Metamouse: Specifying Graphical Procedures by Example

David L. Maulsby, Ian H. Witten, Kenneth A. Kittlitz

Knowledge Sciences Laboratory, Department of Computer Science
The University of Calgary, 2500 University Drive NW
Calgary, Canada T2N 1N4

Abstract

Metamouse is a device enabling the user of a drawing program to specify graphical procedures by supplying example execution traces. The user manipulates objects directly on the screen, creating graphical tools where necessary to help make constraints explicit; the system records the sequence of actions and induces a procedure. Generalization is used both to identify the key features of individual program steps, disregarding coincidental events; and to connect the steps into a program graph, creating loops and conditional branches as appropriate. Metamouse operates within a 2D click-and-drag drafting package, and incorporates a strong model of the relative importance of different types of graphical constraint. Close attention is paid to user interface aspects, and Metamouse helps the user by predicting and performing actions, thus reducing the tedium of repetitive graphical editing tasks.

CR Categories

I.2.2 [Artificial Intelligence] Automatic Programming – program synthesis; I.2.6 Learning – knowledge acquisition; I.3.6 [Computer Graphics] Methodology – interaction techniques.

Other Keywords and Phrases

Geometric constraints, apprenticeship learning.

1 Introduction

Aesthetically pleasing, visually coherent, meaningful pictures are characterized by the spatial relationships that join components, suggest relative importance, lead the eye through a visual narrative, and reveal subtle connections. These relationships are called "constraints." Often they compete with each other and must be considered as a group, called a "constraint system." With or without the help of a computer, a graphic artist must manage constraints that may be complex and require compromise or careful ordering to be resolved. A drawing evolves as new objects and constraints are added and as some attributes and constraints change while others remain in force. These elements often interact; for example, changing a text font may require enlarging and re-positioning boxes in a flowchart. Despite the features provided by interactive graphics editors to automate constraints, editing still involves repetitive manual work that requires precision and planning.

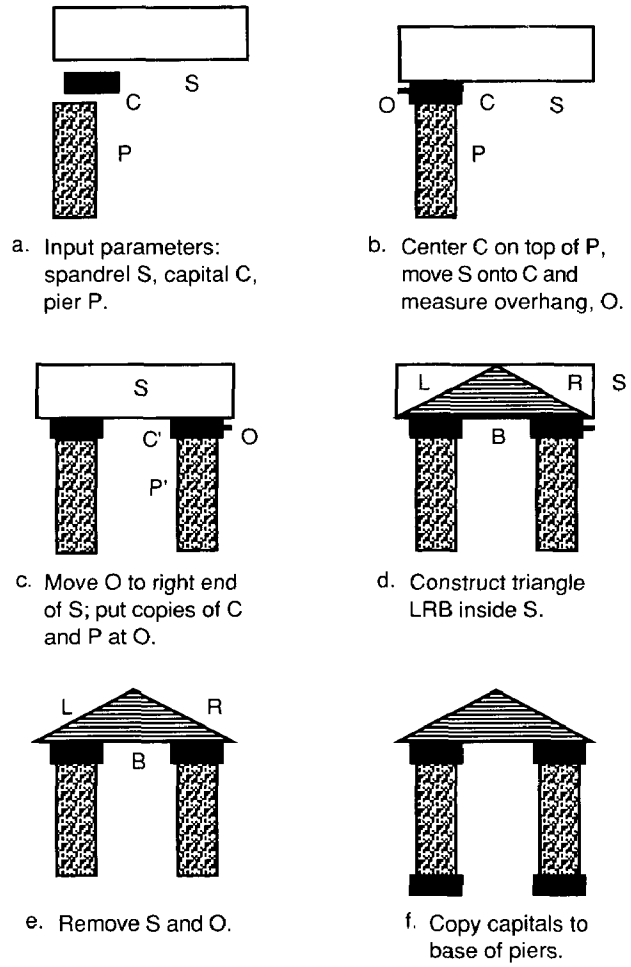


Figure 1. Constructing an arch from a rough sketch.

This paper describes a system that induces picture-editing procedures from execution traces. It observes the user at work, performs a localized analysis of changes in spatial relations to isolate constraints, and matches action sequences to build a state graph that contains conditional branches and loops. Moreover, it induces variables for objects and distinguishes constants from non-deterministic (ie. run-time input) parameters. The system includes a constraint solver to perform the actions it has learned.

©1989 ACM-0-89791-312-4/89/007/0127 $00.75

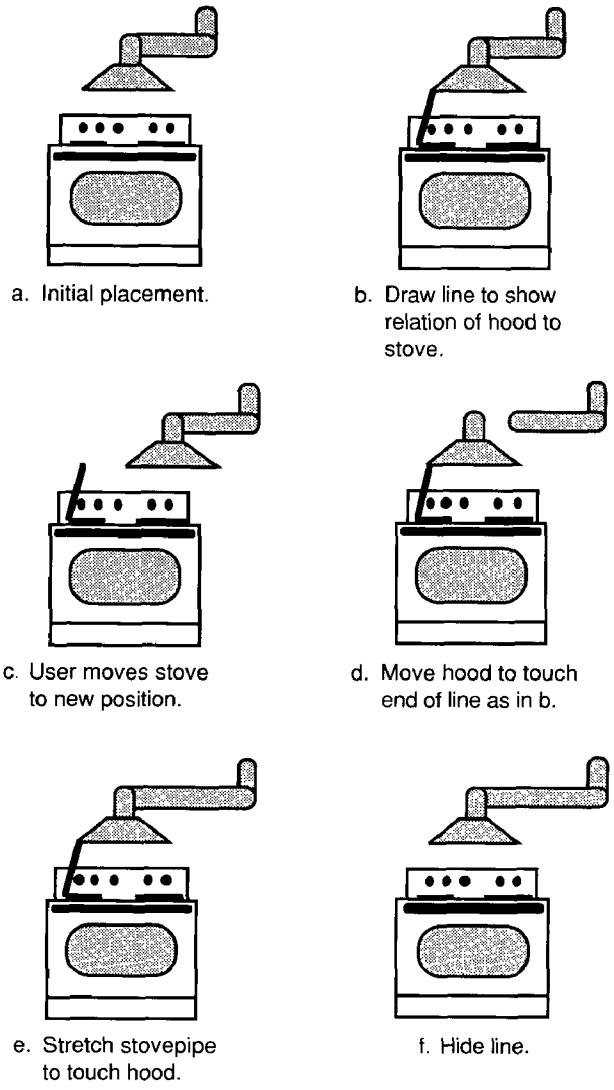


Figure 2. Maintaining constraints amongst objects.

