Visual Programming Languages and Logic Programming

Beatrice Vaienti

May 2, 2019

1 Introduction

Visual Programming Languages (VPLs), in all their fields of application, have the advantage of making programming more understandable to neophytes and may represent a useful tool for the education of students. In this article we will explore the possibilities offered by the visual language in making logic programming and first order logic more accessible to non-logicians and non-programmers.

The work is organized as follows. In the first section, we outline what is a Visual Programming Language and its history, giving some relevant examples. Furthermore the advantages and disadvantages of using a VPL will be analyzed.

In the second section, we introduce an existing application of a Visual Logic Programming Language, created in the 90's with the aim of design a visual alternative syntax for Prolog. In this section we will also explore briefly the visual formalism underlying this Programming Language.

2 Visual programming

A Visual Programming Language is a typology of programming language which enables the user to manipulate the program elements graphically rather than textually. Thus, the main difference from traditional programming is that visual programming uses more than one dimension to convey semantics [1]. Each object and relationship can be regarded as a token and the collection of one or more token is a visual expression, which can be represented by diagrams, icons, sketches and others. Usually the core idea is to use boxes as entities, connected by arrows, lines or arcs in order to represents relations. In the application of VPL to Logic Programming that we will later outline, for example, higraphs - a topological formalism that combines VENN diagrams together with graphs, developed by David Harel- are used as visual expression. In fact, the choice of

the right visual syntax represents a keypont in the definition of a visual logic.

The first experiences in visual programming explored two directions: how to translate visually the traditional programming languages and independent approaches to visual programming, not directly related to a specific traditional language.
However these early works led to a disenchantment with visual programming, as the obtainable programs seemed exciting only for their feature of being intuitive, while they couldn't reach a realistic complexity and size.
For this reason visual programming researchers began to develop way of integrating the visual approach to the textual one, making VPLs a way of integrating and visualize relationships among data structures. Then developers moved to a more domain specific application of VPLs, thus overcoming the early disadvantages of this languages. In fact, using visual artifacts to solve particular and limited problems makes it possible to work directly in the communication style of the particular problem domain. Today commercial VPLs are available in many different domains, such as programming laboratory data acquisition (labVIEW), education, multimedia, automation and others. Regarding the multimedia and design field, in the latest years many VPLs became more and more popular as modelling tools combined with 3D modelling/CAD software (for example Grasshopper 3d for Rhino and Autodesk Dynamo for Revit). This kind of approach to 3D modelling became fundamental for Parametric Design and Arts in general (suffice it to say that important architecture firms, like Zaha Hadid Architects, use this tools in order to overcome the complexity of freeform design). Moreover algorithmic-aided shaping has created a lot of new possibilities of performing physical simulation oriented to structural optimization.

In conclusion, the main goals sought with VPL research have been to make programming more accessible and understandable to a broad audience, to increase the programming speed for certain task and to avoid errors during this kind of performances.

3 Visual logic programming

First order logic can be difficult to understand for novices, however it is possible to use an intuitive form of problem description, by exploiting visual cues to make the structure of logical expressions more intuitive. In the paper we analyzed [2] a set of customized higraphs are used to create an alternative notation for a limited subset of FOL. The design of a Visual Logic Programming Language involves many subjects, like interactive behaviour of a visual language and the cognitive aspects of diagrammatic reasoning, which is claimed to be closer to our physical intuition than the usual mathematical notation. There are many ways to express logical predicates graphically, however the authors [2] defined a notation which does not require users to deal with logic programming concepts like predicates, arguments, variable matching and others. We will now describe

the features of this particular VPL, which is also able to be implemented as a visual alternative syntax for pure Prolog.

We have two types of terms: (1) FOL terms that represent elements (variables, constants or functional applications) and (2) set terms that represent sets of elements that satisfy a given predicate (predicates, predicate applications, their logical union or their logical intersection).

A visual vocabulary has been defined in order to represents all of these elements: Square boxes represent set terms, rounded boxes and circles represent FOL terms. **Predicates** are represented using a square box with the predicate symbol, a **function** is represented by a rounded box with the function symbol, **variables** are represented by circles. Boolean operators of union, intersection or inclusion are simply represented by the relative position of the graphical elements.

