From: AAAI Technical Report SS-02-08. Compilation copyright © 2002, AAAI (www.aaai.org). All rights reserved.

Interpreting Schematic Sketches Using Physical Reasoning

Tolga Kurtoglu and **Thomas F. Stahovich**
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
stahov@andrew.cmu.edu

Abstract

We describe a program that uses both geometric and physical reasoning to interpret schematic sketches of physical devices. The program takes freehand sketches of physical devices as input. It begins by recognizing the symbols used in the sketch. It then uses geometric and physics-based reasoning techniques to identify the meaning of each symbol within the context of the sketch. The physical reasoning relies on qualitative behavior models, simple qualitative models describing energy flow through physical components. Through the physical reasoning, the program is able to disambiguate the meanings of the individual symbols and produce a natural language (text) description of how the sketched device would operate.

Introduction

Sketching plays an essential role in the early stages of product design, when a product is nothing more than a collection of abstract ideas. Sketches are used in an iterative manner to instantiate design ideas and stimulate new ones. In each cycle, designers express their ideas as sketches, which are then examined and reinterpreted. This examination may inspire new ideas, which are then drawn, reexamined, reconceived, redrawn, and so on (Suwa & Tversky 1997). This iterative usage of sketching supports the designer's creativity and productivity, and increases the quality of the final product. Sketching is also an essential tool for capturing and communicating design ideas. Our informal experiments suggest that describing devices graphically is usually substantially easier than describing them verbally.

Because sketching is such a useful tool for design, we would like to develop techniques that enable sketch-based user interfaces for CAD tools. Traditional CAD software relies on mouse driven graphical user interfaces (GUI's). Although these interfaces are intended to be user-friendly, often a considerable amount of training and experience is required before the tools can be used efficiently. Furthermore, the overhead of traditional GUI interfaces often interferes with the designer's creativity. We believe that sketch-based interfaces will greatly increase the utility of CAD tools, even in the earliest conceptual phases of design.

There have already been a number of efforts aimed at developing sketch-based applications. These can be grouped into two major themes: Systems for creating shapes and systems for recognizing shapes. Examples of the first group include (De Bonet 1995), who built a system that reconstructs rectangular polyhedra from hand drawn wire-frame sketches. (Shpitalni & Lipson 1996) tackled a similar problem and constructed a program that converts a sketch of a 3D wire-frame into a 3D model using 2D edge-vertex graphs. (Eggli *et al.* 1997) developed a pen-based tool for 2D and 3D modeling. With this tool, one uses simple pen strokes to define a model. Exact shapes and geometric relationships are inferred from the sketch using a graph-based constraint solver.

Shape recognition approaches include the work of (Rubine 1991) who developed a technique for recognizing pen strokes. He considered symbols composed of single pen strokes. His program identifies a symbol by computing various geometric properties, such as the total stroke length, and comparing them to previous examples. (Fonseca & Jorge 2000) developed a similar method for recognizing multi-stroke shapes. Shapes are characterized by the convex hull of the pen strokes, and special polygons inscribed in the convex hull. (Calhoun *et al.* 2002) have also developed a trainable recognizer for multi-stroke symbols. They define a symbol in terms of the constituent geometric primitives, their properties, and the relationships between them. Because the method considers the constituent parts of the symbol, rather than aggregate properties, it can distinguish between similar shapes and can also identify the individual parts of a symbol.

In our research, we are attempting to recognize schematic sketches of mechanical devices like the example shown in Figure 1. Our work has something in common with shape recognition, because as a first step, we must recognize the individual symbols in the sketch. (We employ the trainable symbol recognizer described in (Calhoun *et al.* 2002).) The difficulty is that symbols frequently have multiple meanings. For example, the symbol commonly used to represent an electrical resistor is the same as the one used to represent a spring. Determining which meaning is intended requires understanding the overall context of the sketch. To understand context, we have developed an approach based on qualitative physical reasoning. Thus our sketch system uses both

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

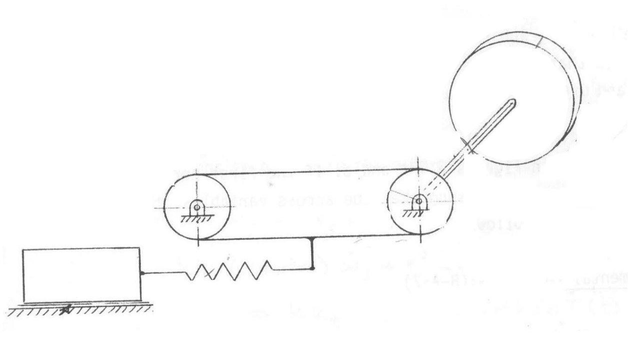


Figure 1: A sketch taken from a system dynamics textbook (Platin, Caliskan, & Ozguven 1991).

geometric reasoning and physical reasoning to interpret a sketch.