A key component of the system is its metaphorical apprentice, Metamouse, an icon that follows the user's actions and represents the focus of attention. The metaphor embodies the system's limited model of spatial relations: Metamouse is near-sighted but touch sensitive. The user understands that relations at a distance must be constructed, for example by using a line to demonstrate alignment.

Section 2 describes some example editing tasks that can be taught to Metamouse. Section 3 discusses research issues and related work. Section 4 describes the current implementation. This is followed, in Section 5, by an evaluation of its performance on the sample tasks.

2 Applications

Several types of graphical task are appropriate for automation by a system such as Metamouse. Primary problems for users are 1) achieving precision, 2) maintaining integrity of constraints throughout the editing process, and 3) coping with the tedium of repetition. Examples of each type of task are described in detail below.

Figure 1 illustrates the construction of an arch from a sketch of its main components. Initially the artist draws one of the piers, a capital, and the spandrel's extents box. The capital is then centered over the pier and the spandrel box is moved down onto it with the desired lateral overhang. After measuring the overhang, the artist duplicates the pier and capital at the other end of the arch. A triangular spandrel is then constructed inside the box. Finally, plinths are added to the base of the piers. This editing sequence specifies a procedure for constructing a type of arch from four graphical inputs: pier, capital, spandrel box, and overhang. The construction requires precision but need only be done once.

Figure 2 illustrates constraints that must be maintained throughout the long-term editing of a picture. If the stove is moved, the ventilation hood must be re-positioned above the burners, and the stove-pipe must be stretched or shortened to reach the hood from the wall exit. The editing sequence proceeds as follows. First, the user expresses the constraint between burners and hood by drawing a tie-line between them. The user then moves the stove to its new position. The constraints are re-established as follows. The tie is moved to touch the burner as before. The hood is moved to contact the tie, and the stove-pipe is stretched to the hood. Finally, the tie-line is removed (or hidden). This task illustrates the use of an auxiliary object (the tie) to express a constraint. Other constraints stem from the role that touches play in terminating actions. This procedure could be invoked manually whenever the stove is moved, but it would be desirable to "attach" it to the operation of moving the stove, which would automatically trigger it.

Figure 3 illustrates a repetitive editing operation or an animation sequence. A teapot moves up and down rows of cups laid out on a buffet, filling them with tea, and then returns to its initial position. Since cups are not perfectly aligned, a row is defined by a line passing through the center of one cup and touching the others. A procedure looping on rows and cups would allow us to change these numbers without re-scripting. Moreover, a constraint-oriented description of the teapot's path (eg. "move rightward to next cup" rather than "move to (x, y)") would tolerate adjustments to the layout. A program for this task is shown in Figure 4.

The teapot's initial position is marked with a slash and the pot moves to the table's nearest corner. For each iteration of the main loop, the row-line advances upwards to the center of some cup — a sweep-selection method [19]. The pot moves to the row-line's near end. For each cycle of the inner loop, the pot advances to meet the following constraints: i) the spout is at the center of some cup, *C*; ii) *C* is touching the row-line; and iii) *C* was not already visited. At the end of a row, some of these constraints fail. Note that the constraints ensure that the pot moves in opposite directions in successive rows. The main loop ends when no cups remain in the row-line's upward path. The teapot then returns to its initial position marker via the buffet's perimeter and, finally, the marker is removed.

