

# Visual Languages '94

Reprint:

Repenning, A., "Bending Icons: Syntactic and Semantic Transformation of Icons," *Proceedings of the 1994 IEEE Symposium on Visual Languages*, St. Louis, MO, 1994, pp. 296-303.

## Bending Icons: Syntactic and Semantic Transformations of Icons

Alex Repenning

Department of Computer Science and Institute of Cognitive Science  
Campus Box 430  
University of Colorado, Boulder CO 80309  
(303) 492-1349, ralex@cs.colorado.edu  
Fax: (303) 492-2844

### Abstract

The notion of icons in visual environments is limited by perceiving icons as tacit entities that have meaning only to human beings and not to the machines that display them. This perception leads to visual tools that provide very little support for the creation of *related icons* representing *related concepts*. A large number of complex icons can be generated automatically by applying simple syntactic and semantic transformations to more fundamental icons. These transformations can significantly reduce the laborious work of icon designers and programmers. This paper describes some of the essential icon transformations that have emerged from the experience of 25 designers using the Agentsheets system and creating a total of 500 icons.

### Keywords

agents, incremental programming, spatial metaphor, visual programming, syntactic transformation, semantic transformation, flow metaphor, topology, picture extrapolation

1. Introduction.....	296
2. Agentsheets: A Programming Substrate.....	296
3. Syntactic Transformations of Icons .....	299
4. Semantic Transformations of Icons.....	301
5. Conclusions .....	303

# Bending Icons: Syntactic and Semantic Transformations of Icons

Alex Repenning

Department of Computer Science and Institute of Cognitive Science  
Campus Box 430  
University of Colorado, Boulder CO 80309  
(303) 492-1349, ralex@cs.colorado.edu  
Fax: (303) 492-2844

## Abstract

The notion of icons in visual environments is limited by perceiving icons as tacit entities that have meaning only to human beings and not to the machines that display them. This perception leads to visual tools that provide very little support for the creation of *related icons* representing *related concepts*. A large number of complex icons can be generated automatically by applying simple syntactic and semantic transformations to more fundamental icons. These transformations can significantly reduce the laborious work of icon designers and programmers. This paper describes some of the essential icon transformations that have emerged from the experience of 25 designers using the Agentsheets system and creating a total of 500 icons.

## 1. Introduction

Designing icons is laborious work and should be automated wherever possible. The lack of automatic icon creation tools is rooted in the perception that icons are tacit entities that have meaning only to human beings and not to the machines that display them. In this model it is typically the role of visual designers to render abstract concepts defined by application designers to concrete icons that visually represent these concepts. To automate icon creation would mean to build mechanisms that take over the intricate role of visual designers. However, the design of good icons efficiently communicating concepts is demanding and has often been underestimated by application designers [7]. Consequently, the feasibility of approaches to automatically render abstract concepts to concrete visual representations is problematic.

The problem of automatic icon creation can be reduced to the problem of icon transformation after realizing that in

the majority of icon-based applications *related concepts* are represented by *related icons*. That is, applications typically do not consist of completely orthogonal concepts that need to be represented with radically different icons. Instead, groups of icons often are variations revolving around a common theme and could, in many cases, be created through automatic transformations of icons. These transformations, because they are concerned with the look of icons, are *syntactic transformations*.

Syntactic transformations of icons can imply *semantic transformations*. Icons that are related with respect to how they look are typically also related with respect to what they mean. Generally speaking, the semantics of icons cannot be determined by analyzing them on a pixel level. This would require complex interpretation abilities that parallel the visual recognition skills of human beings [1]. A more practical approach allows icon designers to annotate icons with semantic information. Syntactic transformations of icons, then, can also transform the semantics captured in icon annotations. To that end, a transformation of the look of an icon, for example by rotating the icon, determines how the rotated icon behaves.

This paper describes the experience of 25 users of a programming substrate, called Agentsheets, creating 40 applications with more than 500 icons during the past four years. This paper briefly introduces Agentsheets, and describes syntactic and semantic icon transformations

## 2. Agentsheets: A Programming Substrate

This section provides only a brief summary of the Agentsheets system in order to give the reader an intuition about the design process of icon-based applications using Agentsheets or related systems. For a more detailed discussion of Agentsheets and applications created with

Agentsheets the reader is referred to other papers describing the design of a visual programming language for voice dialog design at US WEST [14], illustrating the use of new interaction styles for visual problem solving [15], analyzing trade-offs in grid-based systems [13], and providing an elaborate philosophical argument for the use of agents in visual programming [12].

