a field test for the LADDER framework. The LADDER framework was designed to free builders of domain-specific sketch-recognizers from needing to also be an expert in the statistical processing of low-level stroke classification (Hammond 2004). This is the first time that the LADDER framework was used to create such a sketch-recognition system. Several deficiencies of LADDER were identified, and possible ways to deal with those problems are described in section 5.1.

The integration of the reasoner with the recognizer also provides several benefits. The reasoning system automates the generation of guided examples. This may save time for instructors and allow examples to be automatically explained in unsupervised settings. The ability to allow students to create examples and to customize the detail level and speed of the explanations means that the system can be adapted to the pace and learning style of each individual student. In this manner, the automated reasoning and immediate feedback may make this system a very useful learning supplement. Further user studies can show the extent to which this effect helps students learn in a circuit theory class.

6. References

Alvarado, C., and Davis, R. 2001. "Resolving ambiguities to create a natural computer-based sketching environment", *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), Seattle (WA), 1365–1374.

De Kleer, J. and Sussman, G.J., 1978. "Propagation of Constraints Applied to Circuit Synthesis", *Massachusetss Institute of Technology Artificial Intelligence Laboratory Memo No. 485.*

Gennaria, Leslie., Levent Burak Karaa, Thomas F. Stahovichb, Kenji Shimadaa, 2005. "Combining geometry and domain knowledge to interpret hand-drawn diagrams". *Computers & Graphics*, 29 (2005) 547 - 562.

Hammond, T., and Davis, R. 2004. "Automatically Transforming Symbolic Shape Descriptions for Use in Sketch Recognition", *AAAI.*

Hammond, T., and Davis, R. 2003. "LADDER: A Language to Describe Drawing, Display, and Editing in Sketch Recognition", *Proceedings of the 2003 International Joint Conference on Artificial Intelligence.*

Hammond, T., and Davis, R. 2002. "Tahuti: A Sketch Recognition System for UML Class Diagrams", *AAAI Spring Symposium on Sketch Understanding.*

Hanson, C., and Sussman, G.J., 2003. "Circuit Language for the Intelligent Book", *Intelligent Book Project*, MIT Project for Mathematic and Computation.

Liwicki, M., and Knipping, L. 2005. "Recognizing and Simulating Sketched Logic Circuits". *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information & Engineering Systems*, LNCS, Volume 3683, pp. 588 – 594.

Mahoney, J.V., and Fromherz, M.P.J. 2002. "Handling Ambiguity in Constraint-based Recognition of Stick Figure Sketches", *SPIE Document Recognition and Retrieval IX Conference*, San Jose, CA.

Masayuki, N., and Takeshi, A., 1984. "Pattern recognition for logical circuits diagrams written by freehand". *Technical Report 015 - 002, SIGNotes Computer Graphics and cad*

Sezgin, T. M.; Stahovich, T.; and Davis, R. 2001a. "Sketch based interfaces: Early processing for sketch understanding", *The Proceedings of 2001 Perceptive User Interfaces Workshop* (PUI'01).

Sezgin, T.M., 2001b. "Feature Point Detection and Curve Approximation for Early Processing of Free-Hand Sketches", *Master's Thesis*, Massachusetts Institute of Technology.

Stallman, R., and Sussman, G. J., 1977. "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", *Artificial Intelligence* 9:135.

Sussman, G.J., 1977. "SLICES: At the Boundary Between Analysis and Synthesis". MIT Artificial Intelligence Memo No. 43.

Appendix A. User Manual

Software Requirements

To run the reasoning component, the user should download the latest version of MIT/GNU Scheme and install it in the default directory (C:\Program Files\MIT-GNU Scheme).

Launching the System

After MIT/GNU Scheme has been installed, the user can now launch the system by running "CircuitTutorMain" under the package edu.mit.sketch.language.circuitTutor.gui.

Main User Screen

The main user screen is split into three areas. First, the text area at the top of the window is used to relay messages to the user. The drawing surface in the middle is where users will sketch the circuit. The slider and the buttons on the bottom of the window can be used to trigger commands for both sketch editing and circuit analysis.