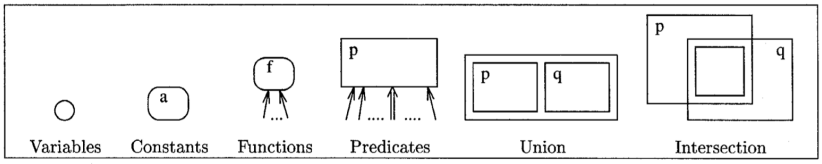

Figure 1: Vocabulary of the fundamental graphical elements

Doing this translation we obtain the nodes of the graph, which we connect in order to specify their relation. We use arrows to connect the subterms to the node that represents the functor of the term, and a double arrow to translate the textual use of an asterisk "*" (meaning "every").

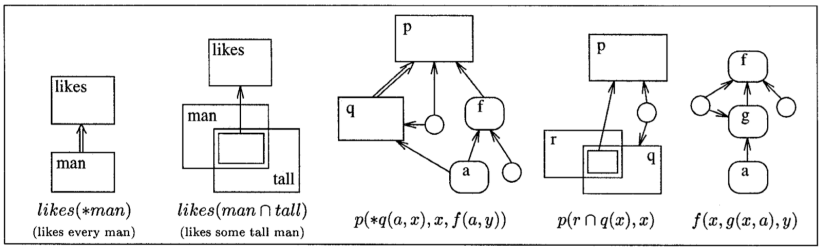

Figure 2: Examples of logic expressions and their relative graphical translation.

Thick lines in the diagram indicate the goal set term, which is the term that is being defined. The authors extended this notation creating a secondary notation but we will not explore all of its details.

A remarkable application of this research is certainly represented by its capability of being a visual alternative syntax for pure Prolog. In fact any diagram can be easily translated into Prolog, and the FOL terms of our visual logic are

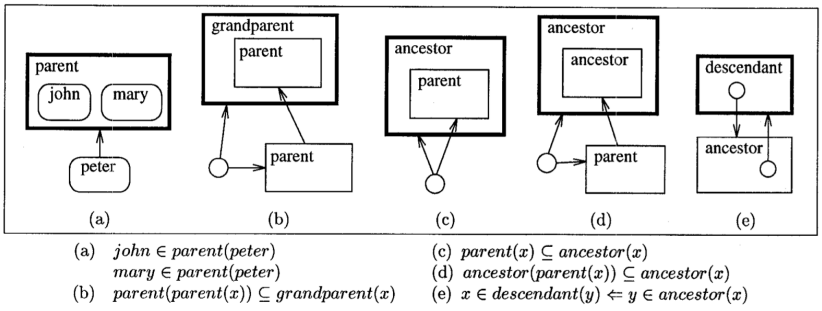

Figure 3: Examples of thick lines used to indicate the goal set term.

now seen as Prolog terms. In conclusion, the work developed by Puigsegur, Agusti and Robertson is clearly in its early stages, and in their paper [2] the authors express their intention to widen their VPL implementing other extralogical Prolog predicates, for instance the negation by failure predicate "not".

3.1 Visual formalism

It may be useful to digress for a moment and deepen the topic of the visualization of information. It is in fact an intricate subject and has been studied for many years for different fields of applications. Our interest is, more specifically, orientated to non-quantitative, but rather relational, information. For this reason we are interested in diagrammatic paradigms that are *topological* and not geometrical in nature [3]. We will refer to this kind of visual topological formalism as *topovisual* formalism.
Leonhard Euler (1707-1783) developed two of the best known topo visual formalisms: (1) the formalism of graphs and (2) Euler circle, that later led to Venn diagrams. It is curious that both of them were developed in the period he lost sight in his right eye, in fact some claim stereoscopic vision reduces one's ability to estimate distances, facilitating on the other hand the awareness of topological features. As we have seen, Puigsegur et Al. used *higraphs* as a graphical base to design a structured VPL. We will now explore better what a higraph is in order to understand better its features and potential.