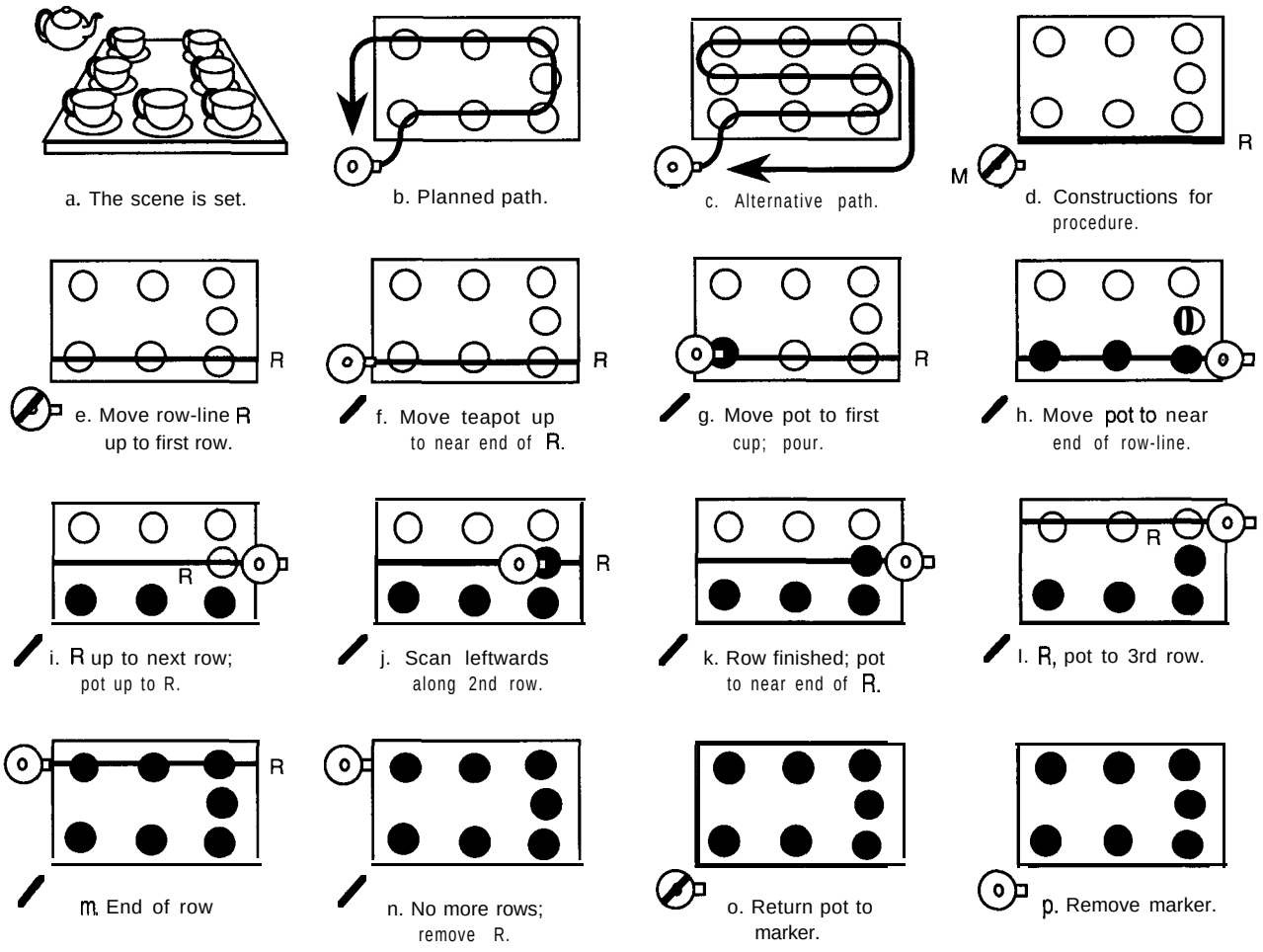


Figure 3. The tea-party animation procedure.

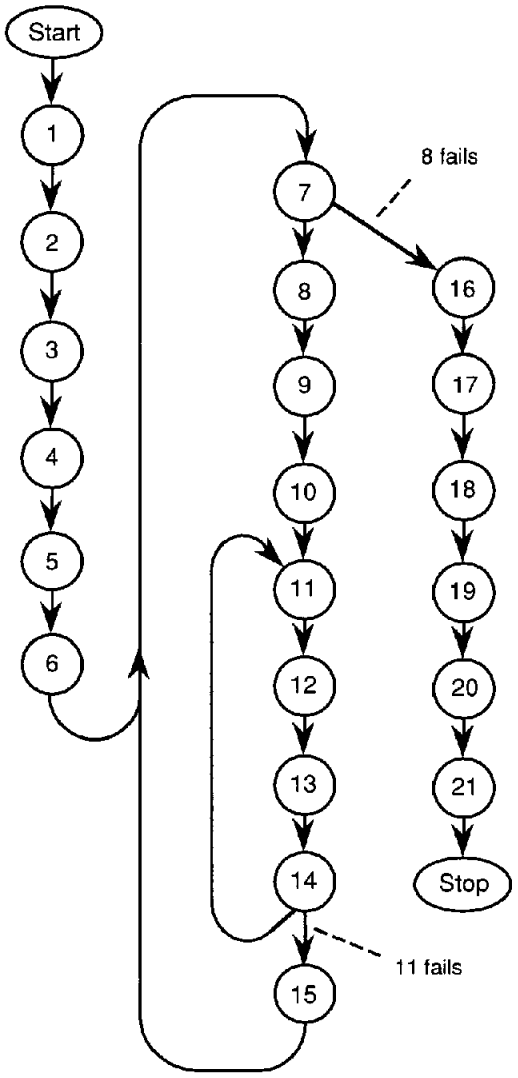
The power of a system like Metamouse lies in its ability to isolate constraints and predict actions. The user performs only a few steps of the tea-party task. Once Metamouse detects repetition, it predicts subsequent actions until it cannot meet the constraints or until the user objects. It observes the teapot move to the second cup and predicts all actions for the rest of row 1. When it fails to find a fourth cup, it asks the user to take over. The user moves the row-line; Metamouse recognizes this action and hence predicts the move-and-pour sequences for the second and third rows.

3 Background and Related Work

Automation of graphical editing tasks has followed two streams of development: interactive tools to help users with constraints; and graphics-oriented programming systems. Interactive help began with Sketchpad [22], which used iterative numerical relaxation to resolve several types of constraint among object parameters. A similar approach is adopted in [27], which lets users compose constraints based on least-squares relaxation. Recent research has also produced a system that automatically selects and applies appropriate construction tools [4]. These systems offer simple, appealing interfaces to a restricted set of constraint-satisfaction methods.

End-user programming is one way to support repetitive, customized editing operations and the invention of arbitrary constraint systems. Given that most users are non-programmers, research has focussed on graphical methods, often based on geometric construction [5, 7, 8, 17]. With their graphical interfaces and use of examples, these systems greatly simplify program construction, but users must still work with abstractions. When programming with L.E.G.O. or a macro facility such as [24], the user declares loops and conditional branches, albeit by menu selection. Users of ThingLab must conceive an algebraic model of constraints in order to produce equational networks that define them [5].

An alternative is to observe the user at work and infer loops and branches, constants and variables. A number of systems for programming by demonstration have been produced [2, 9, 11, 16, 21]. Programs are constructed incrementally from several execution traces. Only Noddy, a robot teaching system, relies completely on automatic generalization [2], but it performs an exponentially complex induction of functions and is incapable of coping with errors. SmallStar [9] operates in a very general desktop domain but requires the user to identify variables and their type and value range. Peridot [16] infers value ranges and certain spatial relations (such as "centered within box"), but not loops or branches.