Drawing Surface

When the user places the pen down on the drawing surface, sketching begins. The path traced by the user between pen down and pen up is considered a stroke. The stroke is then processed by the recognizer. The color of the stroke can be changed by the recognizer depending on how the stroke was classified. When primitive shapes meet the constraints for a more complex shape, the primitive shapes may also change colors.

Selection and Deletion are the editing commands that are triggered using the pen. When the user clicks on a location on the drawing surface, the shape whose strokes passes through the clicked location is selected. The user can choose to click on a selected shape to choose a sub-shape of it, or to click on an empty spot on the drawing surface to deselect all selected shapes. Once shapes are selected, the next stroke is checked to see if it intersects the selection. If the stroke intersects any of the selected shapes, then all selected shapes are deleted, along with the new stroke. If the new stroke does not intersect any of the selected shapes, then all selected shapes are deselected and the new stroke is processed by the recognizer.

When the system is recognizing strokes, often times the drawing surface may not register the new strokes being drawn by the user. If "Recognizing…" is displayed on the upper-left hand corner of the drawing surface, it is recommended that the user allow the recognizer to finish before drawing another stroke.

Command Buttons and Slider

Slider – controls the speed of the reasoning playback. Sliding the indicator to the left speeds up the playback, sliding to the right slows down the playback. The speed cannot be changed after the user presses the "START" button.

Disable/Enable Drawing Board – toggles whether the drawing surface accepts user input.

Load Component List – can be used to change the list of LADDER shape descriptions that the system uses to recognize shapes. CircuitTutor.ldl is the list used by the system in this thesis.

Clear – clears the drawing surface of all shapes. This action cannot be undone.

Print Shapes – writes the text representations of the drawn shapes to the user message area at the top of the main user screen.

Undo – removes that last stroke that was drawn. If the last action was a deletion, the deleted stroke is restored.

Options – Clicking on this button will bring up a popup window containing the devices and relations that the system handles. This allows the user to specify which reasoning justifications to display and which ones to skip.

Start – begins the reasoning process.

Pause/Continue – pause or resume the reasoning playback.

Back – while the system is paused, rewinds the reasoning playback by one step of the reasoning process.

Forward – Proceeds to the next step of the reasoning playback process.

Text Recognition

The text recognizer is a popup window triggered by double-clicking or right-clicking on any spot on the drawing surface. The recognizer window has a user input box where the user writes the text that he or she wishes to place on the drawing surface.

OK – recognizes the strokes drawn on the user input box. It automatically selects the highest ranked guess made by the text recognizer and places it on the drawing surface.

Cancel – cancels the text recognition. Nothing is added to the recognizer.

Recognize – Shows a list of possible candidates for the strokes that the user drew in the text input box. The user can choose a candidate and then press the "ok" button to add the selected candidate string to the drawing surface.

Clear – clears all strokes drawn by the user in the text input area.

Circuit Analysis

Before analyzing the circuit, the options and the display speed can be adjusted for the user. Once the user clicks "START", the drawing surface as well as the options and display speed buttons will be disabled. Only after the "RESET" button is pressed will the drawing surface be enabled again. After the user presses start, the deduced values for the circuit as well as their justifications will be shown on the drawing surface. The text that is used to display the deduced values does not participate in recognition and is removed when the "RESET" button is pressed.

Appendix B. User Study Data Tables

Summary Statistics:

	Overlay	Tablet	Mean	Difference
Total devices drawn	100	100	100	0
Average # of Devices recognized	41	53	47	-12
Average # of Stroke level errors	30	20.3	25.15	9.7
Average # of Constraint level errors	29	25.7	27.35	3.3
Stroke error / total error	51%	44%	47.5%	N/A
Constraint error /total error	49%	56%	52.5%	N/A
Undos per diagram	3.51	0.96	2.24	2.55

Results for the Interactive Overlay Group broken down by diagram:

	1	2	3	4	5	6	7	8
# recognized	1.60	2.20	2.20	4.00	3.00	1.20	2.20	1.40
Stroke errors	1.20	0.40	1.80	1.60	1.00	1.40	2.60	2.80
Constraint errors	0.20	1.40	2.00	2.40	3.00	1.40	1.20	2.80
Undo #	4.40	1.80	1.00	5.80	4.00	3.80	2.60	5.00

