Visual Semantics – Or: What You See is What You Compute

Martin Erwig
FernUniversität Hagen, Praktische Informatik IV
58084 Hagen, Germany
erwig@fernuni-hagen.de

Abstract

We introduce visual graphs as an intermediate representation between concrete visual syntax and abstract graph syntax. In a visual graph some nodes are shown as geometric figures, and some edges are represented by geometric relationships between these figures. By carefully designing visual graphs and corresponding mappings to abstract syntax graphs, semantics definitions can, at least partially, employ a visual notation while still based on abstract syntax. Visual semantics thus offers the "best of both worlds" by integrating abstract syntax and visual notation. These concepts can also be used to give visual semantics for traditional textual formalisms. As an example we provide a visual definition of Turing machines.

1: Introduction

Language semantics are conveniently expressed based on abstract syntax. This allows to abstract from details of concrete syntax and leads to more succinct semantics definitions, and in some cases it makes semantics definitions even tractable at all. In [5] we have presented a framework for defining semantics of visual languages. The approach is fundamentally based on an abstract graph syntax for visual languages. Thus, it is essentially a textual formalism which is unfortunate (to a certain degree) for two reasons: first, visual relationships, such as *inside* or *adjacent*, are represented in a topological way by appropriately labeled edges. By this transition from a visual to a textual representation, many characteristics of the visual language under consideration are difficult to grasp in the abstract representation, they might even get lost. Second, the textual treatment of a visual language presents in a sense a "modality mismatch", more precisely, it means a retrogression from visual to textual. This might cause psychologically grounded reluctances to using the formalism – just because it is not visual.

These drawbacks might be obstacles for a wide-spread use of the formalism in the VL community. Therefore, we extend the semantics formalism to stay, to a large degree, with visual notation when defining semantics. The main idea is to re-visualize some relationships of abstract syntax that have been translated more or less directly from geometric relationships. This means that some nodes are visualized as geometric figures, and some of the edges between these nodes are represented by relationships that hold between the figures. Since this will not cover, in general, all relationships, we arrive at a semi-visual notation, that is, a mixture of graphs and pictures, called *visual graphs*, in which complex relationships are still shown as edges between nodes. The goal is then to use visual graphs in semantics definitions. Therefore, we need a mapping from visualizations back to abstract graph syntax. This is achieved by typed graph rewriting systems.

A further application of visual graphs and visual semantics is to give visual semantics definitions for traditionally non-visual formal systems. It seems that this means just to define a visualization for such a formalism. However, only defining a mapping from the mathematical structures to a visual domain is not enough. To have a truly visual semantics we again need a mapping from the visual representation back to the formal structures. Thus, visual semantics in that application domain means to formally define a (semi-) visual language for that application.

Hence the paper has the following contributions: First of all, we provide a more intuitive notation for semantics definitions of visual languages. In addition, we offer a framework for defining (semi-) visual languages for formal systems, and by that we disclose a further application area for visual languages that has not been investigated so far.

In the next section we illustrate the approach by giving a visual semantics for traditional, textual formalism.

2: A visual semantics for turing machines

Let Q denote a set of states ($F \subseteq Q$ is the set of final states), and let S denote a set of symbols. The (partial) function $\delta : Q{\times}S \to Q{\times}S{\times}\{L, R\}$ defines the state transitions of a Turing machine. We will represent a Turing machine by *two* graphs. By this we can avoid complex graph pattern matching and keep the definition quite simple. This also illustrates that our basic graph formalism is more versatile than graph grammars which work on just one single graph. The ability to use more than one graph offers some kind of modularity in semantics definitions.

We represent the δ function as a $(\emptyset, S{\times}S{\times}\{L, R\})$-graph, that is, we use Q as node identifiers (needing no node labels), and each edge (v, w) is labeled by a triple "$X{\to}Yd$" expressing that on reading X in state v the Turing machine prints Y, moves its head according to d, and enters state w. For example, the Turing machine accepting the language $\{a^n b^n \mid n > 1\}$ is shown in Figure 1. This abstract graph representation of a Turing machine program nicely follows the graph representation of finite automata.

