INDUCING PROGRAMS IN A DIRECT-MANIPULATION ENVIRONMENT

David L. Maulsby and Ian H. Witten

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

Phone: (403) 220-7683
Email: .halbertalcalgaryimaubslby, lnn@epsc.ucalgary.ca

Abstract

End users who need to program within highly interactive direct-manipulation interfaces should be able to communicate their intentions through concrete demonstration rather than in terms of symbolic abstractions. This paper describes a system that learns procedures in interactive graphics taught to it "by example" by minimally trained users. It shows how techniques of machine learning and reactive interfaces can support one another - the former providing generalization heuristics to identify constraints implicit in user actions, the latter offering immediate feedback to help the user clarify hidden constraints and correct mistakes before they are planted into the procedure. The teacher's attention is focused on the learning system's perceptual and inferential shortcomings through a metaphorical apprentor called MetaMouse, which generalizes action sequences on the fly and capacity, carries out any actions it can predict. The success of the induction process is measured quantitatively by counting erroneous predictions made during example tasks.

Topic area: Interface Designs or Interface Design Tools and Techniques

Keywords: end user programming, programming by example, induction

Total words: 4000 (including abstract and references)
Graphics: 6
Tables: 1
Introduction

A serious shortcoming of current highly-interactive iconic computer systems is their failure to supply a natural way for end users to create programs within the user interface. This shortcoming is inherent in the literal language of such interfaces; programs must contain abstractions of objects and relations. One way of introducing abstraction is to annotate action traces with the desired generalized operators [Halbert 84]. Another is to use inductive generalization [MacDonald 87], combatting its inherent unreliability and computational intractability [Angluin 83] with heuristics. The system presented here brings interaction to bear upon the problem, so that inductive generalization, which necessarily involves extensive search [Mitchell 82], is highly focussed. The system learns generalized procedures by observing and cooperating with the user at work.

Our test domain is interactive computer graphics— a MacDraw-style drafting utility. To be useful, graphical demonstrations must be abstracted and generalized into procedures with variables, constants, conditionals, and loops. Although most clearly expressed in formal language, program constructs can be induced from concrete examples. Previous systems for programming by example addressed the problems of induction by requiring the user to augment examples with symbolic annotation ([Borning 86], [Halbert 84], [Smith 75]). A number of projects have tackled the problem of inducing functions [eg. Andreea 85]. A recent pilot experiment relied upon a teaching metaphor [MacDonald 87]. The work described here expands upon this latter idea.

Three elements of interaction are essential. First, a metaphorical apprentice, called Metamouse because it follows the teacher’s mouse as in [Myers 87], embodies the learning system’s task model, including limits on its ability to make generalizations. The user, as teacher, understands that she must express actions at a finer level of granularity by graphically constructing measurements normally done “by eye.” This extra information limits the search for generalizations but is expressed without abandoning the drawing program’s direct-manipulation interface.

Second, Metamouse demonstrates what it has learned at the earliest opportunity, so the teacher can benefit from it or correct it, as appropriate. It observes the teacher at work until it recognizes actions it has already learned; it then predicts future actions, performing them for the teacher’s approval. If it errs, or cannot find an action that fits the current situation, it asks the teacher to resume the demonstration.

Third, the learning system reacts immediately to the teacher’s actions so that she is constantly aware of its beliefs about their pre- and postconditions, from which it will abstract program variables and conditional operators. To avoid swamping the teacher with information, feedback is graphical and limited to a simple classification of the system’s perceptions— objects are highlighted according to their significance. If the system needs more information about the current situation, it asks the teacher a few simple questions.

Figure 1 illustrates the complete system. Principal components are the drawing utility A.Sq, which constitutes the target domain; Metamouse itself, which maintains a consistent metaphor to allow the teacher to model the system’s capabilities; and the learning program Daedalos which infers graphical procedures. The loop is closed by a constraint solver which performs predicted actions.
The next sections describe some details of the metaphor, the task domain, the representation of knowledge, and the learning mechanism. We then evaluate the intuitive appeal of the metaphor through a questionnaire and present quantitative results on the efficiency and reliability of the learning method.