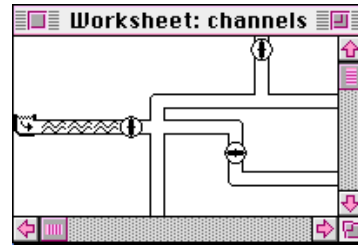
Agentsheets is a programming substrate for building dynamic, visual environments. In the past four years, Agentsheets has been used to create domain-oriented [3] visual programming languages in domains such as art, artificial life, distributed artificial intelligence, education, environmental design, and simulation.

Visual environments created using Agentsheets consist of a large number of autonomous, communicating agents [8] organized in a grid (see Figure 3: (7)), called the *agentsheet*. Agentsheets is an object-oriented spreadsheet extension similar to the system described by Piersol [10]. An agentsheet cell can contain any number of stacked-up agents. Users interact with agents through direct manipulation. Agents can be animated, move among cells, play sounds, and use speech synthesis.

**Metaphors of Flow**

Many Agentsheets applications revolve around the metaphor of *flow* [6]. Flow is an important concept in many different visual programming approaches such as Prograph [5], HI-Visual [16], and Khoros [11].

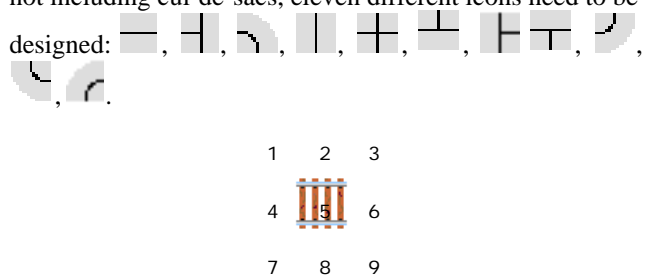
Agentsheets adapts a different model of flow that treats flow as the propagation of agents through a discrete space. This space is often constrained by conductors, such as pipes, wires, rivers, and railway tracks that are represented by agents. For instance, in the Channels application agents are used to model pipes conducting water (Figure 1). The flowing agents can represent either fluids or discrete entities. Discrete entities such as cars, have to make explicit decisions when faced with the topology of a fork. A car can go only one way or the other. Fluids, on the other hand can be distributed.



**Figure 1. Channels Application**

Modeling flow with adjacent agents in a grid can have advantages over visual representations based only on topological considerations [2]. The discretizing effect of grids can turn secondary notation [9] (the use of white space, adjacency, and clustering) found in many diagrammatic representations into tangible information. Pipe systems represented with the Channels application, for instance, do not merely represent a *topology* of connected points. Instead, they allow the derivation of information based on *Euclidean* characteristics such as distance. In the case of the pipe system, pipes model non-ideal conductors of fluids. Water, while moving through pipes, is evaporating. Simply drawing the pipe system based on pipe segments will implicitly derive the water loss, due to evaporation, between any two points in the system.

To create the set of required icons representing components such as pipes or roads connecting neighboring cells is laborious. In Figure 2 the icon in cell 5, representing a railway track, implies a connection between cells 4 and 6. In order to express all possible connectivity patterns between adjacent neighbors (cells 2, 4, 6, and 8), not including cul-de-sacs, eleven different icons need to be designed:



**Figure 2. An Agentsheet Cell and its Neighbors**

Designers need tools to efficiently create sets of related icons and they need to have means to attach semantics to icons in order to simplify the task of creating icon-based systems.

