A Syntax Directed Approach to Picture Semantics

Dejuan Wang*

Department of Interactive Systems,
CWI (Centrum voor Wiskunde en Informatica)
P.O. Box 94079, 1090 GB Amsterdam,
The Netherlands
e-mail: dejuan@cwi.nl

Henk Zeevat

Department of Computational Linguistics,
University of Amsterdam
Spuistraat 134, 1012 VB Amsterdam,
The Netherlands
e-mail: henk@mars.let.uva.nl

Abstract

If one constructs a visual language, it is necessary to have a methodology for constructing the relationship between aspects of the pictures and the corresponding aspects of the application domain. The various methods for specifying Visual Languages emphasize computational aspects, where the semantics of the picture is embedded in the computational setting of the specification, but they are not based on an understanding of the cognitive issues involved in the semantics of pictures and in the use of the pictures for a better grasp of the application domain and for manipulating the domain. We think that the analogy between a picture and its meaning is what can help people to understand the meaning represented by the picture. and that the match between the syntactic structures of the picture and what it represents is an important element of analogy. We formalise the notion of matching in an approach to picture semantics based on order-sorted algebra. Pictures are described in a well-structured framework (order sorted signature) and so is the application domain. Constructing the relationship between pictures and their meanings is guided by a formal notion of signature morphism which, combining with the formal description of pictures, enforces a structural match between pictures and their represented. We also discuss the various issues brought out by this algebra approach.

*I did this work under the auspices of the ERCIM Computer Graphics Network funded under the CEC HCM Programme.

1 Introduction

If one constructs a pictorial representation of some domain, it is necessary to have effective methods for constructing the relationship between aspects of the pictures and the corresponding aspects of the application domain. The use of a picture specification language [4] — the normal method in a computational setting for specifying new classes of pictures according to the requirements of the application domain— is not based on an understanding of the cognitive issues involved in the semantics of pictures and in the use of the pictures for a better grasp of the application domain and for manipulating the domain. There are no rules to guide users in giving a proper interpretation to pictures. For instance, there is no standard reasoning by which one can throw doubt on the wisdom of a specification like: *John loves Mary* is represented by a *straight line*. It seems we want to enforce a structural relationship between the picture and what it represents, so that the structure that is recognised by the user in the picture reflects the structure of the application domain and that the mapping is natural in that the user easily recognises its principles.

These issues are vital in generating visual programming environments [1] [7], but they are not limited to this area or to visual representation on the computer as such. These are but special cases of knowing objects by means of other objects which are employed as a metaphor for them. Indurkhya's theory on metaphor [5] explains the cognitive function of a metaphor in two steps: first, the construction of a correspondence (an isomorphism or a homomorphism) between one domain and the domain it is compared to in the metaphor. The particular structure that this imposes on the first domain is the "concept" through which we know the first domain, Second, the operations which are available on the second domain construct through the correspondence similar operations on the domain that is known in the metaphor. Such metaphors are important in science: e.g. the comparison of gas with balls bouncing in vacuum, atoms compared to the solar system etc. What makes visual languages on the computer have a special status is that here we have the possibility of effectively manipulating the graphical object which can be connected to changes in the application domain.

We present an approach to picture semantics and visual language interpretation, which is naturally interpreted as a (partial) formalisation of the relation which a metaphor constructs between a picture and what it represents. Our intuition is that there exists an analogy between a picture and its meaning if the picture can help people in understanding the meaning represented by it. The match between the syntactic structures of the picture and what it represents is part of the analogy. Therefore looking for a well-structured framework for picture description languages is the most basic step towards a syntax-directed approach to picture semantics. We use order-sorted algebra [3] to specify graphical domains and application domains and signature morphisms to express the relationship between pictures and their meanings [2]. These methods are also useful in the study of how an interpreted picture is used in visual reasoning [8].

An algebraic approach to picture description [6] and an algebraic approach to pictures in the study of metaphors [5] provide inspiration for our approach. Our approach makes it possible to implement systems to support visual reasoning (e.g. see GAR in [8]) and it is helpful for implementing systems which can generate visual reasoning and visual programming environments.

For more details about reasoning with graphical representations, see [8], [9] and [10].