The Metamouse Metaphor

Our metaphor encourages teachers to augment demonstrations with non-symbolic annotation—graphical constructions. The Metamouse, known as Basil, has a body and "touch" sensors, through which he isolates spatial relations in his own vicinity, ignoring larger patterns. The teacher recognizes Basil's restriction to these superficial attributes because she can see his reactions to touch (by highlighting objects). Basil also has a memory associating objects with situations, and a generalization model of spatial constraint. These are manifested in his predictions and thereby conveyed indirectly.

Teachers are introduced to Basil through his autobiography, excerpted in Figure 2.

The teaching protocol is a set of "felicity conditions" [van Lehn 83] adapted from the teaching of arithmetic procedures to children. Teachers must show all steps in a task when demonstrating it; examples must be correct; "invisible objects"—such as the gap between pillars of an arch—must be represented; and irrelevant activity and variability in the order of actions should be kept to a minimum. Satisfying these conditions makes induction feasible by improving reliability and vastly reducing the search space. The interaction techniques that the system employs are designed to help teachers meet the felicity conditions.
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 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.

So in general each step of a task means moving until I bump into something.

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.

Now, this is very important. 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.

If I don't sense a bump I'll ask you why you stopped where you did.

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.

Figure 2. Excerpts from description of Metamouse.

**Graphical Tasks**

Our drafting utility has two primitives, line and box, which the user manipulates by moving icons that represent key-points called "handles" (as in MacDraw). The program is implemented on the Macintosh computer and follows its user interface conventions. Convenience features such as automated alignment are not built in, but can be taught to the system.

Figures 3 and 4 illustrate three simple tasks a user might like to automate: distributing boxes at equal intervals across the screen; maintaining the connectivity of line segments after one of them is moved; and aligning boxes to a guideline specified at run-time.
Consider the “Box-to-line” procedure as taught by example, illustrated in Figure 4. The teacher performs the task, directly manipulating objects as usual; Basil follows along, highlights objects’ handles according to the touch relations he senses, occasionally asks questions, and eventually takes over execution of the task. The teacher begins by placing the guideline’s two end-points (Figure 4b, c); Basil records two actions — move to first end-point and draw a line to second. Observing the absence of a contact constraint on these steps, Basil interrupts to ask (through a dialogue box, Figure 4c) whether the locations are constant or run-time inputs. The teacher indicates that both are the latter.

She then leads Basil through the main iterative sequence (Figure 4e-k). In general, selection and iteration depend on any number of properties of objects or situations. Iteration should be ordered and conditioned on events Basil can sense by touch. A horizontal sweep-line serves this purpose, and also constrains the boxes’ path of translation. The teacher draws a “sweep-line” near the bottom of the screen and indicates (through dialogue boxes) that its initial placement is constant (Figure 4d). She then grabs the sweep-line at its mid-point handle and moves it upwards until it touches the bottom edge of some box (Figure 4e).

The teacher picks and drags the first box rightward until its lower right corner touches the guideline, its bottom edge still on the sweep-line (Figure 4f). She then re-grasps the sweep-line (Figure 4g); this action patently repeats that of Figure 4e. Consequently Basil conjectures a loop and performs the next sweep-until-contact himself. Since the teacher accepts this prediction, Basil continues executing the loop. The second box, however, must be moved to the left. Basil ranks constraints and generalizes the weaker of them: having learned to move the box until contact with the guideline and sweep-line, objects he knows individually, he ignores the direction.