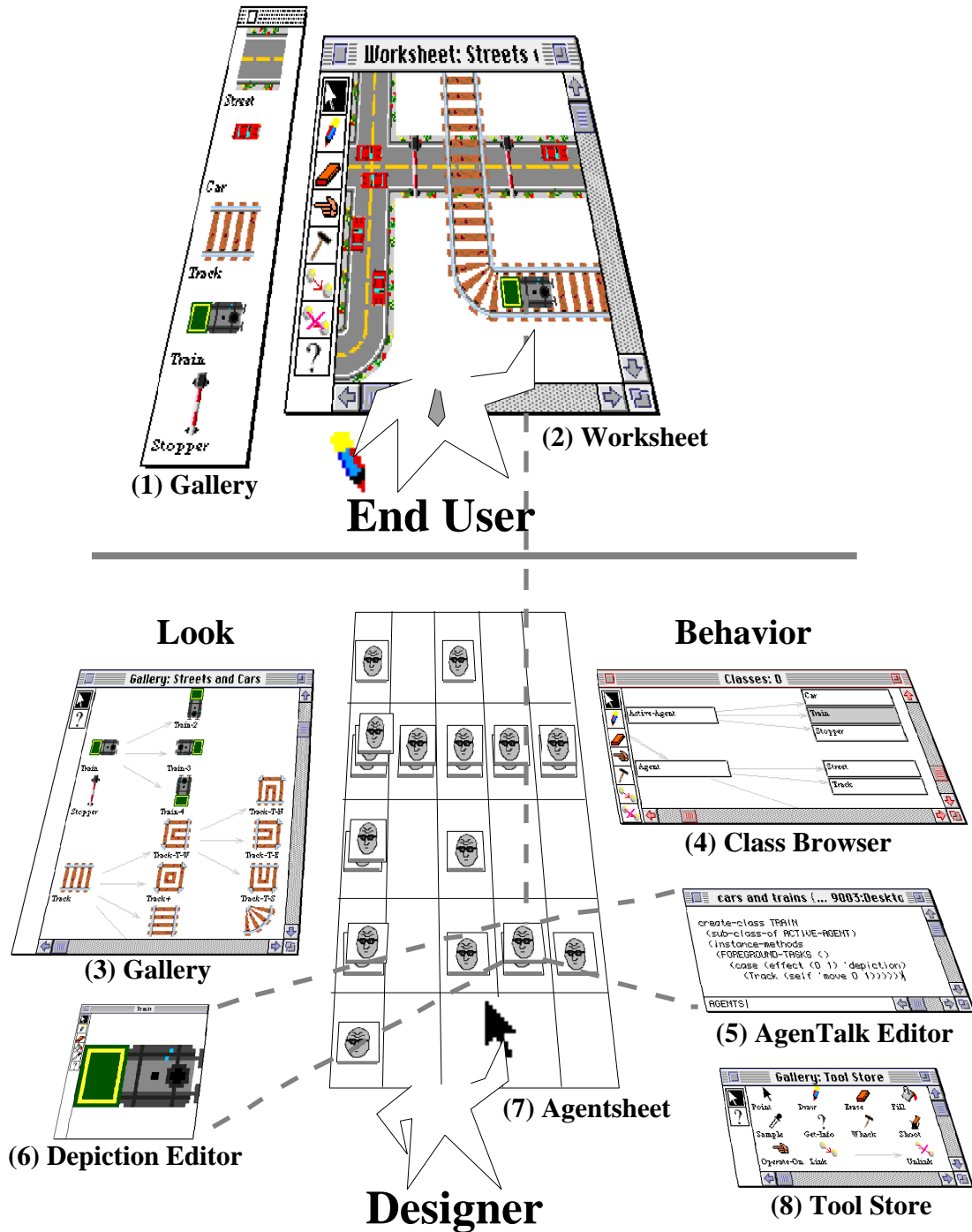


Figure 3 : The Structure of an Agentsheet

Agentsheets provides user interfaces for end users and designers of visual programming languages. End users compose programs by selecting components in the gallery (1) and putting them into a worksheet (2). Designers perceive worksheets (2) as agentsheets (7), i.e., agents organized in a grid. They create networks of related depictions in the expanded gallery (3), design icons with the depiction editor (6), define behavior with the AgenTalk editor (5) by reusing existing agent classes found in the class browser (4), and create or subscribe to tools in the tool store (8), allowing end users to interact with agents. Worksheets, galleries, depiction editors, class browsers, and tool stores are all agentsheets.

