Debugging Shape Definitions for use in Sketch Recognition

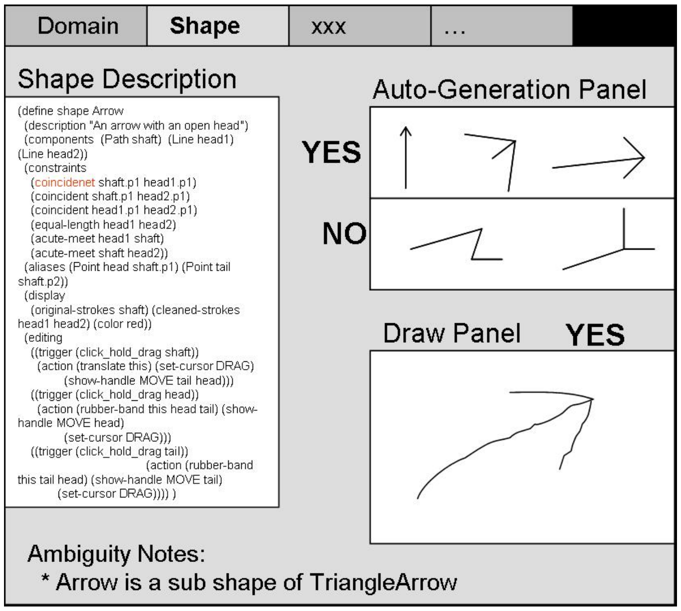
Tracy Hammond and Randall Davis

What: We are building a graphical tool to aid in the creation and debugging of LADDER [1] shape definitions, which are written by developers for use in a domain independent sketch recognition system. The graphical user interface will allow the developer to type a shape definition, open a previously typed definitions, or automatically generate a definition based on a single drawn example [2]. Because the initial shape definition may have errors in it, the graphical user interface will help the developer debug the definition not only for improper syntax, but also to see if it is over or under constrained by integrating a constraint checker to automatically test the constraints specified by the developer and generate implied valid shapes.

Why: To date, sketch recognition systems have been domain-specific, with the recognition details of the domain hard-coded into the system. Developing such a sketch interface is a substantial effort. We propose instead that recognition be performed by a single domain-independent recognition system that uses a domain specific sketch grammar (an approach used with some success in speech recognition [3]). Programmers could then create new sketch interfaces simply by writing a domain description sketch grammar describing all of the domain specific information. The domain description consists of several shape definitions which specify how a shape is drawn, displayed, and edited in a domain.

Domain descriptions are written in LADDER, a language for describing how shapes are drawn, displayed, and edited in sketch recognition. Although efforts are being made to make LADDER as intuitive as possible, shape definitions can often be difficult to describe textually as it is much more intuitive to draw a shape rather than type out a verbal description. Thus we would like to build a user interface to aid sketch interface designers in writing descriptions. Often developers may forget a constraint, allowing unintended shapes to be recognized. Or developers may add too many constraints, and the developers intended shape may not be recognized. Thus we have built a graphical user interface help the developer test when a definition is over or under constrained.

How: The graphical user interface will allow the developer to type a definition, open a previously typed definitions, or automatically generate a definition based on a single drawn example [2]. The developer will then be able to debug the shape definition, not only checking the syntax, but also checking if the definition is over or under constrained. Figure 1 shows an example of what the GUI may look like.



```
(define shape Arrow
  (description "An arrow with an open head")
  (components  (Path shaft)  (Line head1)
(Line head2))
  (constraints
    (coincidenet shaft.p1 head1.p1)
    (coincident shaft.p1 head2.p1)
    (coincident head1.p1 head2.p1)
    (equal-length head1 head2)
    (acute-meet head1 shaft)
    (acute-meet shaft head2))
  (aliases (Point head shaft.p1) (Point tail
shaft.p2))
  (display
    (original-strokes shaft) (cleaned-strokes
head1 head2) (color red))
  (editing
    ((trigger (click_hold_drag shaft))
      (action (translate this) (set-cursor DRAG)
            (show-handle MOVE tail head)))
    ((trigger (click_hold_drag head))
      (action (rubber-band this head tail) (show-
handle MOVE head)
            (set-cursor DRAG)))
    ((trigger (click_hold_drag tail))
                              (action (rubber-band
this tail head) (show-handle MOVE tail)
           (set-cursor DRAG)))) )
```

Ambiguity Notes:
* Arrow is a sub shape of TriangleArrow

Figure 1: An arrow shape definition being debugged.

If a shape definition is over constrained, a drawn shape example will not be recognized (giving false negatives). The GUI will have a draw panel available for the developer to draw several example shapes. If the shape is not recognized based on the definition given, the system will highlight the failed constraint. The developer can then decide to remove or adjust the specified constraint. To recognize the shape, we will use the recognition techniques used to build a recognition engine for LADDER as described in [1].

Once the shape definition is determined not to be over constrained, we can then test if the definition is under constrained. If a shape definition is under constrained, drawn shapes other than the one intended will be recognized (giving false positives). The GUI will randomly generate several shapes that agree (and don't agree) with the shape definition in the auto-generation panel. (See Figure 1.) If any of these shapes are incorrect, the definition is improperly constrained. The developer will then be able to just the missing or under specified constraint. Each shape definition specified the components which make up a shape (for instance, an arrow is made up of three lines). To generate random shapes agreeing with the definition, we will first generate random components. (For example, for an arrow, we will generate three lines of random angle, location, and length.) We will then solve the constraints one by one, altering the components to fit the constraints.

We are currently performing a user study to determine what constraints are commonly forgotten. The GUI will also help remind the developer of commonly forgotten constraints when relevant.

A domain description consists of several shape definitions, so it is also helpful to note if a definition is similar to a previous definition to help debug shapes and prevent ambiguity. The GUI will tell the developer if a shape definition is equivalent or possibly confusable with another definition. Also, shape definitions can be described hierarchically, making them simpler to understand. The GUI will tell the developer when a shape is a sub- or super- shape of another previously defined shape. The GUI will also display shape relation diagrams, such as inheritance or composition hierarchy, such as the one in Figure 2.

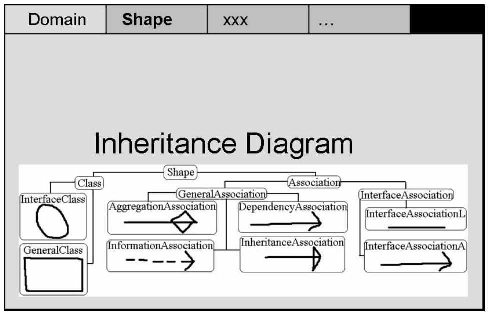


Figure 2: A sample inheritance diagram produced by the GUI.

Progress: We have built LADDER, a language for describing drawing, display and editing in sketch recognition. Olya Veselova [2] has built a system for automatically recognizing salient features of a shape based on one example which we hope to integrate into our GUI.

Future: We have just commenced with building this tool and hope to find it a useful tool for creating and debugging shape definitions.

Research Support: This work is supported in part by the MIT/Microsoft iCampus initiative and in part by MIT's Project Oxygen.

References:

[1] Tracy Hammond and Randall Davis. Ladder: A language to describe drawing, display, and editing in sketch recognition. *Proceedings of the 2003 Internaltional Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[2] Olya Veselova. Perceptually based learning of shape descriptions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[3] Zue and Glass. Conversational interfaces: Advances and challenges. *Proc IEEE*, pages 1166–1180, 2000.