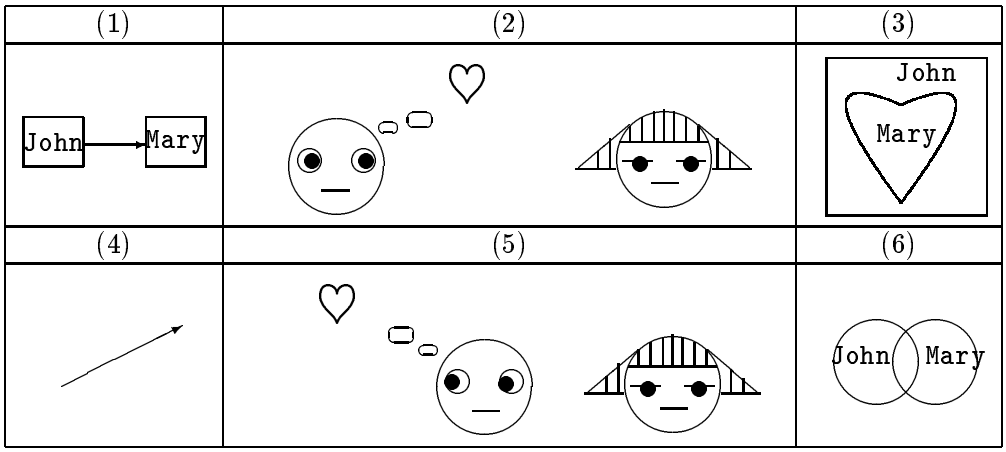


Figure 1: Example graphical representations of *John loves Mary.*

2 Graphical representations as analogies

If a picture helps people to understand the subject matter represented by the picture, usually there is an analogy between the picture and its subject matter. Understanding this analogy depends on various aspects. Here we attempt to compare the structural similarities between the different subject matters and the pictures that represent them. We believe that structural similarity plays an important role in determining what is a good graphical representation. Consider graphical representations of the English sentence *John loves Mary.* The pictures in Figure 1 (1) (2) (3) represent this sentence properly, while the pictures in Figure 1 (4) (5) (6) do not.

The sentence *John loves Mary* has three components, *John, Mary* and *loves. John* and *Mary* are two objects of the same kind, i.e. persons. *loves* is a relational verb which relates John and Mary, making John the subject and Mary the object of *loves.* The picture in Figure 1 (1) has a similar structure, where boxes containing labels correspond to objects of the same kind. The arrow represents the verb *loves.* The spatial relationship *arrow-connect* between the two boxes corresponds to the love relation between John and Mary. The same holds in the pictures in Figure 1 (2) (3) and we can see a certain match between the structure of the sentence and the structures of the pictures. However, there is no proper structural match between the sentence and the pictures in Figure 1 (4) (5) (6). The arrow in Figure 1 (4) may represent the love relation but it can only represent the abstract concept *love* and not the instance of love reported in the sentence. In this representation, only one of the components of the sentence is represented. The picture in Figure 1 (5) can represent that John is in love, but not that he is so with Mary: The spatial relation *heart-thought-direction-and-eye-direction* starts from John but does not end with Mary. It looks more like a representation of *John loves somebody.* In this example, although all three components are represented, there is no match between the spatial relation and the relation in the sentence. Although the picture in Figure 1 (6) matches the sentence in every detail, unfortunately it also suggests that Mary loves John. Because the the spatial relation *overlapping* is symmetric, but the relation *love* is not.

The above examples lead us to consider a syntax-directed approach to picture semantics.

In this approach, a graphical domain (the pictures) is structured and so is the application domain (the subject matter) and with a mapping between the two structured domains. In this paper, we first present a structure for building picture description languages. The same structure can be assumed for the application domain description languages. Second, we define an interpretation which associates pictures with their subject matter. Third, we describe a syntax-directed approach to specify picture semantics and finally, we tentatively discuss an extension of picture semantics to the interpretation of visual languages.

3 Picture description languages

In order to find syntactic similarities between pictures and the things they represent, we need, first of all, to understand the syntactic structure of a picture used in visual communication. This means that we have to look for a proper framework for picture description languages. The framework should provide a useful kind of structure which reflects the various features of the pictures that are used in visual communication.

We consider a picture description language consisting of a *graphical signature* and a *graphical theory*. The graphical signature provides the symbols to generate expressions of a picture description language and the graphical theory gives geometrical meanings to the symbols in the signature.