	9	10	11	12	13	14	15	Mean
# recognized	4.00	6.80	1.80	2.60	2.80	3.60	1.60	41.00
Stroke errors	3.00	6.00	1.80	0.80	3.00	1.80	0.80	30.00
Constraint errors	2.00	3.20	2.40	1.60	3.20	1.60	0.60	29.00
Undo #	7.80	4.60	1.40	3.60	2.20	3.20	1.40	3.51

Results for the Table PC Group broken down by diagram:

	1	2	3	4	5	6	7	8
# recognized	1.33	2.00	2.33	3.67	3.67	2.67	3.33	4.00
Stroke errors	0.00	0.33	1.67	1.67	1.00	0.33	2.67	1.33
Constraint errors	1.67	1.67	2.00	2.67	2.00	1.00	0.00	1.67
Undo #	0.00	0.00	0.00	1.00	1.33	1.67	1.33	1.67

	9	10	11	12	13	14	15	Mean
# recognized	5.67	7.67	4.33	2.67	4.33	4.00	1.67	53.33
Stroke errors	1.00	5.67	0.67	0.67	1.67	1.67	0.00	20.33
Constraint errors	2.33	2.67	1.00	1.67	3.00	1.00	1.33	25.67
Undo #	1.00	1.33	0.00	3.00	1.67	0.33	0.00	0.96

Improvement in Number of Devices Recognized between pairs of identical diagrams:

	#1, #15	#3, #11	#5, #14	% improvement
Overlay Group	0.00	-0.40	0.60	0.01
Tablet Group	0.33	2.00	0.33	0.17

Appendix C. Guide to System Implementation

The software architecture of the system presented in this thesis can be divided into 5 major parts – the user interface, the recognizer, representational translation, the reasoner, and the reasoning playback. The majority of classes reside in the Java package "edu.mit.sketch.language.circuitTutor". Unless otherwise stated, all package names used in this section are relative to this root package.

C.1. User Interface

Most of the user interface resides in the package named gui. The text recognizer was build using C# and does not fall under the circuitTutor source directory. The components and subpackages of the user interface are described below.

Package: gui

CircuitTutorMain is the class that is used to launch the system. It contains nothing other than a main method, which constructs the JFrame containing the various parts of the user interface.

CircuitTutorFrame is a container for the other parts of the user interface.

CircuitDrawCheckPanel coordinates amongst the different gui components that is shown on the main user screen. It contains a JTextArea object used to relay system messages to the user, a CircuitDrawPanel object that is the drawing surface, and a JPanel object containing the command buttons available to the user. This class also contains references to the reasoning controller (AnalysisController) and the reasoning playback controller (InferenceDisplayController) in order to connect the gui, the recognition component, and the reasoning component.

CircuitDrawPanel is the drawing surface that the user sketches on. It contains MouseListeners to relay user strokes to the recognition component. It also contains methods used by other classes to add shapes to the recognition component directly.

AnalyzeButtonsPanel inherits from JPanel. It contains the command buttons for the user to interact with the system as well as the slider used to adjust the speed of reasoning playback.

Package: gui.analysisOptions

AnalysisOptionsPopup inherits from JDialog. It is the popup window that allows the user to specify which relations are displayed when the justifications for deducted values are shown. The list of relations is organized by device type. This dialog also allows the user to (de)select all relations, all KVL relations, or all KCL relations at once.

RuleNode & RuleNodeRenderer inherit from the JTree abstraction and is used by the AnalysisOptionsPopup to organize and display the relations in a tree structure.

Package: gui.editing

SelectionListener handles the actions of the system when a user clicks on a shape. The SelectionListener performs two main tasks. First, when the user clicks on the drawing surface, it checks whether the user has clicked on a shape. Second, the SelectionListener listens for changes to the list of selected shapes so that it can automatically changes the color of the selected and deselected shapes accordingly.

DeletionHandler handles the actions of the system when a user deletes selected shapes. This is accomplished by intercepting strokes before they are passed to the recognizer. The DeletionHandler checks if a stroke that was just drawn intersects any of the shapes that were selected. If so, all selected shapes are deleted. The deletion stroke is deleted from the drawing surface as well without ever going through the recognition process.