1. move to touch (Table.bottom.left)
2. draw-line R to touch (Table.bottom.right)
3. move to touch (Pot.top.right)
4. draw-line M to touch (Pot.bottom.left)
5. move to grasp (Pot.center)
6. drag Pot to touch (Pot.center : Table.bottom.left)
7. move to grasp (R.midpt)
8. drag R upwards to touch (R.line : F.center),
   where F is first cup found by scanning upwards
9. move to grasp (Pot.center)
10. drag Pot upwards to touch (Pot.center : R.endpt)
11. drag Pot horizontally to touch (Pot.center : C.center),
    where C is first cup found by scanning horizontally
12. move to touch (C.top.right)
13. draw-line to touch (C.bottom.left) ; ie. pour tea!
14. move to grasp (Pot.center)
15. drag Pot to touch (Pot.center : R.endpt)
16. delete R
17. move to grasp (Pot.center)
18. drag Pot to touch (Pot.center : Table.bottom)
19. drag Pot to touch (Pot.center : M.midpt)
20. move to grasp (M.midpt)
21. delete M

Figure 4. Procedure learned for tea-party animation.

Inferring a program is not easy, but induction of complex picture transformations from examples of input and output is intractable [3]. Moreover, systems of equations to represent these transformations would be numerically unstable and difficult to solve. Thus, it is better to induce a sequence of simpler transformations. Drawing is inherently procedural, often systematically ordered with each step governed by very few constraints [26]. Nonetheless, it is hard to induce procedures even from simple steps. Typical users do not always construct (or know how to construct) the relevant measurements and relations, but work instead by visual inspection. In effect, their drawings include invisible objects, as illustrated in Figure 5. Curve-matching methods such as those employed in graphical search and replace [10] are not sufficient for inducing patterns in traces that contain invisible objects. On the other hand, examining the screen for implicit spatial relations clearly involves an enormous amount of search and vastly expands the space of hypotheses for generalization. Therefore the system should isolate a small neighborhood of attention, and restrict itself to explicit relations of touch. It follows that user must specify these constructively.

To worsen matters, a preliminary study of MacDraw users performing a set of graphical tasks [13] revealed that execution traces are riddled with extraneous and erroneous actions. Users not only made mistakes, but were observed performing experiments or simply fidgeting. The order of actions varied greatly within the first several iterations of loops.

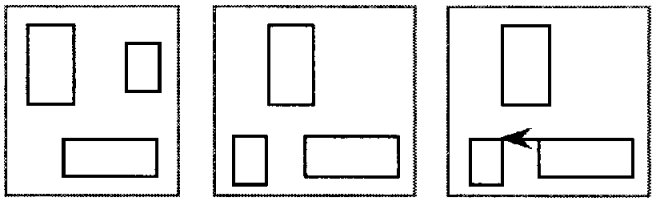


Figure 5. An invisible object as a constraint.

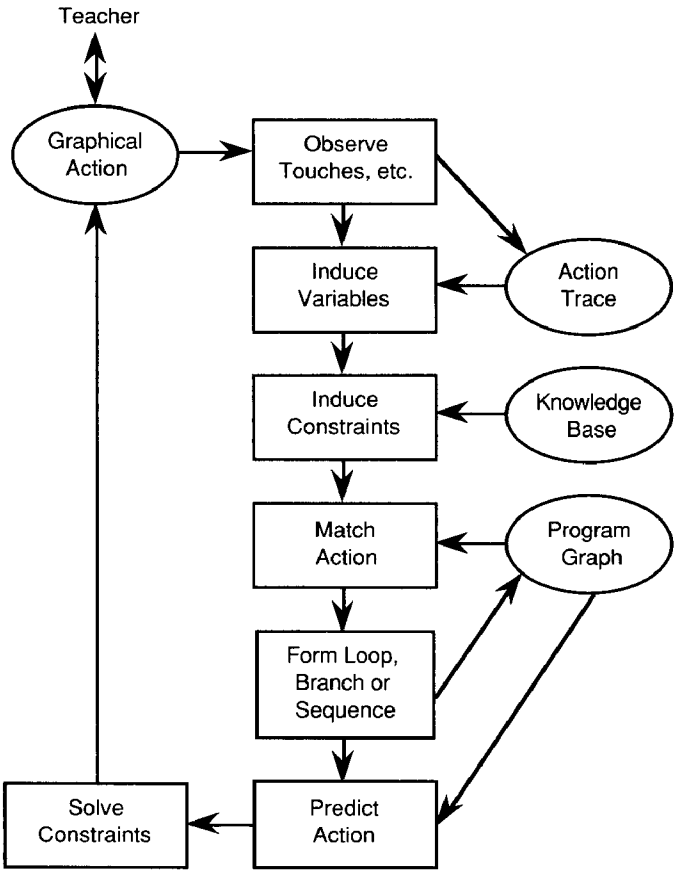


Figure 6. Main components and data flow of system.

These elicitation problems are well-known in human-human communication, and rules of interaction between human teachers and pupils have been formulated as "felicity conditions" [25], four of which apply when inducing graphical procedures: *correctness*, *show-work* (demonstrate execution rather than just input and output), *no-invisible-objects* (express constraints by graphical construction), and *focus-activity* (eliminate extraneous actions).

These conditions are difficult for untrained teachers to satisfy. The Metamouse system uses a metaphorical apprentice, intensive interaction, and generalization to help the teacher. The Metamouse is the system's focus of attention; only touch relations involving it or an object it is grasping are examined. The system tries possible generalizations and predicts actions as early as possible during a teaching session, to eliminate free variation and extraneous actions and also to reduce errors. It can learn alternative actions and re-order their precedence in order to overcome errors. It has an internal model of graphical constraints and asks for explanation when an action seems arbitrary, ie. insufficiently constrained. The metaphor encourages the teacher to demonstrate constraints and adopt an intentional stance toward the system [6] rather than understand the details of its constraint and generalization models. Whether or not the metaphor succeeds is an experimental question; some pilot tests have yielded encouraging results [14].