### 3. Syntactic Transformations of Icons

This section illustrates the automatic generation of icon sets through syntactic transformations. Similar to some of the morphological transformations described by Fujii [4], syntactic transformations in Agentsheets are concerned with visual features of icons. The transformations described in this paper reflect the metaphor of flow.

The principles of syntactic icon transformations are illustrated in the context of the CityTraffic application created for urban planners to analyze traffic patterns. CityTraffic makes use of the flow metaphor: road segments and railway tracks are flow conductors. Cars and trains are discrete entities of flow. Again, in the CityTraffic application Euclidean characteristics are relevant. It is not sufficient to know that two places in a city are connected. What also matters is the likelihood of collision, the time it takes to move from one point to another, and the interaction between cars, trains, and traffic signals.

The *gallery* (Figure 3: (1), (3)) is used by a designer to create a network of related road icons. First, the designer creates a base icon (Figure 4) representing a straight street segment using the depiction editor (Figure 3: (6)). This icon will be the basis for syntactical transformations that yield variations of the street icon representing different connectivity patterns of roads.

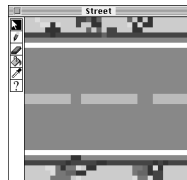


Figure 4. Base Icon Representing Street

The transformations have emerged from analyzing related icons in earlier applications created with Agentsheets. Designers often created sets of related icons by painstakingly drawing each icon from scratch because the kind of transformations they required were not supported in traditional icon editors. Building some of the observed manual transformations by icon designers into the Agentsheets substrate has literally reduced the time it takes to create icon families from hours to minutes. This is especially true for complex color icons.

In addition to simple icon transformations, such as rotation and flipping, new transformations can be defined that are relevant to the metaphor of flow. One important transformation is the *bending* of icons. Bending the base icon<sup>1</sup> of Figure 4 yields the icon in Figure 5:

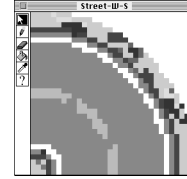


Figure 5. Curve

The color of each destination icon pixel at a location  $\langle x, y \rangle$  is defined by looking up the color of the source icon pixel of the at location  $\langle x', y' \rangle$ :

$$x' = N \frac{\sin^{-1} \frac{x}{r}}{\sqrt{2}} \quad y' = r$$

where N is the size of the icon and radius r is determined by  $r = \sqrt{x^2 + y^2}$

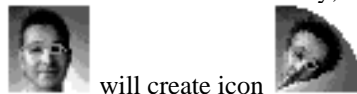
More complex icon transformations require *picture extrapolation*. That is, in the process of the transformation some source icon pixels need to be copied to multiple locations in the destination icon. The selection of appropriate source pixels often requires heuristics. Many transformations require the segmentation of icons into four areas (Figure 6):



Figure 6. Icon Segmentation

For example *forking*, *crossing*, and *sharp bending* (Figures 7-9) are transformations based on rotating, flipping and reassembling these four segments.

<sup>1</sup>Base icons can be arbitrary; for instance bending icon



will create icon

- *Forking:*

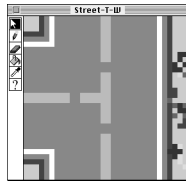


Figure 7. T Segment

- *Crossing:*

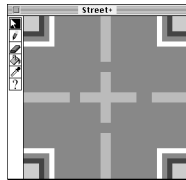


Figure 8. Intersection

- *Sharp Bending:*

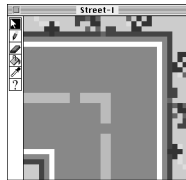


Figure 9. Corner

Instead of applying transformations to icons individually, designers can run *transformation scripts* to create frequently used networks of related icons. In a typical scenario a designer would start by creating a set of base icons. In the gallery shown in Figure 10 the base icons for a traffic simulation have been drawn.

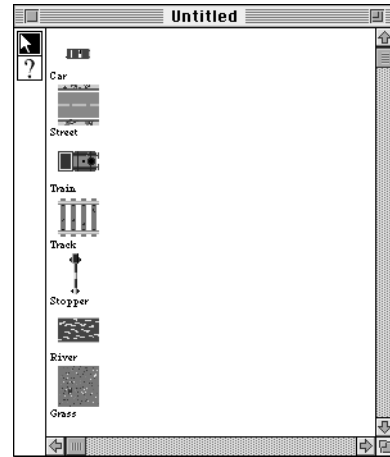


Figure 10. Gallery Containing Base Icons

The designer selects the track icon and applies a transformation script to create all icons required to represent connectivity among the four adjacent neighbors of the icon. The script creates the icons and generates names for the icons (Figure 11).