After processing the third and final box, Basil realizes that he cannot complete the action of moving the sweep-line as he has learned it (Figure 4k). Hence he terminates the loop on the condition of being unable to perform its first step, and calls upon the teacher to demonstrate what to do. At this point, she has Basil remove the sweep-line and guideline (Figure 4l), and then announces that the lesson is over.
Representing Graphical Knowledge

A procedure is modelled as a directed graph of program steps, where each may have several predecessors and successors; a program may contain arbitrary branches and loops. Each step is a 4-tuple (precondition, operator, path, postcondition), based on the STRIPS
paradigm [Fikes 71] and the robot programming language in the NODDY system [Andreee
85]. The next program step is predicted only if precondition holds and the constraint solver
can apply operator along path to achieve postcondition. Pre- and postconditions comprise
Basil's position and touch-sensor feedback. The operator is one of (draw-line, draw-box,
move, drag). The path is a generalization into one of eight octants of the vector between
the operation's start and end points.

Notions of touch constraint are fundamental in the domain knowledge, as evidenced in
geometry [Freudenthal 67], in a cognitive theory of drawing [van Sommers 84], and in
recent work on graphical knowledge [Geller 87]. Basil distinguishes touch relations
according to the types of parts involved: self is Basil's snout, a point; handles (the
rectangular buttons used to manipulate objects) are in effect points surrounded by gravity
fields; edges connecting handles are lines with gravity. The various types of touch (eg.
point-to-point) express degrees of constraint (see below). Basil senses touch relations
involving himself (touching or grasping) and what he is grasping. (The latter gives the
system much of its power to represent graphical constructions.) More remote touches are
ignored. Figure 5 illustrates.

![Figure 5. Examples of Metamouse touch.](image)

Individual objects occurring in each trace step are generalized to program variables, to
be instantiated at run-time by selector functions. A variable captures the recurrence of an
object throughout an action sequence. In Box-to-Line, for example, Basil draws the sweep
line, then grasps its mid-point. Some objects are recurrences of existing bindings, while
others are new bindings forced by a selector function implicit in the action. Selector
functions include create, which sets a variable to the object just drawn, and find, which
invokes the constraint solver to obtain an object satisfying path and touch constraints.

Having determined the variables used in a step, the system proceeds to analyze its pre-
and postconditions. This analysis not only permits generalization but also enforces the "no
invisible objects" felicity condition, since it reveals whether the action was sufficiently
constrained. Touch relations express constraints of different stringency. Moving to the
mid-point of the line just drawn, for example, is a completely determining constraint.
Domain knowledge is used to classify each element of Basil's sensory feedback — touch
relations, direction and position. Using this initial classification, ineffective constraints are
ignored (a determining constraint makes all others ineffective). If insufficient constraint is
found, Basil asks the teacher to "explain" why the action ended where it did. Several
options are given: position may be a constant (eg. the sweep line S in Figure 4), or a run-
time input (e.g. the guideline G); or distance moved may be a constant or input; or the teacher may have neglected to use a visible construction.

Inducing Procedures

A preliminary experiment with several potential users underscored the importance of reducing the amount of free variation—doing the same thing in different ways on different occasions—in the teaching sequence [Maulsby 88]. In order to remove such opportunities from the teacher, the learning algorithm described below takes advantage of interaction, by predicting the teacher's actions.

At any stage there is a nascent program, represented as a directed graph of program steps as described above. The learning algorithm alternates between executing and constructing the program. It steps through the graph until: the end of the program is reached; the teacher rejects the current step predicted; or no immediate successor of the current step is executable (no precondition matches actual Metamouse feedback or no postcondition is attainable by constraint satisfaction).

When such a failure occurs, Metamouse asks the teacher to perform the next operation. If possible, one of the failed successors is generalized to subsume the new operation. If not, an attempt is made to match and merge the demonstrated step with another elsewhere in the graph by successfully predicting one of that step's successors. If a merge succeeds, the learning system continues executing the program; otherwise the teacher retains control.

Metamouse acts as an eager apprentice. This eagerness could become irritating if the system made many erroneous predictions, but is dampened by the stringency of action and condition matching. Selecting program steps for execution and matching them with observed actions raise a number of other technical details which there is insufficient space to describe here.