4 The Metamouse System

Our learning system works within an interactive 2D graphics editor. "Teaching mode" is distinguished from normal editing only by the presence of the Metamouse icon: there are no special programming commands except to start and stop Metamouse. The flow of data through the system is sketched in Figure 6. When the teacher performs a drawing operation, the system records it and augments it with explanatory features, matching objects with variables and identifying probable constraints on the cursor's new position. Metamouse moves to the point at which the action terminated and highlights object parts involved in constraints. The augmented action is then matched with program steps previously learned. If a match is found, the learning module may conjecture a loop or joining of branches. It then predicts subsequent actions to confirm this. Predicted actions are performed by a constraint solver. Metamouse autonomously moves and highlights objects, and continues to do so until the teacher rejects a prediction or the constraint solver fails.

The next two subsections give brief accounts of the graphics application and the Metamouse interface. Following that we examine individual modules of the learning system.

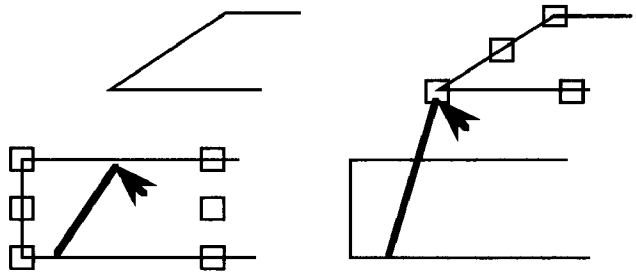


Figure 7. Highlighting distinguished points near cursor (arrowhead) while rubber-banding a line.

4.1 A.Sq

Constraints are easier to identify and resolve if primitive operations have few degrees of freedom. Thus a drawing program with a point-and-click user interface, like MacDraw [12], is suitable. Our drawing program, A.Sq (after the protagonist of Flatland [1]), emulates MacDraw but at present includes only box and line primitives. The user draws and transforms primitive objects by moving iconic handles (as in MacDraw). These handles delimit parts of objects distinguished by the learning system. They appear whenever the cursor (or Metamouse) approaches them, as illustrated in Figure 7.

The choice of primitives and operators has a great impact on the user's expression of constraints. Languages such as L.E.G.O. [8] and the primitives of [17] provide a basis for traditional "ruler-and-compass" methods of construction. A.Sq's primitive object types P, auxiliary objects A, modes of operation M, user-interface commands U, and internal operators I, are summarized in Table 1 below.

At present, the drawing program is relatively simple yet rich enough to study programming-by-example issues. No conceptual difficulties are envisaged in extending the learning system to cope with new primitives such as points, polygons, ellipses, and splines, since only an object's distinguished parts have any significance. We also expect to be able to accommodate new operations such as rotation, grouping, and coloring.

P : box (bottom-left, top-right)
line (endpt$_1$, endpt$_2$)
point (x, y)
action (operator, startpt, endpt, object)

A : CurrentPoint : point
PreviousPoint : point
CurrentObject : {box, line}
DisplayList : list of {box, line}
ActionList : list of actions

M : create-lines
create-boxes
transform-objects

U : set-mode (mode ∈ M)
set-point (PreviousPoint, CurrentPoint, x, y)
delete-object (CurrentObject)
undo (ActionList)

I : create-line (CurrentObject, PreviousPoint, CurrentPoint, DisplayList)
create-box (CurrentObject, PreviousPoint, CurrentPoint, DisplayList)
translate-handle-of-object-to-point (handle, CurrentObject, CurrentPoint)

Table 1. Elements of the A.Sq drawing program.

4.2 Metamouse

The focus of the teacher's attention is the Metamouse, Basil, a graphical turtle in the tradition of [18]. Prior to working with Basil, teachers skim a bio-sheet, excerpted in Figure 8. When the user of A.Sq senses an opportunity to automate a task, she calls Basil from his den. Rather than follow the cursor continuously, the turtle icon moves to *CurrentPoint* at the closure of each A.Sq operation. If the system finds no tactile constraint, Basil asks the user whether position or distance are inputs, constants, or should have been constructed. Should the teacher suspend recording temporarily, Basil withdraws into his shell. In all other respects the A.Sq commands operate as usual.

4.3 Touch Relations

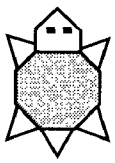
Metamouse is described as near-sighted but touch-sensitive; the teacher understands that only touch relations involving Basil or an object in his grasp are analyzed for constraints. The system highlights relevant parts of objects, as illustrated in Figure 9. Touches considered important (see Section 4.5 below) are colored red, others yellow. Associated with each touch is a triangular button; selecting this toggles it from red to yellow or *vice versa*, so the teacher can override the system's decision.

A touch relation is defined as *touch* $(Object_1.Part_1 : Object_2.Part_2)$, where $Part_i$ indicates some part of $Object_i$. Distinguished parts are handles (vertices, mid-points) and the line segments between them.

4.4 Variables

The use of variables allows different objects to assume a particular role in successive iterations of actions [20]. The learning system substitutes variables for objects in touch relations. Variables are defined as *variable-definition (Name, Type, Value)* structures maintained in a global symbol table. *Type* is one of {box, line}.

References to variables in touch relations are defined as *variable-reference (Variable, Valuation-flag)* tuples, where Variable points to the definition, and Valuation-flag indicates whether the constraint solver should use the variable's current value or try to assign a new one. The variable inducer looks back through recent steps of the example action trace for previous occurrences of the object; if none is found, the Valuation-flag is set to indicate that the solver must search for it.



My name is Basil and as you can see I'm a turtle. You teach me repetitive and finicky tasks. I learn by acting as your apprentice — I follow you around till I think I know what you'll do next, then I do it for you.

If I guessed wrong I'll undo it and wait for you to show me what's right. I only predict after I see you do something you've already taught me.

I can draw lines and boxes and carry them by their iconic handles (grasping with my jaws).