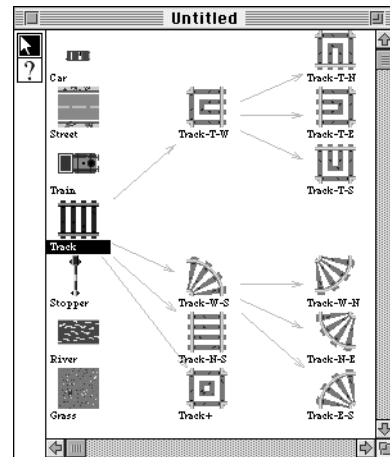


Figure 11. Transformed Tracks

Automatic transformations not only save time, they also encourage designers to experiment with different looks of icons while maintaining the consistency between related icons. Without automatic transformations the need to change a base icon would force designers to manually touch up all related icons.

After transforming all base icons, the gallery can be used to draw a complete scene (Figure 12).

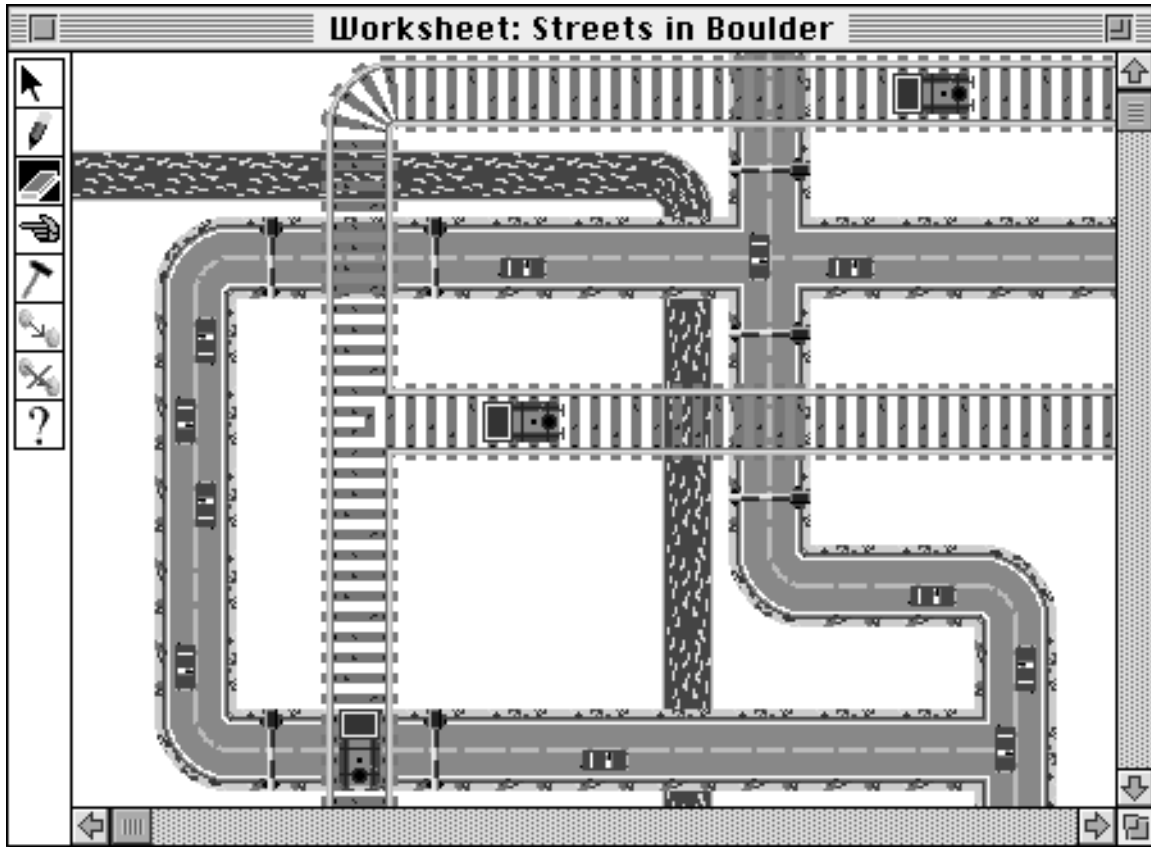



Figure 12. CityTraffic Simulation for Urban Planners

#### 4. Semantic Transformations of Icons

In addition to transforming syntactic aspects of icons, transformations can be applied to the semantics of icons. Icons are augmented with their connectivity pattern. That is, designers specify which neighbors are connected by the icon. For instance, a diode is a semiconductor that lets current flow through it in only one direction. Designers specify the connectivity of icons with the connectivity editor in Agentsheets (Figure 13) by clicking at neighboring cells. Each neighboring cell can either be an input (arrow, , pointing in the direction of the icon), an output (arrow pointing away from depiction), a combined input/output, or neither.

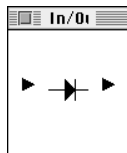



Figure 13. Connectivity of a Diode

The diode icon, , implies connecting the left with the right cell but not the other way around. The arrows to the left and to the right of the diode icon have been specified by the designer. They are the *semantic augmentation* of the icon.

Applying icon transformations will not only change the look of the icons but also their semantic augmentation. In the simple case of rotating the diode icon by 180 degrees the resulting connectivity pattern is as shown in Figure 14.

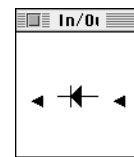


Figure 14. Connectivity of a Diode Rotated 180 Degrees

The same mechanism works for more complex transformations. A straight piece of a two-way street (Figure 15) has the semantics of a conductor from left to right and from right to left.

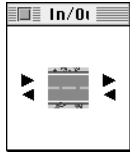


Figure 15. Semantics of Street

Applying bending, forking and crossing transformations to this icon also transforms its semantics (Figure 16).