A primary application for our techniques is as a user interface for analysis tools. Many types of analyses begin with schematics. For example, schematics are common in thermodynamics, system dynamics, controls, and electrical engineering. In fact, the sketch in Figure 1 is taken directly from a textbook on system dynamics (Platin, Caliskan, & Ozguven 1991). The techniques we are developing will enable analysis tools to work directly from the types of simple schematic sketches engineers already commonly draw. Such tools will allow the user to focus on the analysis itself, rather than on how to operate the analysis tool. Furthermore, software capable of working from quickly drawn sketches will be of particular use in the early creative phases of design, when many ideas are explored at a shallow level, and many of the details have yet to be determined.

Our work is related to that of (Narayanan, Suwa, & Motoda 1994), who use a diagram of a device to reason about its behavior. They start with a pre-parsed description of the behaviors of each component. They then use a grid-based description of the sketch to perform geometric calculations required for computing a qualitative simulation. In effect, they use the sketch as a tool for simulating devices with known behavioral models. By contrast, we use behavioral reasoning, including qualitative simulation, as a tool for identifying the behaviors of the various symbols in a sketch. An interesting distinction is that, in some sense, they reason “with” the sketch while we reason “about” it.

Our work is complementary to Stahovich’s SketchIT system (Stahovich, Davis, & Shrobe 1998), which can transform a rough sketch of a mechanical device into multiple families of new designs. SketchIT is concerned with devices composed of rigid bodies and springs. By contrast, the current system is concerned with devices composed of a variety of components such as motors, heaters, pulleys, wheels, etc. SketchIT uses behavioral reasoning to identify geometric constraints that ensure that an assembly of rigid bodies and springs will produce an intended behavior. The system described here uses behavioral reasoning to identify the intended meaning of each of the symbols in a schematic.

Our physical reasoning techniques are related to work in compositional modeling (Falkenhainer & Forbus 1991; Mashburn & Anderson 1994; Nayak, Joskowicz, & Addanki 1992). These techniques automatically construct a model for a system by selecting appropriate model fragments for each of the components. Typically, the process is driven by a user query about a causal relationship between two state variables. Our task is similar in that sometimes we use sketched annotations of the intended behavior to guide the interpretation process. However, our approach can work in the absence of such annotations. Our task is to identify what kind of component each symbol in the sketch represents. By contrast, compositional modeling begins with a description of what each component is, and then selects the appropriate behavioral model for each component in order to answer a user query. The two tasks are analogous, addressing similar problems, but at different levels of detail.

Our qualitative simulation technique is similar to traditional qualitative techniques such as (de Kleer 1979) and (Williams 1984). Those techniques operate by propagating *changes* in the magnitudes of physical quantities. Our technique reasons about the propagation of nominal signals through a device.

Hardware

Our sketching system is implemented on a rear projection electronic whiteboard as shown in Figure 2. The data projector inside the whiteboard projects the computer desktop (screen image) onto the full area of the whiteboard. Our software is window-based, with a drawing area and button bar at the top of the window. We run the software in full screen mode, so that it occupies the entire whiteboard. To draw, one uses inkless pens (or a finger tip) on the touch sensitive drawing surface of the whiteboard. The software projects virtual ink as each pen stroke is made. After drawing each symbol, the user presses the pen to the “interpret” button in the tool bar, indicating that the symbol should be recognized. After recognizing the symbol, the software replaces the original pen strokes with “beautified” strokes. Once the entire sketch has been drawn, pressing another button initiates the interpretation process, resulting in a description of the behavior of each individual component and the device as a whole. The software is designed to work with a variety of hardware environments. For example, it can be used on a standard computer with a mouse or digitizing pad and stylus.

The Approach

Figure 3c shows a simple example of the kind of sketches one might find in an engineering document. In examining the sketch, we identify several primitive drawing entities that we will call *symbols*. There are three symbols in the current example: The symbol with a vertical line and crosshatching labeled S1, the symbol with a set of angled line segments labeled S2, and the square symbol labeled S3.