The tape of the Turing machine is given by another, now visual, graph in which cells are represented by adjacent rectangles. This is mapped to an abstract graph representation in which nodes are connected by *next*-edges. The

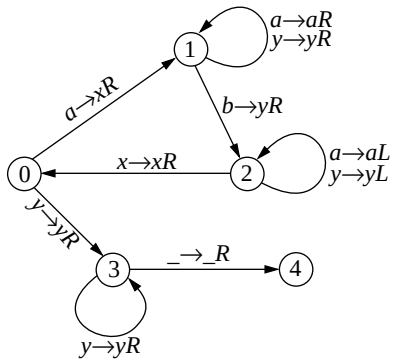
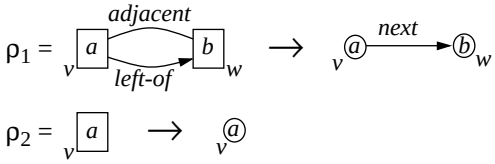


Figure 1: Turing Machine accepting $a^n b^n$

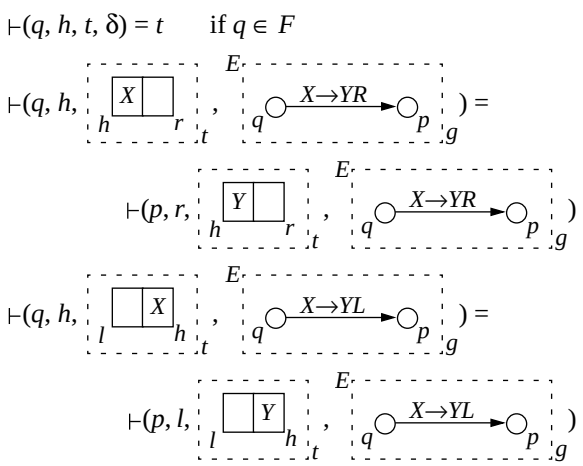
representation of Turing machines as abstract syntax graphs is defined by a graph rewrite system ρ_{TM}:



Note that ρ_2 is needed to ensure relabeling of (the final) node of a rewriting chain that has no incident edges anymore.

A configuration (q, h, t, δ) of the Turing machine is represented by two graphs (the tape t and the transition function δ) and two nodes defining the current state (q) and the tape cell currently seen (h).

The semantics of the Turing machine can now be given by defining a state transition function $\vdash$ by just three simple equations:



The first equation gives the terminating case: if q is a final state, we simply return the tape as a result. Otherwise, we match the current head node h in the tape graph, retrieve its label X, and find that outgoing edge (q, p) of the current state q in the δ graph that consists X as its first label component. If the last label component is R, we continue with the next state p and the right neighbor of h as the new head. In addition, we overwrite the label of h (that is, re-insert node h) with the new label Y, and we also have to re-insert the matched (and removed) edge (q, p). The case for L is analogous.

In the full paper [6] we give a detailed description of visual graphs and their translation to abstract syntax graphs by typed graph rewriting.

3: Related work

Visual semantics in the sense of pure visual rewriting has received considerable attention [1, 2, 3, 7, 8, 9]. All these approaches do not specify relationships to a mathematical domain (which when well-understood can serve as a profound explanation of the defined language) which means a complete devotion to operational descriptions. The purely visual treatment is very attractive, but semantics are sometimes difficult to apply and to deal with in proofs. It should also be noted that the semantics definitions must always be changed when the syntax of the language changes. A visual Turing machine description, quite different from ours, is also given in ChemTrains [1].

4: Conclusions

We have introduced the notion of visual graphs as a model for semi-abstract visual syntax. By defining specialized visual rewrite systems the use of visual graphs in semantics definitions becomes possible. Together with a structured processing of graphs that is offered by the underlying inductive graph definition [4] we are able to define precise semantics of visual languages in an intuitive and easily understandable way. This should make visual language semantics accessible to a broader audience, not least because the application of semantics definitions is greatly simplified.

5: References

[1] Bell, B. & Lewis, C.: ChemTrains: A Language for Creating Behaving Pictures,*VL*'93, 188-195.

[2] Citrin, W., Doherty, M. & Zorn, B.: Formal Semantics of Control in a Completely Visual Programming Language, *VL*'94, 294-301.

[3] Citrin, W., Hall, R. & Zorn, B.: Programming with Visual Expressions, *VL*'95, 208-215.

[4] Erwig, M.: Functional Programming with Graphs, *ICFP*'97, 52-65.

[5] Erwig, M.: Abstract Syntax and Semantics of Visual Languages, *JVLC*, Vol. 9, 1998, to appear.

[6] Erwig, M.: Visual Semantics – Or: What You See is What You Compute, Report 230, FernUniversität Hagen, 1998.
http://www.fernuni-hagen.de/inf/pi4/erwig/papers/VisualSemantics_REPORT.ps.gz

[7] Furnas, G.W.: New Graphical Reasoning Models for Understanding Graphical Interfaces, *CHI*'91, 71-78.

[8] Kahn, K.M. & Saraswat, V.A.: Complete Visualizations of Cuncurrent Programs and Their Executions, *VL*'90, 7-15.

[9] McIntyre, D.W. & Glinert, E.: Visual Tools for Generating Iconic Programming Environments, *VL*'92, 162-168.