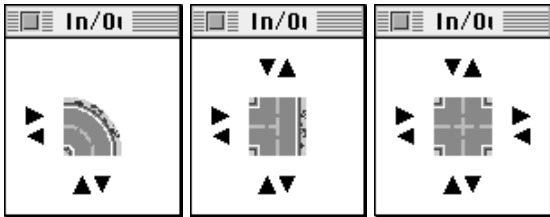













Figure 16. Street After Bending, Forking, and Crossing

The ability to transform the syntax and the semantics of icons with the same type of operations narrows the gap between the icon image and the icon meaning [7]. If a base icon has been successfully defined, that is, the icon's intent is recognizable by users, then transformations are likely to produce recognizable icons as well. The novelty of this approach is that syntactic operations on the pixel level of icons, such as bending icons, drives the meaning of the icons.

The ability to augment icons with semantic information and to transform that semantic information reduces the complexity of programming. If semantics are not explicitly built in to icons they would need to be provided, implicitly, in the program defining the behavior of components. Without this information a designer specifying the interactions of cars with streets in the CityTraffic application would have to write tedious fragments of code to express that cars should follow roads.

For each direction the car is heading the car has to distinguish between a large number of street icons it could be on top of in order to make the decision of either going straight, or making a left turn or a right turn (Figure 17).

```

TASKS (Car) :
CASE my- icon
 : {car is moving east}
CASE ON-TOP-OF
 : go (straight);
 : go (right);
 : go (left);
 : go (left) OR go (right);
 : go (left) OR go (straight);
 : go (right) OR go (straight);
 : go (left) OR go (right) OR go (straight);
 : {car is moving north}
CASE ON-TOP-OF
 : {car is moving west}
CASE ON-TOP-OF
 : {car is moving south}
CASE ON-TOP-OF
...

```

Figure 17. Code For Car Moving On Road

In the above program the mapping between the look and meaning of icons is hard coded. The direction of the car and the connectivity of the street segments are implicit in the code. By making use of the connectivity semantics associated with cars and streets the code can be reduced to:

```

TASKS (Car) :
go (choose-one (directions-to-go (heading (my- icon),
                                exits (ON-TOP-OF)))

```

Possible directions for the car are determined based on where the car was heading and where the car could go . The car selects randomly one of the possible directions and moves.