Graphical signature

A graphical signature consists of a set of graphical sorts with a partial order over it, a set of graphical function symbols and a set of graphical predicate symbols.

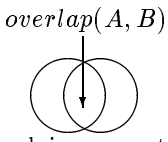
Graphical sorts ($\mathcal{S}$): $\mathcal{S}$ is a set of graphical sorts. Graphical objects are divided into many sorts, such as $Circle$, $Line$, $Arrow$ etc. Furthermore, sorts are divided into two categories according to the way in which people use pictures in communication.
1. *Normal sorts* ($\mathcal{S}_N$): Example normal sorts are $Circle$, $Square$ etc. whose objects usually represent objects in an application domain. For instance, a circle is used to represent a set.
2. *Relational sorts* ($\mathcal{S}_R$): Example relational sorts are $Arrow$, $Cross$, $Tick$ whose objects usually represent the names of predicates and they often appear together with other graphical objects to form certain spatial relations to represent relations in the application domain. For example, crosses often mean negation. As the logical connective $\neg$ must be followed by a predicate to form a well-formed formula which means that the predicate is not true, a cross on a picture usually means that what is represented by the picture is not true. For instance, if we put a cross on the arrow part in the picture in Figure 1 (1), that picture may represent that John *does not love* Mary. If it is put on the box labeled with John, the picture may represent that *it is not John* who loves Mary. Objects in relational sorts may also be used without forming any spatial relations with other objects together, e.g. just a cross. In such cases, they represent abstract concepts, such as *negation, direction.*

The intersection between $\mathcal{S}_N$ and $\mathcal{S}_R$ is not necessarily empty. Some objects can be both normal and relational. For instance, a line connecting two circles may represent a road between two cities or represent two persons who married to each other. The line is used as a normal object in the former case and a relational object in the latter.

A partial order ($\leq$): There is a partial order relation over the graphical sorts, which characterises the subsort relation between sorts. For instance, $Square$ is a subsort of $Rectangle$

($Square \leq Rectangle$) in the sense that all the properties satisfied by rectangles must be satisfied by square. A clear subsort relation can be of help in understanding the geometrical meanings of graphical objects and in giving a semantics to the pictures. For instance, in the use of Venn Diagrams for set theory, both circles and closed curves represent sets. If the subsort relation has been specified, we need only to point out that closed curves represent sets, from which it naturally follows that circles represent sets since $Circle$ is a subsort of $Closed\text{-}curve$. It also helps in defining graphical functions and predicates (see below).

Graphical functions ($\mathcal{F}$): Graphical functions are the possible operations over graphical objects. They are used to build the terms of the picture description language. We classify graphical operations into four categories according to their features.
1. *Constants* ($\mathcal{F}_C$): Constants (i.e. nullary function symbols) represent basic graphical objects. For instance, $C : Circle$ represent a circle.
2. *Natural functions* ($\mathcal{F}_N$): Natural functions represent emergent graphical objects. For instance, applying the function $overlap : ClosedCurve \times ClosedCurve \rightarrow ClosedCurve$ to two overlapping circles represented by two constants A and B, we obtain a new term $overlap(A, B)$ which represents the emergent closed curve.



3. *Artificial functions* ($\mathcal{F}_A$): By applying an artificial graphical function to a picture, new graphical objects (which do not exist in the picture) may occur and other graphical objects (which exist in the picture) may disappear. For instance, two forces a and b apply on an object. We want to see the resultant force on the object. Suppose we use a box to represent the object, and arrows to represent the forces. Then we need a graphical operation *diagonal* which creates the diagonal line to represent the resultant force. *diagonal* is an artificial graphical function, since the graphical object represented by the term $diagonal(a, b)$ does not exist in the original picture.

4. *Attribute functions* ($\mathcal{F}_{At}$): The terms generated by attribute functions do not represent graphical objects but their attributes. Example attribute functions are: $length : Line \rightarrow Real$, $area : Circle \rightarrow Real$, $colour : ColourRectangle \rightarrow Color$, etc. which calculates the length of a line, the area of a circle and finds the colour of a coloured rectangle.