These symbols are familiar to most engineers, and one would easily identify Figure 3c as a mass-spring oscillator. However, the problem is more interesting if the symbols are considered individually. The single symbol in Figure 3a,

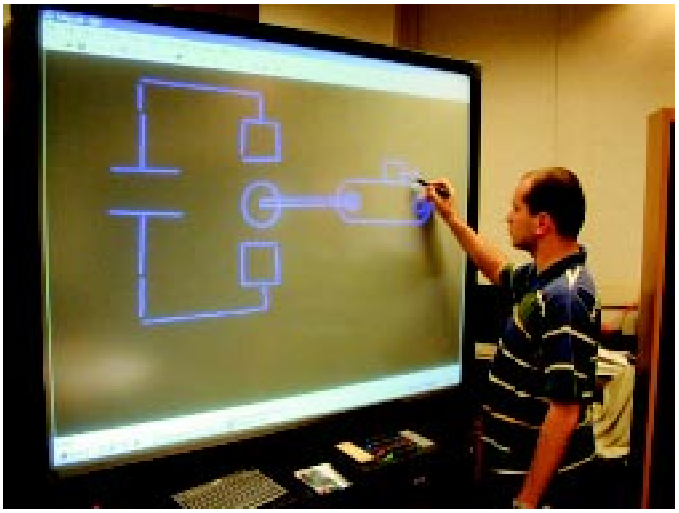


Figure 2: Sketching environment: rear projection whiteboard.

for example, could have multiple interpretations. To a mechanical engineer this might represent the rack of a rack and pinion assembly, or it might be a spring. To an electrical engineer, this same symbol is an electrical resistor. Adding a second symbol, as in Figure 3b, adds more information, but there are still multiple interpretations: This could be a spring with one end fixed to the frame, or it could be an electrically grounded resistor, for example. It is not until all three symbols are present, that we can reliably identify the sketch as a spring connected between a rigid frame and a mass.

As this example illustrates, with schematic sketches there are often multiple ways of interpreting the meanings of the symbols. The same symbols are often used to represent very different physical components. Figure 4 shows more examples of this. Notice that the symbol representing a battery in the conveyor system (Figure 4a) is used to represent a bearing in the piston-cylinder system (Figure 4b). Similarly the symbol representing a heater in the piston-cylinder system (Figure 4b) represents a spring in the oscillator (Figure 4c). Likewise, the rectangular geometry representing a shaft in the conveyor system (Figure 4a) represents a rigid body in the piston-cylinder system (Figure 4b).

Unambiguously interpreting the meanings of such symbols would be straightforward if there were a one-to-one mapping between them and the physical components they represent. The problem would reduce to implementing an efficient symbol recognition algorithm to identify the individual pieces of the sketch. However, because the same symbols are used to represent different physical components, the problem becomes more than shape recognition.

Returning to the sketch in Figure 3, what allowed us to uniquely identify the meaning of each symbol was the context provided by the complete set of symbols in the sketch, taken as a whole. The correct interpretation for each symbol was the one that was consistent with the interpretations of all of the other symbols in the sketch. The problem is,

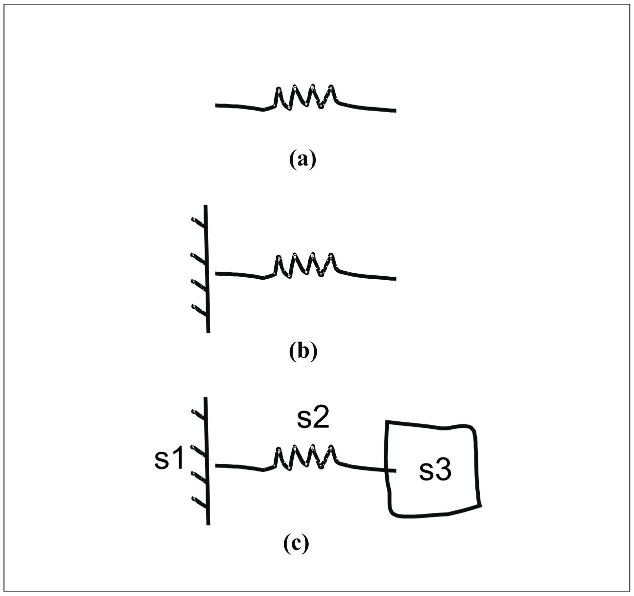


Figure 3: Various pieces of a schematic sketch.