The point here is not to write an optimal program for city traffic simulation. Instead, it is important to note that quite often related icons are used to represent related concepts. The syntactic as well as the semantic aspects of these relationships can be captured in icon-based systems. This leads to a new perception of icons no longer reducing icons to tacit entities that have only meaning to human beings.



## 5. Conclusions

While describing syntactic and semantic transformations, such as bending icons, this paper would like also to “bend” people’s perceptions regarding icons. Icons are not just visual representations of abstract concepts. Instead, icons can have intrinsic semantics and syntax that can be transformed. Tools can help designers of icon-based systems to create *related icons* representing *related concepts* through automatic icon transformation. Syntactic transformation of icons, on one hand, can significantly simplify the laborious task of drawing icons. Semantic transformations of icons, on the other hand, can facilitate the task of programming by attaching semantics to icons instead of embedding semantics implicitly in programs dealing with icons.

## Acknowledgments

Many thanks to the numerous users of Agentsheets who have created wonderful applications. I also thank the HCC group at the University of Colorado, and Clayton Lewis, who contributed to the conceptual framework and the systems discussed in this paper. Roland Hübscher has provided crucial insights to the bending problem. This research was supported by the National Science Foundation under grant No. RED-9253425, Apple Computer Inc., and US West Advanced Technologies.

## References

1. Arnheim, R., *Visual Thinking*, University of California Press, Berkeley, 1969.
2. Citrin, W. V., “Requirements for Graphical Front Ends for Visual Languages,” *Proceedings IEEE 1993 Workshop on Visual Languages*, Bergen, Norway, 1993, pp. 142-149.
3. Fischer, G. and A. C. Lemke, “Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication,” *HCI*, Vol. 3, pp. 179-222, 1988.
4. Fujii, H. and R. R. Korfhage, “Features and a Model for Icon Morphological Transformation,” *Proceedings 1991 IEEE Workshop on Visual Languages*, Kobe, Japan, 1991, pp. 240-245.
5. Golin, E. J., “Tool Review: Prograph 2.0 from TGS Systems,” *Journal of Visual Languages and Computing*, pp. 189-194, 1991.
6. Lakeoff, G. and M. Johnson, *Metaphors We Live By*, The University of Chicago Press, Chicago and London, 1980.
7. Levialdi, S., P. Mussio, M. Protti and L. Tosoni, “Reflections on Icons,” *1993 IEEE Symposium on Visual Languages*, Bergen, Norway, 1993, pp. 249-253.
8. Minsky, M., *The Society of Minds*, Simon & Schuster, Inc., New York, 1985.
9. Petre, M. and T. R. G. Green, “Learning to Read Graphics: Some Evidence that Seeing an Information Display is an Acquired Skill,” *Journal of Visual Languages and Computing*, pp. 55-70, 1993.
10. Piersol, K. W., “Object Oriented Spreadsheets: The Analytic Spreadsheet Package,” *OOPSLA '86*, 1986, pp. 385-390.
11. Rasure, J. R. and C. Williams S., “An Integrated Data Flow Visual Language and Software Development Environment,” *Journal of Visual Languages and Computing*, pp. 217-246, 1991.
12. Repenning, A., “Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments,” University of Colorado at Boulder, Ph.D. dissertation, Dept. of Computer Science, 171 Pages, 1993.
13. Repenning, A. and W. Citrin, “Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction,” *1993 IEEE Workshop on Visual Languages*, Bergen, Norway, 1993, pp. 77-82.
14. Repenning, A. and T. Sumner, “Using Agentsheets to Create a Voice Dialog Design Environment,” *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, Kansas City, 1992, pp. 1199-1207.
15. Repenning, A. and T. Sumner, “Programming as Problem Solving: A Participatory Theater Approach,” To Appear in: *Workshop on Advanced Visual Interfaces '94*, Bari, Italy, 1994.
16. Yoshimoto, I., N. Monden, M. Hirakawa, M. Tanaka and T. Ichikawa, “Interactive Iconic Programming Facility in HI-VISUAL,” *IEEE Computer Society, Workshop on Visual Languages*, Dallas, 1986, pp. 34-41.