ViewableSelectedSynchronizingListener is a listener added to the list of visible shapes on the drawing surface, when a shape is deleted from the visible shapes list, ViewableSelectedSynchronizingListener will automatically delete the same shape from the selected shapes list.

Package: textRecognizer

CInkHandler communicates with the C# text recognition component by sending and receiving strings via a TCP socket connection. It inherits from Thread and whenever it receives string input from the text recognizer, it sends the input to the TextRecognizer.

TextRecognizer creates a new CInkHandler thread and launches the text recognition window (written in C#). It has a method that allows other classes to specify where the popup window should appear, and also a method that sends the string input from the CInkHandler to the drawing surface.

Non-Java Code

TextRecognizer.exe (not to be confused with TextRecognizer.java in package gui.textRecognizer) is the text recognition panel written in C#. It does not reside in the Java packages. Instead, it can be found in drg/language/circuitTutor/TextRecognizer. The C# source code for the text recognizer can be found in code/windows/TextRecognizer. This application is the popup window that is triggered when the user double clicks on the drawing surface. It communicates with the main system by sending and receiving strings via a TCP socket connection.

C.2. Recognizer

The code for the LADDER framework was built by Tracy Hammond. Only the extensions to the LADDER recognizer are listed here

Package: toolkit

SimpleClassifier3Biased is a stroke classifier that inherits from the stroke classifier built by Metin Sezgin. It biases the classification process so that arcs do not have to be drawn as carefully to be classified correctly.

CircuitTutorActionFactory handles communication between the drawing surface and the recognizer. Strokes drawn by the user are first given by the CircuitTutorActionFactory to the DeletionHandler to see if the stroke should delete any selected shapes. If the new stroke is not a deletion stroke, then the CircuitTutorActionFactory will hand it off to the recognizer.

C.3. Representational Translation

The representational translator component consists of classes and subpackages under the package named "reasoner".

Package: reasoner

ComponentLoader translates the shapes drawn by the user into the appropriate representation for the reasoner. To do so, it uses an intermediate representation of circuit devices. The intermediate representation has methods that return the appropriate Scheme expressions (String objects) that define each circuit device instance for the reasoner.

Package: reasoner.circuitRepresentation,
reasoner.circuitRepresentation.primitives

The classes contained in these two packages make up the intermediate representation used by the ComponentLoader. The circuitRepresentation package contains the generic definitions while the circuitRepresentation.primitives subpackage defines specialized types to represent common devices like a voltage source, current source, resistor, capacitor, inductor, ground, and transistor. Instances of these types should be created with the class ***PrimitiveComponentFactory***.

C.4. Reasoner

The reasoning component was developed by Hanson and Sussman (2003). The reasoner was written in Scheme and is contained in the file named "reasoner-write-to-java.com". This file resides in the directory "drg/language/circuitTutor/reasoner". The reasoner communicates with the other parts of the system via a TCP socket connection. Several classes were created to allow the system to interact with the reasoner.

Package: reasoner

AnalysisController is in interface that handles communication between the recognition component and the reasoner.

CircuitAnalysisController implements the AnalysisController interface. It calls ComponentLoader to translate the shapes into the appropriate circuit representation before sending the circuit to the reasoner (Scheme) for analysis.

Package: toolkit

SchemeConnectionManager is a thread that communicates with the reasoner via a TCP socket connection. It provides methods to write Scheme expressions to the reasoner so that the flow of analysis can be controlled. It also saves the output from the reasoner so that it can be read at another time.

C.5. Reasoning Playback

Package: reasoner

InferenceDisplayController is an interface between the reasoning system and the drawing surface. It provides methods that allow the user to pause, rewind, and change the speed of the reasoning playback.

CircuitDisplayController implements InferenceDisplayController. It receives the circuit from the AnalysisController and uses the ReasoningDisplay to place the appropriate reasoning output onto the drawing surface.

ReasoningDisplay uses the SchemeConnectionManager to communicate with the reasoner to obtain the appropriate values and justifications. It filters the justifications, showing only the ones requested by the user. It also controls the pace of the playback according to the display speed set by the user. The ReasoningDisplay runs on its own thread so that displaying the justifications can happen in tandem to user commands (e.g., pause and reset).