Graphical predicates ($\mathcal{P}$): Graphical predicates are used to generate formulas (atomic formulas) which represent the spatial properties of (spatial relations between) graphical objects. For instance, an atomic formula $in(a, A)$ (generated by applying $in : Point \times Circle$ to point a and circle A) represents that point a is inside circle A.

With the partial order relation over the sorts, a function symbol with principal type $s_1 \times ... \times s_n \rightarrow s$ also has $s'_1 \times ... \times s'_n \rightarrow s'$ as its type, if $s_i \geq s'_i$ and $s \leq s'$. The same happens with predicate symbols. For instance, if in the signature there is an attribute function symbol $area : Closure \rightarrow Real$, then the function $area$ can be applied to closed

Σ	$\mathcal{S}$	$\mathcal{S}_N$	$Rectangle,\ Polygon,\ Circle,\ ClosedCurve,\ Vector,\ TripVector...$
		$\mathcal{S}_P$	$Vector,\ Cross,\ Tick,\ ...$
	$\leq$		$Rectangle \leq Polygon,\ ...$
	$\mathcal{F}$	$\mathcal{F}_C$	$R_1, R_2, ... : Rectangle,\ P_1, P_2, ...\ Polygon,\ ...$
		$\mathcal{F}_N$	$polygon : Rectangle \times Rectangle \rightarrow Polygon,\ ...$
		$\mathcal{F}_A$	$diagonal : Vector \times Vector \rightarrow TripVector,\ ...$
		$\mathcal{F}_{At}$	$area1 : Polygon \rightarrow Real,\ area2 : ClosedCurve \rightarrow Real,\ ...$
	$\mathcal{P}$		$ac : Rectangle \times Vector \times Rectangle,\ cv : Cross \times Vector,\ ...$

Figure 2: An example graphical signature.

curves, circles, polygons, triangles etc. whose sorts are subsorts of $Closure$, and returns their areas.

The above gives the structure of a graphical signature. Following this structure, one can build different graphical signatures. Figure 2 gives an example graphical signature.

Graphical theory (graphical inference)

A graphical signature presents the syntax of the picture description language. In the above explanation of the picture description language, we pretended that there was a 'common-sense' understanding of the graphical sorts, functions and predicates. The meanings of the symbols and expressions in the language can be completely determined by the associated graphical inference. Graphical inference is used to compute the graphical objects formed by graphical operations such as *overlap* and to infer the properties of graphical objects in a picture (e.g. whether a point is inside a circle). In practice, graphical inference is realised by geometrical algorithms. In other words, graphical operations and predicates are implemented by programs which give an (operational) semantics to graphical expressions in the language. A theoretical characterisation of graphical inference can be obtained in different ways, for instance by an axiomatic semantics, i.e. a logical characterisation of the general properties of all pictures. For this, we assume that graphical inference is axiomatisable by a logical (geometrical) theory over the graphical signature, called *the graphical theory* of the picture description language. Let Σ be the graphical signature of a picture description language. Then the graphical theory $\mathcal{T}$ is a set of logical formulas over Σ which is consistent and closed under the consequence relation of the underlying logical system and characterises graphical inference.

4 Mappings between two structures

In the last section, we presented a structure for picture description languages. Now, we assume that an application domain language has the same kind of structure, i.e. there is an (application domain) signature and an (application domain) theory reflecting the natural structure of the application domain. Order-sorted algebra's are extremely general, so any application domain can be formalised as one, but it is not a trivial assumption that the application domain can be formalized in such a way that it (or a subalgebra) corresponds with a given graphical algebra. Only when the application algebra (or a subalgebra of its

polynomial closure) has an isomorphic signature we can give a signature morphism[2]. A signature morphism (an interpretation) maps normal sorts to normal sorts, relational sorts to relational sorts and preserves the partial order relation, maps each kind of function (relation) symbols to the correponding kind of function (relation) symbols and preserves the types of the function (relation) symbols. See the illustration in Figure 3. Under a signature morphism, we can then define whether the picture is a good representation of the application domain.

First of all we want that the picture is a representation of the application domain, i.e. all the facts in the application domain that are expressible in the signature correspond to pictorial facts in the picture. If this is not the case, the user will infer from the absence of the pictorial fact to the absence of the application fact, e.g. because there is no object of John's love to the conclusion that there is no object of John's love. (John's love is ideal or it is unknown who she is).