Although I have a good memory, I don't see too well. Instead I work mainly by feel. I remember which parts — handles and line segments — are connected.

I'm touch-sensitive only at my snout but I can sense contact between what I'm grasping and anything else.

If I have to find, say a box, I set off in the general direction you've taught me (up, down, left, right) until I bump into one. But if you want me to be more selective, give me a tool to carry and teach me to move until it touches.

I can't learn directly how things should *not*touch — I mean how they should be separated. Instead you should give me tools to separate them.

When you want to teach me, choose "Time for a lesson!" from the Basil menu. If you want to interrupt the lesson say "Take a nap." When you don't agree with what I do, tap me and I'll undo it. When I don't know what to do I'll ask you to show me.

So in general you teach me by doing the task yourself, using some extra tools to help me see patterns by feel.

Hope you enjoy teaching me!

Figure 8. Excerpts from description of Metamouse given to teachers.

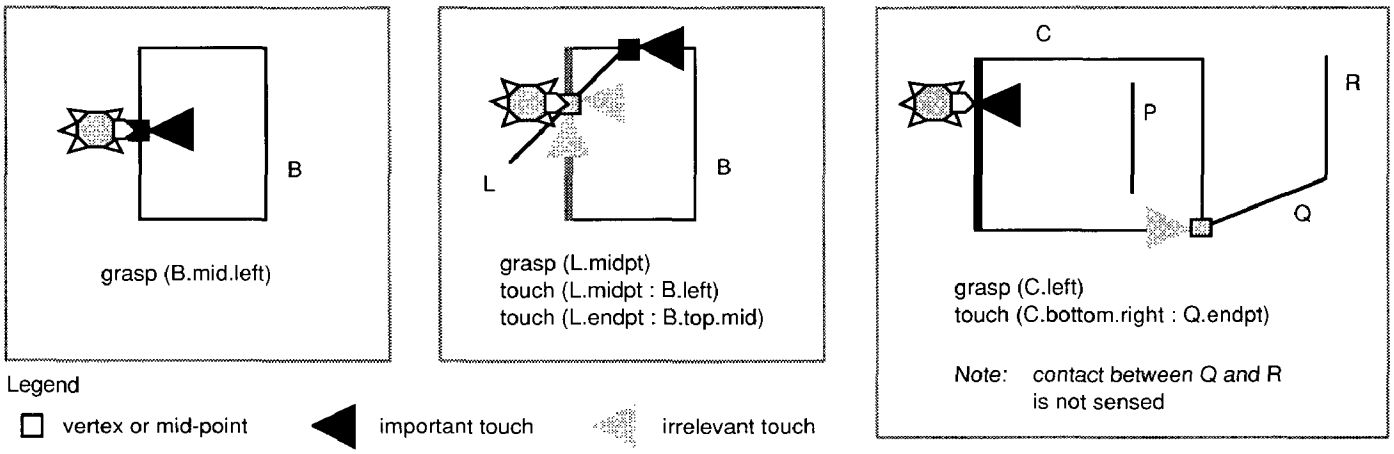
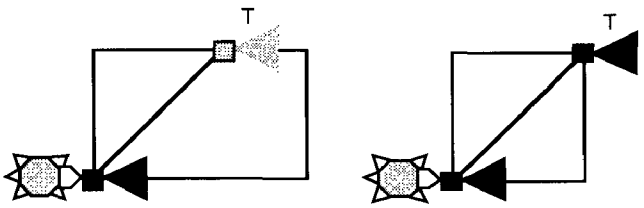


Figure 9. Feedback from Metamouse, highlighting touch relations.

4.5 Constraints

Isolating relevant constraints from the great many that hold in any given situation is, of necessity, a heuristic procedure. When searching for and transforming objects, the constraint solver is governed by touch relations and Basil's path of movement. The constraint inducer examines touch feedback obtained after each step of the trace. It weeds out trivial or irrelevant touches; the survivors comprise the postcondition of a program step.

Although all A.Sq drawing and transformation operators are based on translating *CurrentPoint* in 2-D display space, user actions occur in a model space containing objects with numerous parts, and in the "activity space" in which several alternative actions may be possible at any given time. Thus Basil operates with multiple degrees of freedom (and hence constraint) in the selection of actions and their parameters. Touch relations have 6 degrees of freedom, 3 on each item: selection of object, part, and position within part.



a. Contact T ignored as overdetermined.

b. Use of T to detect a square.

Figure 10. A useful overdetermined constraint.

A constraint is expressed as *constraint (Data, Class, Used)*, where *Data* is a touch relation, path, etc., *Class* is a degree of constraint, and *Used* is a flag indicating whether this constraint is deemed relevant. The constraint inducer assigns each touch relation to one of four classes. "Determining" constraints select a specific object, part and position within it for each item of a touch relation. For example, in Figure 3, contact between Basil's snout (a point) and the teapot's center handle is determining. "Strong" constraints leave one choice free, as in *touch(teapot-spout.endpt2 : C.center)*, where *C* is a cup found by scanning along the row. "Weak" relations leave more degrees of freedom. freedom. "Trivial" touches, like grasping the end-point of the row-line after it is drawn, follow from the definition of A.Sq operators and afford no constraint whatsoever.

If the classifier finds a determining constraint, it marks other touch relations as "overdetermined;" they are not needed to derive a new position for *CurrentPoint*. Such touches may help select an action. They can distinguish otherwise similar situations, as shown in Figure 10. Here, the touch between point T and the rectangle seems irrelevant, since contact between *Box.bottom.left* and Metamouse's snout determines the move. But *touch (Line.endpt : Box.top)* versus *touch (Line.endpt : Box.top.right)* distinguishes a rectangle from a square. The teacher clicks on the relation icon to change its status.

The classifier can detect a lack of sufficient constraint for viable solutions. If all touches are only strong or weak, Basil's path is made a weak constraint with the caveat that the solver may have to relax it. In case of a total lack of touch constraint, Basil asks the user to indicate which of a set of standard tacit constraints, such as input or constant position, applies to this action.