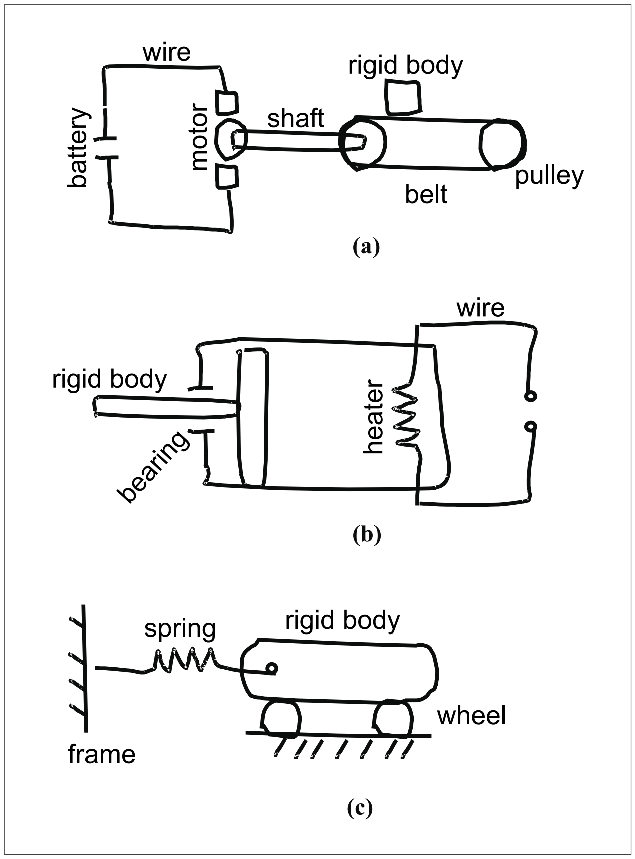


Figure 4: (a) A motor driven conveyor system. (b) A piston-cylinder system. The heater causes the gas to expand and move the piston. (c) A mass-spring oscillator with wheels.

in essence, a constraint satisfaction problem (CSP). The first step in the solution of the problem is to enumerate all possible interpretations of the individual symbols. For each symbol, we then filter out the interpretations that are inconsistent with the interpretations of the other symbols to which it is connected. (This requires both local and global consistency checks as described in Section "Physical Compatibility.") For example, in Figure 3 we can eliminate the resistor interpretation for symbol S2 because it is inconsistent with the rigid body interpretation of symbol S3. This in turn allows us to filter out the electrical ground interpretation of symbol S1, because symbol S2 no longer has the possibility of being a resistor.

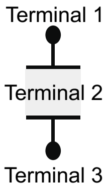
After all of the inconsistencies have been filtered out using both local and global analysis, sometimes there are still multiple globally consistent interpretations for a sketch. Our approach relies on qualitative simulation to identify the intended interpretation in that case. The user sketches special symbols indicating that particular parts in the sketch are supposed to rotate or translate. The program selects the global interpretation that could cause the specified motion to occur. An additional benefit of this simulation capability is that it allows the program to generate a natural language (text) description of how the sketched device would operate.

Symbols

We reviewed various engineering textbooks to collect a list of commonly occurring symbols and cataloged their possible interpretations. To simplify the symbol recognition task (i.e., the first step in the interpretation process), we simplified some of the symbols by replacing isometric geometries with simpler 2D geometries. For example, as a simplification, we draw shafts as simple bars like the one in Figure 4a.

Physical components are often designed to have a set of *terminals* or ports through which they connect to one another. Components designed in this way interact with other components only through connected terminals. Our approach makes use of this fact to simplify the task of identifying component interactions. Included with the library definition of each symbol is the set of terminals that the corresponding physical component possesses. Each terminal has both a *geometric type* and a *physical type*. The geometric type describes the shape of the terminal: a point, a line, an arc, or an area (region). There are five physical types: translational mechanical, rotational mechanical, electrical, toothed mechanical, and fluidic. The geometric type of a terminal is used when performing proximity checks to determine if two terminals are close enough to be connected. The physical type is used to determine if a connection is physically possible: For the connection between a pair of terminals to be valid, the physical types of the terminals must be the same.

Figure 5 shows an example of our representation of a symbol. The symbol shown has two interpretations: It could be either a bearing or a battery. Viewed as a bearing, there are three terminals. Terminals 1 and 3 are point terminals whose physical type is translational mechanical. Terminal 2, which is shown as a shaded region, is an area terminal. The extents of this terminal are defined by the two parallel, horizontal line segments. The physical type of this terminal is rotational mechanical. Viewed as a battery, only the two point terminals are relevant, and their physical type is electrical.



Terminal	1	2	3
Geometric type	Point	Area	Point
Bearing Physical type	Trans. Mech.	Rot. Mech.	Trans. Mech.
Battery Physical type	Electrical		Electrical

Figure 5: Terminals for the symbol that represents bearings and batteries. The point terminals are exaggerated for clarity.

Symbol recognition

Our program uses a trainable recognition system to recognize the individual symbols in a sketch. To train this system, which is described in detail in (Calhoun *et al.* 2002), one provides a few examples of each symbol. The terminals are then manually defined by mapping them onto the geometric primitives (points, lines, arcs, etc.) in the learned symbol definitions.

When creating a sketch, the user draws symbols one at a time. After each symbol, the user touches the pen to a "button" on the drawing surface to indicate that the symbol has been completed and should be identified. (As an alternative to requiring a button press, the software could detect pauses in the sketching process.) Once a symbol has been identified, the locations of its terminals are obtained by extracting the locations of the corresponding geometric primitives.

Connectivity

We assume that symbols interact only through connected terminals. Thus, prior to reasoning about component interactions, we must determine which terminals are connected. To reduce computation, our program first employs a bounding box test. If the bounding boxes of two symbols do not intersect, it is not possible for their terminals to be connected, and no further analysis is necessary.