Second, we want that no facts can be read off from the picture that are not in the application domain. This can even happen to representations. If we represent love by a symmetric graphical relation, the pictur can correctly represent that John loves Mary, but will *ipso facto* also represent that Mary loves John, which can be false.

Both properties define the notion of a *good representation*[1].

Good representations may still be bad as they can be unnatural. There is good sense in using natural similarities and conventional correspondences between the graphical domain and the application domain to underlie the signature morphism. For example, size of representation is a better choice for representing the size of the represented object than the position on the x-axis. Such considerations fall outside the scope of our methods, but they are extremely important, as these natural relations make it possible for the user to guess the nature of the signature morphism without being explicitly informed of it.

A tentative formulation of what happens in metaphor interpretation could be that we have two domains, two signatures and a signature morphism under which one domain is a good representation of the other. Nature and convention are as important here as in the interpretation of graphical representations as these are the basis for finding the structural correspondence for the interpreting subjects.

Consider the picture in Figure 1 (1) and the sentence *John loves Mary.* A language for describing this picture and a language for describing the sentence are necessary to have the signatures in Figure 4.

Guided by the mapping structure in Figure 3, an interpretation between these two signatures can be given as follows:
Box $\mapsto$ Person, Arrow $\mapsto$ Love $B_1 \mapsto$ John, $B_2 \mapsto$ Mary, $A \mapsto$ Loves a-connect $\mapsto$ love

Given an interpretation, it is extended to terms and formulas over the graphical signature. For example, the above interpretation will be extended to formulas like: a-connect(B_1, B_2), i.e. $\mathcal{I}$(a-connect$(B_1, B_2)) = \mathcal{I}$(a-connect)$(\mathcal{I}(B_1), \mathcal{I}(B_2))$ = love(John,Mary).

An interpretation is usually only a partial mapping between signatures. A graphical signature may have many sorts, functions and predicates. When we use pictures, however, we may use only some parts of the signature, i.e. we move to a subsignature of the graphical signature for the purpose at hand. Suppose we have in the graphical signature an attribute function for calculating the area of a box. In representing the sentence *John loves Mary*, this size is not relevant and should be ignored. The same holds for the categories in a signature. In the *John loves Mary* example, the partial order and natural, artificial and attributes functions are not used.

[1] In [8] also weaker notions of representation are considered.

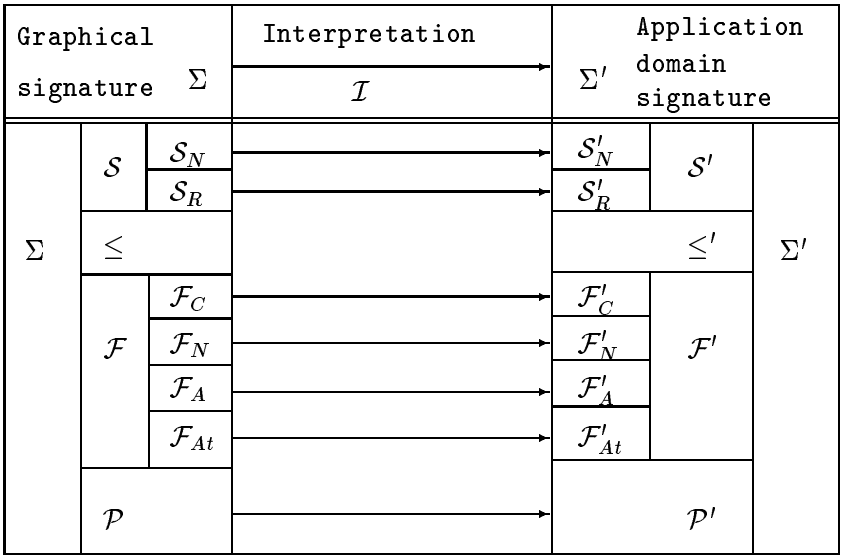


Figure 3: An illustration of an interpretation.

Σ for the picture in Figure 1 (1).

$\mathcal{S}_N$	Box, ...
$\mathcal{S}_R$	Arrow, ...
$\leq$	
$\mathcal{F}_C$	B_1, B_2: Box, A: Arrow, ..
	...
$\mathcal{P}$	a-connect: Box $\times$ Box, ...