Determining, strong, and weak constraints are considered relevant; overdetermined, trivial and sustained touches are ignored. Such heuristic rules can provide at most a best guess as to the teacher's precise intentions. As mentioned in Section 4.3 and illustrated in Figure 10, the teacher can override Metamouse's decision to ignore a touch relation.

4.6 Actions and Procedures

The learning algorithm takes the linear sequence of steps performed by the teacher and builds a directed graph containing branches and loops. It operates interactively, so that each new step is integrated into the graph as it is performed. If the user's action matches an existing step, the system conjectures a link to that step. A link is verified by predicting its successors.

Each step is a *program-node (Predecessors, Action, Successors)* structure. Its location in the graph is expressed as lists of *Predecessors* and *Successors*. The *Action* is an *action-step (Preconditions, Operation, Postconditions)* tuple. *Operation* is one of the A.Sq operators. *Pre-* and *Postconditions* are lists of *constraints* that must hold before and after executing the action. *Preconditions* are checked by examining Basil's current sensory

state; *Postconditions* are instantiated and verified by the constraint solver.

A program step matches an action demonstrated by the teacher if the operator and constraints are the same. A program step can be generalized by dropping constraints in order to match an action. To avoid over-generalizing, Metamouse drops only weak constraints, like path, and remembers them in case they need to be enforced after all.

At every opportunity the system generates the next action itself. It checks *Successors* of the current node to see if any is executable in the present configuration — that is, its relevant *Preconditions* hold and its *Postconditions* are attainable via constraint satisfaction. If there is none, or if the teacher rejects the prediction, Metamouse asks for a demonstration of the next step and forms a branch to it.

4.7 Constraint Solver

Solving constraints is the process by which actions are predicted — both to test whether a step is performable and to generate specific parameters for the action. Since all A.Sq actions result from translating a single point, the solver is much simpler than most. It examines the list of *Postconditions* in order of strength. It generates a solution to the first and then checks that the rest hold. If not, it backtracks to an alternative solution for the first constraint. The process repeats until all constraints hold or no more solutions exist. Details appear in [15]. The constraint solver is potentially able to generalize the *Postconditions;* at present, only Basil's path is generalized.

5 Evaluation of System

A programming by example system should be evaluated with respect to ease of use, real-time performance, and the correctness and generality of the programs it infers. Ease of use has been tested by measuring potential teachers' ability to predict Basil's behavior [14]. It was found that teachers quickly learned what to expect from their pupil, apart from the occasional surprise.

Real-time performance is governed by the number of program steps and touch relations to be checked by the action matcher and constraint solver. The complexity of a prediction is proportional to the product of these numbers. The number of touch relations is normally quite small, due to the limited range of Basil's touch sense. At most, it is twice the number of object-parts, since every relation involves either Basil or the object in his grasp. The number of program steps is unbounded; hence it may be advisable to limit the search for a matching step.

The current experimental implementation in Lisp on a Macintosh II runs rather slowly, requiring two seconds to make a prediction when joining sequences (subsequent predictions are somewhat faster). Since this delay is invariant with the number of program steps, it is attributable to excessive and inefficient garbage collection.

Metamouse has been tried on a number of example tasks — aligning boxes to arbitrarily rotated axes, distributing boxes at equal spacing along a line, and re-connecting a polyline when one segment is moved. The system learned correct and sufficiently general programs for most of these simple tasks after just one trace. Some examples contained erroneous actions and bizarre coincidences, but errors in the programs were corrected during subsequent lessons.

Table 2 shows performance data for the three tasks presented in this paper. For each execution trace, the number of actions correctly predicted by Metamouse is compared with the total number performed (whether by the user or Metamouse). Task competence is measured as their ratio. The number of predictions rejected by the teacher is also shown. The size and growth of the program graphs is given as the number of edges (ie. transitions between actions).

The Arch task contained some repeated actions, such as asking the user to create three boxes and copying the two capitals to form two plinths. The former actions were distinguished by the prompt strings specified by the teacher after rejecting predicted prompts. The latter were differentiated because the capitals were known as individuals. In a second trace of Arch, the system correctly predicted all actions.

The Stove-hood task was a simple sequence of actions. No predictions could be made during the first trace since the task contains no repeated actions.

The Tea-party task is a two-dimensional iteration on cups within rows. Table 2 shows performance on each row during the first trace. The system was able to generalize direction and number so that covered the second and third rows. In the first row, the teacher moved the row-line into contact with the cups, then moved the teapot to the row-line's nearest end-point, and then rightward to the first cup. After marking the cup (that is, pouring the tea), the teacher advanced the pot to the next cup, which triggered predictions to mark it and repeat for the third. On attempting a fourth iteration, the constraint solver failed to find another cup in Metamouse's path; this failure was the loop's terminating condition. When the teacher advanced the row-line to the next row, the matcher generalized the contact constraints between row-line and cups, since row 2 contained only one cup. The system then successfully predicted that the teapot would advance and thus induced the outer loop. The direction of the teapot's movement along the row was generalized from rightward to horizontal in order to make subsequent predictions. The loops were now general enough to cover the third row.

Task		Steps Performed in Task				Edges in Program Graph	
	Trace #	Total	Predicted	Ratio	Rejected	Total	Growth
Arch	1	41	6	15 %	5	42	42
	2	41	41	100 %	0	42	0
Stove-hood	1	12	0	0 %	0	13	13
	2	12	12	100 %	0	13	0
Tea-party	1	57	34	60 %	5	24	24
	row 1	*18*	*7*	*40 %*	*3*	*7*	*7*
	row 2	*9*	*8*	*90 %*	*0*	*7*	*0*
	row 3	*18*	*18*	*100 %*	*0*	*7*	*0*
	2	65	65	100 %	1	25	1

Table 2. Performance of learning system on example tasks.