Evaluating the Metaphor

Basil's operation is far too complex to explain in full to potential users. Instead, they are given a shallow description and then improve their understanding through interaction with the system. We have tried to gauge the metaphor's intuitive appeal by measuring the exposure teachers require before they find Basil predictable.

We gave a number of subjects already familiar with the drawing program a 1-page description of Basil (Figure 2), and asked them to work through a questionnaire that presents increasingly complex examples of his actions and feedback regarding his sensory perceptions. After answering each group of questions, subjects were shown the correct answers before proceeding. Figure 6 plots actual vs potential cumulative scores for three typical subjects: this learning curve shows that subjects were able to predict Basil quite well with minimal prior experience.
Evaluating Procedure Induction

Assessing Basil’s performance as a pupil is more straightforward. Since the teacher’s object is to transfer task competence to Basil, we are interested in the rate of learning a correct procedure. “Correctness” means that erroneous predictions are not made (i.e. program steps are not overgeneralized), and perhaps also that predictions are made when the teacher expects them to be (i.e. program steps are not overspecialized).

<table>
<thead>
<tr>
<th>Task</th>
<th>Steps in Trace</th>
<th>Executed by Basil</th>
<th>Edges in Program Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trace #</td>
<td>Passive</td>
<td>Interactive</td>
</tr>
<tr>
<td>Colonnade</td>
<td>1</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>29</td>
<td>27</td>
</tr>
<tr>
<td>Connectivity</td>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5*</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Box-to-Line</td>
<td>1</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

* variant of task: move one end-point rather than entire line

Table 1. Performance data for inducing three procedures.
Table 1 (ignore columns headed “Passive”) records Basil’s performance while being taught the tasks illustrated in Figures 3 and 4. Competence is measured as the ratio of accepted actions performed by Basil to the total performed during a task trace. Overgeneralization is reflected in the ratio of rejected to accepted predictions. For these tasks, it is clear that Basil learned quickly: for “colonnade” competence rose from 12/35 to 27/27 in two traces.

Assessing the Effectiveness of Interaction

Prediction as an interaction technique was evaluated by performing induction with and without it. We captured traces of user actions and Basil’s sensory feedback. These were filtered through programs to infer variables and generalize conditions, and then passed on to the procedure inducer. The interactive sessions were simulated by accepting predictions as the user would have; the passive learning sessions by accepting only predictions that matched exactly what the user did. Table 1 compares the complexity of traces and programs induced in passive vs interactive modes. Interactive traces tend to have fewer steps; extraneous actions are eliminated. Interactive traces also produce simpler programs (having fewer edges in their graph); variation in the order of actions is eliminated and coincidental sensory events are ignored. Since compact programs tend to be more general and robust, these results indicate that interaction does indeed make induction easier and more reliable.

Conclusion

The system demonstrates the feasibility of inducing programs in a direct-manipulation environment. The teacher is able to communicate a procedural abstraction of task behavior without having to conceptualize it or adopt an unnatural—and essentially unproductive—language of abstraction.

The Metamouse metaphor provides a model of conditional action based on intuitively appealing sensory feedback. This helps the teacher conceive of the appropriate abstractions in concrete terms. The intensive interactivity of the teacher’s relationship with Metamouse streamlines the teaching demonstration to render inductive generalization both tractable and reliable.

Acknowledgements

This research is supported by the Natural Sciences and Engineering Council of Canada. We gratefully acknowledge the key role Bruce MacDonald has played in helping us to develop these ideas, and the stimulating research environment provided by the Calgary Machine Learning Group.
References

[Andreæ 85]

[Angluin 83]

[Behnke 74]

[Borning 86]

[Fikes 71]

[Freudenthal 67]
H. Freudenthal, A. Bauer. “Geometry — a phenomenological discussion.” in [Behnke 74].

[Geller 87]

[Halbert 84]

[MacDonald 87]

[Maulsby 88]

[Mitchell 82]

[Myers 87]

[Smith 75]

[van Lehn 83]

[van Sommers 84]