Σ' for *John loves Mary*

$\mathcal{S}'_N$	Person, ...
$\mathcal{S}'_R$	Love, ...
$\leq'$	
$\mathcal{F}'_C$	John, Mary : Person, Loves : Love, ..
	...
$\mathcal{P}'$	love : Person $\times$ Person, ...

Figure 4: a-connect means two boxes are connected by an arrow.

5 Syntax directed meaning specifications

The framework for picture description languages and the notion of interpretation presented in the last two sections naturally leads to an approach to picture semantics. There are the following components in this approach: a picture description language, an interface to help the user to build up her application domain signature and an interface to guide the user to choose graphical representations.

In the discussion, we can assume that the signature of the picture description language is rich enough to satisfy all possible requirements. In practice, the graphical signature is adapted to specific applications. The graphical theory of the picture description language corresponds to a graphical inference engine which implements the graphical functions (e.g. the computation of the overlap of two circles) and computes the truth-values of the graphical predicates.

In order to construct the application domain signature, the system should help the user

to, first, classify object classes, functions and predicates in her application domain. If an entity is a function (predicate), then it asks her to give its type (domain). For instance, consider set theory as the application domain. *Sets*, *elements* are two classes of objects, $\cap$, $\cup$, *cardinality* are functions over sets and $\in$, $\subset$ are predicates. The first classification fills $\mathcal{S}'$, $\mathcal{F}'$ and $\mathcal{P}'$ in the domain signature Σ'. Then, the system should help her to classify different kinds of sorts and functions. For instance, we classify $\cap$, $\cup$ and *cardinality* into two kinds: $\cap$ and $\cup$ are natural functions and *cardinality* is an attribute function.

Since the application domain signature is given, the user can choose graphical representations for the subject matter. Each time, the system provides the possible graphical entities to the user when she looks for a graphical representation. For instance, if the user wants a graphical representation for the sort Set, then the system shows all the *normal* graphical sorts to her because Set is a *normal* sort in the application domain signature. Suppose she selects $Circle$ for Set and then $Point$ for $Element$. Now when she wants a graphical representation for the membership relation $\in$, the system, according to the previous interpretation ($Circle$ for Set and $Point$ for $Element$) and the domain of $\in$ ($\in$: $Element \times Set$), shows the existing graphical predicates whose domains are $Point \times Circle$. Suppose the user now wants the natural graphical function *overlap* to represent the set operation $\cap$. To her surprise the system does not show *overlap* to her after she clicked $\cap$ in the application domain signature. The reason is that the type of $\cap$ is: $Set \times Set \rightarrow Set$, according to her previous interpretation (i.e. $Circle$ represents Set), only those graphical functions whose types are $Circle \times Circle \rightarrow Circle$ can be used to represent $\cap$. The type of *overlap* is: $Closed\text{-}curve \times Closed\text{-}curve \rightarrow Closed\text{-}curve$. According to the subsort relation, *overlap* also has $Circle \times Circle \rightarrow Closed\text{-}curve$ as its type, but $Circle \times Circle \rightarrow Circle$ is not its type. In such a case, the user may be allowed to tell the system that *overlap* is what she wants. This prompts the system to compare the types of the graphical functions and to diagnose the problem.

In the last section, an interpretation also included interpreting graphical constants to constants ($\mathcal{F}_C \rightarrow \mathcal{F}'_C$). This is suitable for the understanding and the theoretical study of picture semantics. However, in practice, the interpretation of graphical constants should be postponed to the time when a particular picture is created for visual communication. A graphical object (constant) is interpreted as an object by associating a label to the graphical object. In the example *John loves Mary*, we should only interpret Box to $Person$. When a box is drawn on the screen, a label (John) can be given to the box, which means that this box (whose sort is Box which is interpreted as $Person$) is interpreted as the object $John$ (whose sort is $Person$).

For a predicate (e.g. *love*) in an application domain, the user can either choose a relational sort (e.g. *Arrow*) to represent the name of the predicate and then choose a graphical predicate to represent the predicate (e.g. *love*), or directly choose a graphical predicate to represent it. For the former, when the user wants to select a graphical predicate to represent the predicate (e.g. $love : Person \times Person$), the system will provide all the possible graphical predicates which not only have the matched domains with the domain of the predicate in the application domain, but also are related to the (graphical) relational sort selected before for the name of the predicate (e.g. $arrow\text{-}connect : Box \times Box$), and for the latter, the system just provides all the graphical predicates whose domains match the domain of the predicate in the application domain.