6 Further Work

The current implementation is unsuitable for much further research. It is too slow and unreliable; expansion of facilities will only exacerbate these problems. Hence we are re-implementing Metamouse in C++ on Apollo DN4500 workstations. When a prototype is ready, we will conduct studies with casual users. Graphics primitives such as circles, ellipses and splines are planned, as well as object rotation and grouping. Further desirable additions include the ordering of alternative predictions by generality or frequency, and a pattern-matching command to allow the user to specify a pattern without constructing a procedure.

The nature of Metamouse raises several important questions. The system is designed to build a predictive model of human performance by conjecturing intentions behind isolated actions. This focus of attention should be expanded to sequences so that the system might identify free variation on the order of actions, equivalence, ineffectiveness, and so on. Metamouse also facilitates rich interaction. Methods of eliciting constraints from the teacher should be compared with respect to the trade-offs between inductive generalization and explicit indication. Induction of some implicit spatial relations, such as alignment, is not infeasible. On the other hand, graphical gesturing, as in pointing to interesting touch relations, shows promise as a natural technique for teaching.

7 Conclusions

Metamouse demonstrates that it is indeed possible for users to create graphical procedures by direct manipulation. Applications range from producing complex, repetitive drawings, through constructively specifying figures governed by graphical constraint, to generating simple animated algorithms for tasks such as sorting (and pouring tea). Metamouse eagerly reveals its predictions as soon as it can. This has three advantages. First, users reap early benefits when performing repetitive operations. Second, they can correct errors as soon as they occur. Third, they develop confidence in their programs without ever viewing any kind of listing. The principal shortcomings of the current system are its limited repertoire of graphical objects and transformations, the lack of a formal underpinning for the constraint model, and our limited experience of how users react to the new experience of working with Metamouse.

Acknowledgements

This research is supported by the Natural Sciences and Engineering Research Council of Canada. We gratefully acknowledge the key role Bruce MacDonald has played in helping us to develop these ideas. We would also like to thank the referees for their helpful suggestions.

References

1. Abbott, Edwin A. *Flatland — A Romance of Many Dimensions*. Signet Classics edition. New York. 1984.
2. Andreae, Peter. "Justified generalization: acquiring procedures from examples." PhD thesis. Department of Electrical Engineering and Computer Science, MIT. January 1985.
3. Angluin, Dana and Smith, C. H. "Inductive inference: theory and methods." *Computing Surveys 3* (15), pp. 237-269. September 1983.
4. Bier, Eric A. and Stone, Maureen C. "Snap-dragging." Proc. ACM SIGGRAPH '86 (Dallas, August 18–22, 1986), in *Computer Graphics 20*, 4, pp. 233–240.
5. Borning, Alan. "Defining constraints graphically." *Human Factors in Computing Systems: Proc. ACM SIGCHI '86*. Boston. April 1986.
6. Dennett, Daniel C. *The Intentional Stance*. MIT Press. Cambridge MA. 1987.
7. Fuller, Norma and Prusinkiewicz, P. "L.E.G.O.—an interactive graphics system for teaching geometry and computer graphics." *Proc. CIPS Edmonton 1986*.
8. Fuller, Norma and Prusinkiewicz, P. "Geometric modeling with Euclidean constructions," in [23], pp. 379-391.
9. Halbert, Dan. "Programming by example." Research Report OSD-T8402. Xerox PARC. Palo Alto CA. December 1984.
10. Kurlander, David and Bier, Eric A. "Graphical search and replace." Proc. ACM SIGGRAPH '88 (Atlanta GA, August 1–5, 1988), in *Computer Graphics 22*, 4, pp. 113-120.
11. MacDonald, Bruce A. and Witten, Ian H. "Programming computer controlled systems by non-experts." *Proc. IEEE Systems, Man and Cybernetics Annual Conference*. Alexandria VA. October 1987.
12. Cutter, Mark, Halpern, B., Spiegel, J. MacDraw. Apple Computer Inc. 1985, 1987.
13. Maulsby, David. "Inducing procedures interactively." MSc thesis. Department of Computer Science, University of Calgary. December 1988.
14. Maulsby, David and Witten, Ian H. "Inducing procedures in a direct-manipulation environment." *Proc. ACM SIGCHI '89* (in press).
15. Maulsby, David, Kittlitz, Ken and Witten, Ian H. "Constraint-solving in interactive graphics—a user-friendly approach." *Proc. Computer Graphics International 1989* (in press).
16. Myers, Brad. *Creating User Interfaces by Demonstration*. Academic Press. San Diego. 1988.
17. Noma, T., Kunii, T. L., Kin, N., Enomoto, H., Aso, E. and Yamamoto, T. Y. "Drawing input through geometrical constructions: specification and applications," in [23], pp. 403-415.
18. Papert, Seymour. *Mindstorms*. Basic Books. New York. 1980.
19. Preparata, Franco P. and Shamos, Michael I. *Computational Geometry*. Springer-Verlag. New York. 1985.
20. Rich, Charles and Waters, Richard. "The programmer's apprentice: a research overview." *IEEE Computer 21* (11), pp. 11–25. November 1988.
21. Smith, David C. "Pygmalion: a creative programming environment." Report STAN-CS-75-499. Stanford U. 1975.
22. Sutherland, Ivan E. "Sketchpad: a man-machine graphical communication system." *Proc. AFIPS Spring Joint Computer Conference*, vol. 23, pp. 329-246. 1963.

23. Magnenat-Thalmann, Nadia and Thalmann, Daniel, eds. *New Trends in Computer Graphics: Proc. CG International '88*. Geneva. June 1988.

24. Tempo. Affinity MicroSystems Ltd. Boulder CO. 1986.

25. van Lehn, Kurt. "Felicity conditions for human skill acquisition: validating an AI-based theory." Research Report CIS-21. Xerox PARC. Palo Alto CA. 1983.

26. van Sommers, Peter. *Drawing and Cognition*. Cambridge Univ. Press. Cambridge UK. 1984.

27. White, R. M. "Applying direct manipulation to geometric construction systems." in [23], pp. 446-455.