If the bounding boxes do intersect, each terminal of the first symbol is checked for proximity to each terminal of the second symbol. Note that at this point, no interpretation has been selected for the symbols. Thus, the program considers the union of the terminal sets for all of the interpretations of a symbol. Irrelevant connections will be discarded later. For example, if the symbol is the one in Figure 5, the program will consider possible connections to all three terminals. If the symbol later turns out to be a battery, connections to terminal 2 will be ignored.

The program identifies connected terminals by applying simple proximity tests. One test is defined for each combination of geometric terminal types. The tests employ tolerances to handle cases in which terminals were intended to be connected, but because of a slightly misplaced pen stroke, they are not. The complete set of tests is:

- *Point - point proximity:* True if the distance between the two points is less than a tolerance.
- *Point - line proximity:* True if the point is "between" the end points of the line segment and the minimum distance from the point to the line segment is less than a tolerance. A point is between the end points of a line segment if a perpendicular from the point to the extended line segment intersects the segment between its end points.
- *Point - arc proximity:* True if the minimum distance from the point to the arc segment is less than a tolerance.
- *Point - area proximity:* True if the point is inside the polygon defining the area. If a random ray extended from the point intersects the polygon an odd number of times, the point is inside.
- *Line - line proximity:* True if the two line segments are parallel within a tolerance, the minimum distance between the lines is less than a tolerance, and at least one of the endpoints of the shorter line segment lies "between" the endpoints of the longer line segment. (See point - line proximity.)
- *Line - arc proximity:* True if the minimum distance from the line segment to the arc segment is less than a tolerance and the line is tangent to the arc. The line is tangent if a line perpendicular to the extended line segment and passing through the center of the arc, intersects the line segment itself.
- *Line - area proximity:* True if at least one endpoint of the line segment lies inside the polygon defining the area.
- *Arc - arc proximity:* True either if the distance between the centers of the two arc segments, minus the sum of their radii, is less than a tolerance. Also true if the two arc segments are concentric and have the same radius (within tolerance).
- *Arc - area proximity:* True if an endpoint of the arc segment is inside the area. (See point - area proximity.) For a full circle, this is true if the circle intersects at least one of the line segments defining the area.
- *Area - area proximity:* True if at least one of the line segments defining one of the areas intersects at least one of the line segments defining the other area.

Physical Compatibility

After identifying which terminals are connected, the next step is to determine which symbol interpretations would result in those connections being physically meaningful. This problem is a classical constraint satisfaction problem, or *CSP*. Each symbol has a finite set of possible interpretations. The connections between the terminals act as constraints on the legal interpretations of the symbols. (Recall that two terminals can be connected only if their physical types are the same.)

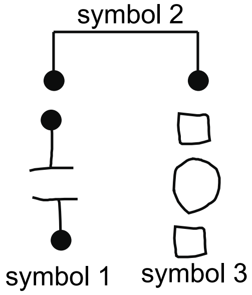
One could conceive of solving this problem with brute-force search, by enumerating all possible combinations of interpretations and filtering out those with invalid terminal connections. Doing so, however, would be expensive. Instead, we use more efficient techniques that preprocess the search space and eliminate any local inconsistencies before searching for globally consistent solutions (see (Waltz 1975; Mackworth 1977)). The particular technique we use is similar to that used in QSIM (Kuipers 1994) to identify consistent values for the qualitative state variables when computing a step of qualitative simulation.

We use two preprocessing steps. The first step is called *node consistency*. Each pair of symbols connected by one or more pairs of terminals is represented by a node in the constraint graph. A node is initialized by enumerating all possible pairs of interpretations for the two symbols. The node consistency filter eliminates all inconsistent interpretation pairs from each node. An interpretation pair is inconsistent if any pair of connected terminals has differing physical types.

Figure 6 shows an example of the node consistency filter. The example considers three symbols from the conveyor system of Figure 4a. Proximity analysis reveals that one point terminal on symbol 1 is connected to one point terminal on symbol 2. The other point terminal on symbol 2 is connected to a line terminal on symbol 3. The initial lists of interpretation pairs for the two nodes are shown Figure 6. Applying the node consistency filter to the first node, the one for symbols 1 and 2, prunes the first and fourth interpretation pairs because they would result in a translational mechanical terminal being connected to an electrical one. In the node for symbols 2 and 3, only the first interpretation pair survives the filtering.