6 Visual Languages

There is something unusual about the notion of semantics we have considered so far. This comes out well when we compare it with natural language semantics: it is as if we are giving a semantics for a single sentence rather than for the language as such. The problem with semantics for pictures is that if we are speaking of pictures as such there is no uniform semantics in terms of an application structure: the structure allows for many different graphical representations with different meaning assignments. (We saw an example of that in figure 1).

Yet every single picture grasped in a particular way assigns meanings to a class of pictures: those we can obtain by varying things in the picture without disturbing the signature morphism. This variation supports intuitions of the form: if the picture had been so and so, the application domain would have been so and so. A simple example: Let P be a representation of various people spread out over some space. We have labeled icons for the different people and the position of the people icons on the screen reflects their spatial position in some room. Once we grasp this, there are different variations we can study: we can move the people about and we can add and remove people icons. The meaning is that some one has shifted position, that more people have come into the room etc.

What remains constant within the variations is the interpretation of the room, the interpretation of the person icon and the rule which assigns real people to labelled icon: it is the person that has the label as a name.

What we obtain by considering variation is a class of pictures. These can be characterised as the set of picture algebras that have the same sorts as the original picture and that may share some of the individual constants. In addition they share the signature morphism to the application domain. In this sense, a single interpreted diagram determines an interpreted visual language, given a range of allowed variations. Each of the variations determines its own signature and we can take the union of all of these. This gives us the signature with all the infinitely many constants of each given sort. A variation is a (graphically interpreted) algebra for a finite subsignature of this union. The interpretation of graphical sorts in the application domain is inherited from the original diagram. The interpretation of constants may be given by a rule (as in our example: an icon is a constant for the person whose name is the label) but can also be underspecified. Here the constants have an indefinite meaning and it may be that the same constant has a different interpretation in a variation.

The algebras can be given as the set of labelled person icons with an attribute function $Pos : Icon \rightarrow Real \times Real$. They are all algebras with a signature Σ that is a subset of a signature $\sum$ which contains all labelled icons and the Pos function.

The application domain is similarly given by the set of algebras which have different people, a name function for people and an attribute function Pos' giving the position of the people in the room. These algebras also share part of their signature, and vary in the objects.

The signature morphisms are constant over the interpretation of the room, the interpretation of the Pos attribute function and in the interpretation of the icons as people and icon labels as their names. They vary however in the set of icons over which they are defined and in the values they give to the icons.

Formally, an interpreted visual language can be defined as a set of triples $< D, F, \varphi >$, where D is a diagram, F an application structure and φ the signature morphism relating the two. But this doesn not give a notion of meaning for the language. For this we require a recipee that given a diagram can give us the depicted application structure and the morphism relating the diagram and the structure.

A simple way to describe the language we are considering is as an infinite signature $\sum$ containing all possible people icons, a similar signature for the application domain and a signature morphism φ between those. The expressions of the formal language are then the algebras A with a finite subsignature Σ. We can restrict φ to the subsignature Σ for a given expression A. The application structure depicted by the expression A, is then the subalgebra B with the signature $\varphi(\Sigma)$, with φ restricted to Σ the morphism linking A and B.

Let us consider one more example: diagrams for deterministic finite state machines (D-FSMs). A D-FSM for an alfabet Σ can be constructed as a structure

$$< T, R_{c_1}, \ldots, R_{c_n}, start, success >$$

where T is finite set, there is a relation R_{c_i} for each character c_i in Σ, and *start* and *success* are predicates over T. We require that there is a single start element, that from each state there is an R_{c_i} successor and that every state can be reached from the start state through the R_{c_i}'s.

Diagrams for these can be given as arrangements from character occurrences (from Σ and the characters $*$ and $<$), arrows (pointed open curves) and circles. This gives us three sorts:

$\mathcal{S}$	$charoc,\ circle,\ arrow.$

In addition we require some predicates. Arrows can be labelled by characters, circles can contain a character and arrows can go from one circle to another. Moreover we need predicates $c(x)$ stating that x is an instance of character c for each of the allowed characters.