Graphs are simply composed of a set of nodes and arcs and can represent a set of elements and some binary relations between them, they can be modified to represent a wide variety of elements and relationships (including ones of temporal, causal, functional or epistemological nature). Hypergraphs, instead, constitute an extension of the graph concept, for the relation specified by arcs is not necessarily binary. For this reason it is more difficult to obtain a clear graphical representation of higraphs than of graphs. The important concept to point out, however, is that in both graphs and hypergraphs what we are considering is the topological notion of connectness, where location, shape, distance

and size have no significance.

On the other hand Venn diagrams are able to represent *collections* of sets of

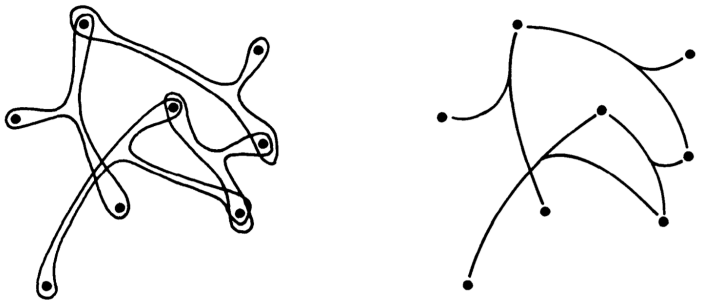

Figure 4: Example of graphical representation of hypergraphs

elements and the relations on them. A set is represented by the inside of a curve, giving the topological notions of enclosure, exclusion and intersection.

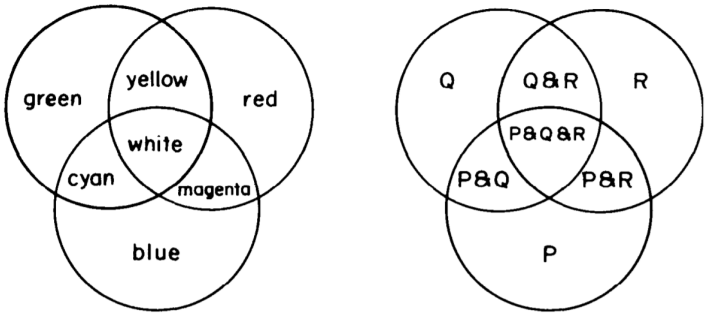

Figure 5: Graphical representation of Venn diagrams

In many computer applications - like the one we are studying - it would be useful to have a system of logical representation able to embed both the characteristics of hypergraphs and of Venn diagrams. It is possible to combine Euler's two formalisms into one topovisual formalism: the higraph, which allows us to work with concepts of connectivity, enclosure, exclusion.

Moreover higraphs can take into account shapes, contrary to hypergraphs, as we have seen in the application of VPL to Logic Programming, where boxes, rounded boxes and circles represented different types of nodes, each one with its peculiarities. For these reasons higraphs are suited for a wide array of applications, and can especially be useful in solving algorithmic problems.

4 Conclusion

The use of visual formalism can represent in a effective way many programming-related situations and problems. In the development of a Visual Programming Language however, it is important to keep together the *visual* feature, which makes it immediate and easily comprehensible, without neglecting *formality*, because they are to be manipulated and analyzed by computers. Many computer-related diagrammatic methods, in fact, accept the *visual*, but reject the *formal*[3], resulting in non rigorous visual instruments. In conclusion, what emerges from the paper analyzed is the confidence in the future developments in a graphical logical way of reasoning and programming, not only for a mere educational purpose, but extended to a wider and more realistic scale of application.

References

[1] Margaret M. Burnett. "Visual Programming". In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. American Cancer Society, 1999. ISBN: 9780471346081. DOI: 10.1002/047134608X.W1707.

[2] J. Puigsegur, J. Agusti, and D. Robertson. "A visual logic programming language". In: (1996), pp. 214–221. ISSN: 1049-2615. DOI: 10.1109/VL.1996.545290.

[3] David Harel. "On visual formalisms". In: *Commun. ACM* 31.5 (1988), pp. 514–530.