After performing node consistency, we begin the second step, called *arc consistency*. Here we consider pairs of nodes having a symbol in common. Such pairs are said to be connected by an arc in the constraint graph. The arc consistency filter is applied to each arc. It eliminates any interpretation pair form one node that has no match in the other node. For example, in Figure 6, the two nodes have symbol 2 in common. For node 1, symbol 2 can be either a wire or a mechanical connector, but for node 2 it can be only a wire. Thus, we can filter from node 1 the interpretation as a mechanical connector. Thus, after node and arc consistency filtering, each of these two nodes has only one remaining interpretation pair.

The arc consistency filter is applied iteratively. In each iteration, the filter is applied to every pair of nodes with a common symbol. If any interpretation pairs are eliminated during an iteration, another iteration must be performed. Iterations must continue until there is a complete iteration with no eliminations. To see why the iterations must run to quiescence, consider node 1 in Figure 6. Imagine that the arc consistency filter had already been applied to an arc between node 1 and some other node. Further, imagine that



Node 1

	Symbol 1	Symbol 2
~~1~~	~~Bearing~~	~~Wire~~
2	Battery	Wire
~~3~~	~~Bearing~~	~~Mech. Conn.~~
~~4~~	~~Battery~~	~~Mech. Conn.~~

Node 2

	Symbol 2	Symbol 3
1	Wire	Motor
~~2~~	~~Wire~~	~~Pump~~
~~3~~	~~Mech. Conn.~~	~~Motor~~
~~4~~	~~Mech. Conn.~~	~~Pump~~

Figure 6: Three symbols from the conveyor system. Point terminals are exaggerated for clarity. The first table is the node describing the connection between symbols 1 and 2. The second is the node describing the connection between symbols 2 and 3. The tables show the possible interpretation pairs. The pairs crossed out with a solid line are removed by node consistency. The pair crossed out with a dotted line is removed by arc consistency.

some interpretation pair from that other node survived because of interpretation pair 3 in node 1. When interpretation pair 3 is later eliminated from node 1, the other node must be revisited. Changes to that other node may then propagate to yet other nodes that are connected by other arcs. Iterating until quiescence ensures that all of the consequences of the elimination of an interpretation pair have been considered.

If each node contains a single interpretation pair at the conclusion of node and arc consistency filtering, a unique, globally consistent solution has been identified. If multiple interpretation pairs remain in any node, it is necessary to perform search to identity any globally consistent solutions. We use exhaustive depth-first search for this purpose. To minimize cost, partial solutions with inconsistent interpretation pairs are pruned without need of further exploration. At each step in the search process, a choice of interpretation is selected for a symbol, from those choices remaining after filtering. Once a selection is made, every pair of connected terminals is checked for inconsistency. If a terminal belongs to a symbol that has not yet been interpreted, the consistency test succeeds, at least temporarily. If two symbols have been interpreted, any connections between them must have the same physical type, otherwise the solution is inconsistent. If any inconsistencies are detected, the partial solution is pruned, and no further expansions of the solution are considered. If all symbols have been interpreted and there are no inconsistencies, the solution is globally consistent and is saved. It is possible for there to be more than one globally consistent solution. If so, the program identifies them all.

Figure 7 shows the efficiency of the approach. The second column in the table contains the number of combinations of symbol interpretations. This is the product of the number of interpretations for each individual symbol. The third column is the product of the number of interpretation pairs for each pair of connected symbols. (An interpretation pair is one choice of interpretation for each symbol in a pair of connected symbols.) This value is typically much larger than the value in the previous column, because many combinations of interpretation pairs would result in a given component having multiple interpretations. For example, from the perspective of one pair of components, a particular component might be a wire, while from the perspective of some other pair, it might be a mechanical connector. (These kinds of inconsistencies are eliminated by arc consistency filtering.)

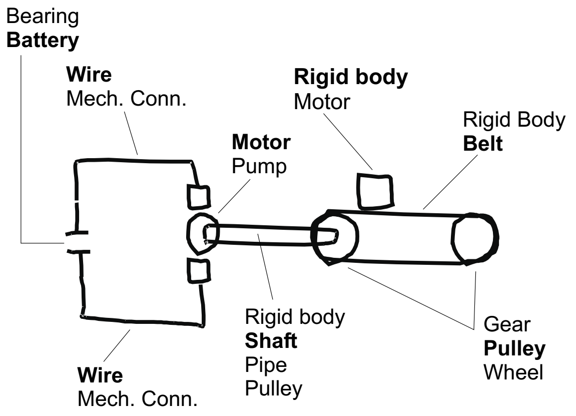
Initially, there are 2,304 possible combinations of interpretations and 3,538,944 combinations of interpretation pairs. After applying the node consistency filter, these numbers are reduced substantially to 144 and 1,296. Finally, after applying the arc consistency filter, both of these numbers are further reduced to 1. Thus, there is no need to use depth-first search in this case. The final, unique interpretation is shown in Figure 7.

The results of this example are typical. Although the number of possible interpretations for a sketch is exponential in the number of symbols, node and arc consistency filtering typical reduce the problem to a very manageable size.