$\mathcal{P}$	$label:$	$charoc \times arrow$
	$contain:$	$charoc \times circle$
	$connect:$	$arrow \times circle \times circle$
	$c:$	$charoc$

A particular diagram is given by a set of instances from the sorts, which give the full signature of the diagram, which codes them a set of constants. A subdiagram of a given diagram is a diagram with a subset of the constants.

The signature morphism is now based on the following subsignature of the polymorphic closure of the diagram signature.

1. the circle constants
2. for each character from the alfabet, the predicate $\lambda x \lambda y \exists z \exists v(c(v) \wedge label(v,z) \wedge connect(z,x,y))$
3. $\lambda x \exists v(<(v) \wedge contain(v,x))$
4. $\lambda x \exists v(*(v) \wedge contain(v,x))$

The signature morphism must map each circle constant to an element of the D-FSM, the predicates in (2) to the relations R_c, the predicate in (3) to *start* and the predicate in (4) to *success*. We can define the relation: diagram D denotes the D-FSM F by demanding that: 1. there exists a signature morphism φ between the signature given in 1-4 between D and F. 2. φ is one-one between the circle constants and the domain of F. 3. there is no subdiagram D' of D that is also related by φ to F.

The first demand can be taken as an expression of the conventions of the drawing style. The second makes sure that for every circle there is one state and for every state there is one circle, a particularity of this kind of diagrams. The third demand makes sure there are no uninterpreted elements in the diagrams like unlabelled arrows, free-floating character occurrences, arrows that do not connect.

Notice that we have defined no constraints so far on the finite state diagrams: any arrangements of elements of the appropriate sorts will be kandidates for interpretation. But we get them back by demanding that a proper diagram denotes a D-FSM. This will enforce the axioms of D-FSMs and the absence of loose elements.

There is another kind of constraint that does not directly follow from the semantics. It is customary and reasonable to demand that arrows do not cross each other and the circles, that the circles do not overlap, that the character occurrences are readable etc. These constraints follow from the cognitive function of the diagram. A user building a D-FSM by drawing a diagram must be able to read the result and must be able to show it to others. Non-observance of such constraints will increase the difficulty for the user (and for the machine that is presented with the graphics in e.g. bitmap format) for grasping the structure of the diagram and thereby will inhibit the understanding. Crossing arrows will allow the construction by the interpreter of different arrows, an arrow crossing a circle can lead to different views concerning what it connects etc. This will thereby lead to different interpretations of the same diagram and so inhibit the understanding of the visual metaphor by the interpreter.

References

[1] B. Backlund and O. Hagsand. Generation of visual language-oriented design environments. *Journal of Visual Language and Computing*, 1:333–354, 1990.

[2] J. Goguen and R. Burstall. Introducing institutions. *LNCS 164*, 1984.

[3] J. Goguen and J. Meseguer. Order-sorted algebra 1: Equational deduction for multiple inheritance, polymorphism, overloading and partial operations. Technical report SRI-CSL-89-10, SRI International, 1989.

[4] Richard Helm and Kim Marriott. A declarative specification and semantics for visual languages. *Journal of Visual Language and Computing*, 2:311–331, 1991.

[5] Bipin Indurkhya. *Metaphor and Cognition*, volume 13 of *Studies in cognitive systems.* Kluwer academic publishers, 1992.

[6] L. A. Pineda. *GRAFLOG: a Theory of Semantics for Graphics with Applications to Human-Computer Interaction and CAD Systems.* PhD thesis, University of Edinburgh, 1990.

[7] Susan M. Uskudarli. Generating visual editors for formally specified languages. In *IEEE Symposium on Visual Languages*, pages 278–285, St. Louis, Missouri, Oct. 1994.

[8] Dejuan Wang. *Studies on the formal semantics of pictures.* PhD thesis, University of Amsterdam, 1995. ILLC Dissertation Series 1995-4.

[9] Dejuan Wang and John Lee. Visual reasoning: its formal semantics and applications. *Journal of Visual Language and Computing*, 4:327–356, 1993.

[10] Dejuan Wang, John Lee, and Henk Zeevat. Reasoning with diagrammatical representations. In N. Hari. Narayanan Janice Glasgow and B. Chandrasekaran, editors, *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, pages 339–393. AAAI press/The MIT press, 1995.