Qualitative Simulation

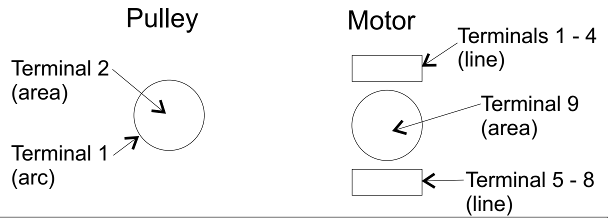
In the example of Figure 7 there was a single, globally consistent interpretation for the sketch. Sometimes, however, there are multiple such solutions. When this occurs, our program uses qualitative simulation to identify which interpretation was the intended one. The user annotates the sketch to indicate the intended behavior of a symbol. A simple qualitative simulator is then used to determine which of the global interpretations is capable of producing the indicated behavior. Currently, the annotations are limited to arrows indicating motion. A straight arrow is used to indicate translation, while a curved one is used to indicate rotation. To annotate a particular symbol, an arrow is drawn so as to intersect that symbol. Each arrow annotates a single symbol.

Each interpretation of a symbol has one or more qualitative models. These models consist of simple qualitative influences between the input terminals and the output terminals. The influences describe how energy flows through a physical component. Figure 8 shows examples of two models. The first is a model of a pulley. Terminal 2 is the input and terminal 1 is the output. The model states that if rotation



	Combinations of Interpretations	Combinations of Interp. Pairs
Initial	2,304	3,538,944
After node consistency	144	1,296
After arc consistency	1	1

Figure 7: Physical compatibility analysis of a conveyor system. All possible choices of interpretation are listed for each symbol, with the actual interpretations shown in bold.



```
(defrule sym4_pulley_1
  (is_a ?x symbol4)
  (interp ?x pulley)
  (flow T2 ?x rot)
  =>
  (assert (flow T1 ?x trans))
```

```
(defrule sym13_motor_1
  (is_a ?x symbol13)
  (interp ?x motor)
  (flow ?ta ?x elec)
  (flow ?tb&~?ta ?x elec)
  =>
  (assert (flow T9 ?x rot))
```

Figure 8: Examples of qualitative behavior models.

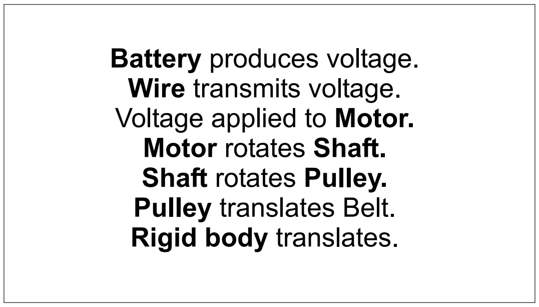
is applied to terminal 2, terminal 1 will produce translation. This model describes behavior in which rotation of the pulley causes a belt or rope to translate. There is also a model describing the converse behavior in which pulling a rope or moving a belt causes the pulley to rotate. The second model shown in the figure is that of an electric motor. It states that if electricity is applied to two different input terminals, the output of terminal 9 will be rotation.

The simulation is implemented with CLIPS, a forward chaining rule-based system (Giarratano 1998). Each qualitative model is a CLIPS rule. The simulation begins at an energy source. Possible energy sources include batteries and springs. These sources produce output without requiring any input. At the start of a simulation, one or more energy sources is selected as active. If a battery is active, assertions are made indicating that its output terminals provide electricity. Similarly, if a spring is active, assertions are made indicating that its output terminals produce translation. Once the initial assertions are made, forward rule chaining is used to continue the simulation. If a flow of electricity, translational motion, or rotational motion is present at a terminal, rules propagate that flow to any other terminals connected to it. Once there are flows at all of the input terminals of a component, a simulation rule fires, producing flows at the output terminals. The process continues in this fashion until no more rules can be triggered, at which point the simulation is complete.

Note that each component can have multiple alternative simulation rules, each describing a different type of behavior. For example, a pulley has one rule describing the behavior in which rotation of the pulley causes translation of a rope, and a second rule describing the converse behavior in which translation of a rope causes the pulley to rotate. The propagation of flows by the rule chainer naturally selects the correct rule for each component.

Our simulator is simple. For example, it does not distinguish between directions of motion, directions of forces, and signs of voltages. We have found that even this simple form of simulation is often adequate for our purposes. Our task is not to predict the detailed behavior, but to decide which of the consistent interpretations best matches the behaviors indicated by the user. For the types of schematic sketches we are considering, selecting the correct interpretation often reduces to selecting the correct type of component behavior from grossly different choices. For example, the question might be whether a particular component is acting as a motor or as a generator. Distinguishing between these two choices requires determining if electricity causes motion or vice versa. This sort of question can often be resolved without considering the direction of motion or the magnitude of the voltage.

Each simulation rule generates a description of the behavior it represents. As rules fire, their descriptions are concatenated to produce an explanation of how the device works. Because the descriptions are produced in parallel with the simulation, the device behavior is explained in the order determined by the flow of energy in the device. Figure 9 shows a typical explanation from the program.



Battery produces voltage.
Wire transmits voltage.
Voltage applied to **Motor.**
Motor rotates **Shaft.**
Shaft rotates **Pulley.**
Pulley translates Belt.
Rigid body translates.

Figure 9: Computer generated explanation of the conveyor system sketch.

Conclusion

We have described an approach for interpreting schematic sketches of physical devices. The approach relies on both geometric and physical reasoning. A symbol recognizer (geometric reasoner) is used to recognize the individual symbols in a sketch. Physical reasoning is then used to identify the meaning of each of the symbols. Part of the reasoning involves identifying which interpretations of the symbols would produce physically meaningful connections between the symbols. For example, it makes sense for a motor to be connected to a shaft but not to a pipe. We treat this compatibility problem as a constraint satisfaction problem (CSP). The number of potential interpretations of a sketch is exponential in the number of interpretations. This number is reduced to a very tractable size by the use of node and arc consistency filters. If the solution of the CSP reveals multiple globally consistent interpretations, qualitative simulation is used to identify the intended one. The sketch is annotated to indicate the intended behavior of one or more components, and the simulation identifies the interpretation that could produce that behavior.

Freehand sketches, by their very nature, contain a variety of inaccuracies and ambiguities. This work has demonstrated that these defects can often be overcome by reasoning about the overall context of the sketch, rather than examining each individual part of the sketch in isolation. This work has also demonstrated that for schematic sketches of physical devices, even simple physical reasoning is adequate for understanding the context and overcoming ambiguities.

References

Calhoun, C.; Stahovich, T. F.; Kurtoglu, T.; and Kara, L. B. 2002. Recognizing multi-stroke symbols. In *AAAI 2002 Spring Symposium Series, Sketch Understanding*.

De Bonet, J. S. 1995. Reconstructing rectangular polyhedra from hand-drawn sketches. `http://www.ai.mit.edu/~jsd/Projects/SketchReconstruction/sketch.html`.

de Kleer, J. 1979. *Causal and Teleological Reasoning in Circuit Recognition.* Ph.D. Dissertation, Massachusetts Institute of Technology.

Eggli, L.; Hsu, C.; Elber, G.; and Bruderlin, B. 1997. Inferring 3D models from freehand sketches and constraints. *Computer Aided Design* 29(2):101–112.

Falkenhainer, B., and Forbus, K. D. 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.

Fonseca, M. J., and Jorge, J. A. 2000. Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In *Proceedings of the 9th Int. Conference on Fuzzy Systems (FUZZ-IEEE 2000)*.

Giarratano, J. C. 1998. CLIPS user's guide. http://www.ghg.net/clips/CLIPS.html.

Kuipers, B. 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge.* MIT Press.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

Mashburn, T. A., and Anderson, D. C. 1994. Automatically deriving behavior constraints for performance variables in mechanical design. *Research in Engineering Design* 6:85–102.

Narayanan, N. H.; Suwa, M.; and Motoda, H. 1994. How things appear to work: Predicting behaviors from device diagrams. In *1994 International Joint Conference on Artificial Intelligence*, 1161–1167.

Nayak, P. P.; Joskowicz, L.; and Addanki, S. 1992. Automated model selection using context-dependent behaviors. In *Proceedings AAAI-92*, 710–716.

Platin, B. E.; Caliskan, M.; and Ozguven, H. N. 1991. *Dynamics of Engineering Systems, Lecture Notes*. Ankara, Turkey: Middle East Technical University, Mechanical Engineering Department Publications.

Rubine, D. 1991. Specifying gestures by example. *Computer Graphics* 25:329–337.

Shpitalni, M., and Lipson, H. 1996. Identification of faces in a 2D line drawing projection of a wireframe object. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18(10):1000–1012.

Stahovich, T. F.; Davis, R.; and Shrobe, H. 1998. Generating multiple new designs from a sketch. *Artificial Intelligence* 104(1–2):211–264.

Suwa, M., and Tversky, B. 1997. What architects and students perceive in their sketches: A protocol analysis. *Design Studies* 18:385–403.

Waltz, D. 1975. Understanding line drawings of scenes with shadows. In Winston, P. H., ed., *Psychology of Computer Vision*. Cambridge, MA: MIT Press.

Williams, B. C. 1984. Qualitative analysis of mos circuits. *Artificial Intelligence* 24(1–3):281–346.