To appear in the Proceedings of the APPIA-GULP-PRODE'98, Joint Conference on Declarative Programming, La Coruña, Spain, July 1998.

Visual Logic Programming by means of Diagram Transformations

Jordi Puigsegur[1,2] **Jaume Agustí**[1]

{jpf,agusti}@iiia.csic.es

1 Institut d'Investigació en Intel·ligència Artificial – CSIC
Campus UAB, 08193 Bellaterra, Catalonia (Spain), EU

2 Visual Inference Laboratory – Indiana University
Lindley Hall #215, Bloomington, IN 47405, USA

Abstract

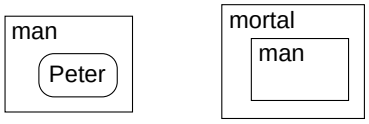
We believe that the pragmatics and understanding of formal logic and also declarative programming languages are sensible to the type of syntax used. Our goal is to study how to apply the new developments in the field of diagrammatic reasoning to declarative programming languages. In this paper we summarize the work done up to now in a visual logic language developped at the IIIA. We also attempt a complete formalization of its syntax, semantics and inference system. We claim that our visual syntax and operational semantics have a higher degree of homomorphism with respect to the mathematical semantics of the language than in conventional textual languages. Finally, we study two interesting new features: the ability of intuitively keeping track of the proof and the possibility to represent several solutions to the query, both using a single diagram.

1 Introduction

Recent advances emphasize the importance of the syntax in logical systems and programming languages. Barwise and Etchemendy [5] emphasize the importance of using homomorphic notations, i.e. notations (often diagrammatic) that are closer to what they represent than the usual predicate-based logic languages. They show how using homomorphic (and heterogeneous) syntax can improve the understanding of the system and ease the reasoning process. Our goal is to apply these ideas to the field of declarative programming and automated reasoning. We believe that the pragmatics and understanding of formal logic and declarative programming languages are sensible to the syntax used.

In [16, 1] we presented a visual logic programming language based on Venn/Euler diagrams, directed acyclic graphs (DAGs) and graphical containment (inspired by David Harel Higraphs [10]), and we formalized it as a front-end to conventional logic programming languages like Prolog. This visual language is based on a set theoretical approach to predicate logic and its scope is similar to Horn Logic. Predicates are represented by sets of elements: their abstractions or sets of elements that satisfy them. Basic facts are represented as the membership of an element into a set corresponding to a predicate, and predicate implication is represented as the inclusion of one set into another set, both representing predicates. The following two diagrams represent respectively the basic fact

"*Peter is a man*" (in FOPC: $man(Peter)$) and the implication "*all men are mortal*" (in FOPC: $\forall x\ man(x) \rightarrow mortal(x)$):



In [17, 3] we showed the possibility of a fully visual operational semantics for this visual language, showing that our visual syntax not only can be an alternative formal notation which emphasizes some semantic and pragmatic features of logical statements, but it could be useful to conduct visual inferences. We claim that this operational semantics is also closer to the intuitive meaning of the diagrams than traditional resolution is to conventional textual logic languages. The inference system of our visual language is based in diagram transformations involving graphical containments, thus being closer to its mathematical semantics which is based mainly in set inclusion.

In this paper we summarize the work done up to now in this visual logical formalism, and we attempt a complete formalization of its syntax (Section 2), semantics (Section 3) and inference system (Section 4). Finally, we study two interesting features provided by its visual nature: first the ability of intuitively keeping track, within a diagram, of the proof that is built while solving a query, and, second, the possibility to represent within a unique diagram several solutions to the query.

2 Syntax

As we pointed out in the Introduction, the visual syntax of our language is based on a topological diagrammatic notation which combines Venn/Euler-like diagrams and DAGs.

The basic syntactic elements (or visual primitives) of our visual language are square boxes, rounded boxes, circles, arrows, lines, function symbols (function symbols of arity 0 are also called constant symbols) and predicate symbols. Circles and rounded boxes are of fixed size. Square boxes are of different sizes, always bigger than circles and rounded boxes. These basic syntactic elements are then combined to form visual terms (Section 2.1) and visual predicates (Section 2.2) which correspond to elements and sets respectively in the semantic interpretation. Visual terms and visual predicates are combined using the graphical containment relation obtaining visual literals (Section 2.3), or basic constituents of diagrams. Finally we obtain diagrams (Section 2.4) —which are the equivalent to formulas in conventional textual logic languages— by enclosing with a square box a collection of visual literals.

2.1 Visual Terms

Visual terms are DAGs built up using circles and rounded boxes with enclosed function symbols. Variables do not need to be assigned a name and are represented as circles. Variable co-reference is performed using sequences of circles or visual term sharing.

Definition 1 (Visual Term) *A well formed visual term is:*

1. *a circle or a sequence of circles joined by lines.*
2. *a rounded box with a constant symbol inside it.*
3. *a compound diagrammatic term (a DAG) constructed with:*

- *a rounded box*
- *a function symbol of arity m placed inside the rounded box*
- *n visual terms t_i ($n \geq 1$)*
- *m arrows ($m \geq n$), with an optional label, each one going from one t_i to the rounded box, in such a way that there are no cycles and each t_i has at least one arrow.*

4. nothing else is a well formed visual term.

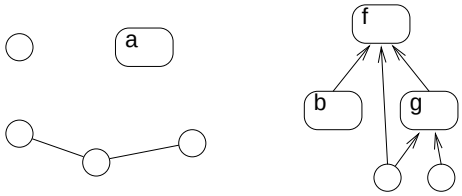


Figure 1: Examples of w.f. Visual Terms

In Fig. 1 we find examples of the four different types of visual terms of our language.

2.2 Visual Predicates

Predicates are represented as labeled square boxes, which are graphical representations of the set of elements that satisfy these predicates. I.e. what in mathematics is usually known as *predicate abstractions*. When the predicate has more than one argument, one of them is selected (by convention the last one) as the range of the set while the rest are explicitly shown by means of arrows. Since in the visual representation there is no evident ordering of arguments, arrows are optionally labeled to identify the arguments. In Section 3 there is a more precise definition of the semantics of a visual predicate. Let us now define the syntax of a visual predicate:

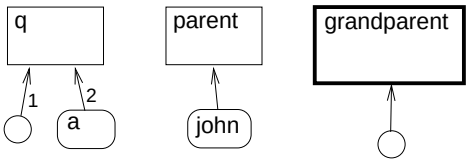


Figure 2: Examples of w.f. Visual Predicates

Definition 2 (Visual Predicate) *A well formed visual predicate, is composed of:*

- *a square box (drawn using either thick or thin lines)*
- *a predicate symbol of arity m ($m \geq 1$) placed in one of the corners of the square box*
- *n visual terms t_i ($n \geq 0$)*
- *$m - 1$ arrows ($m - 1 \geq n$), with an optional label, each one going from one t_i to the square box, in a way that from all t_i departs at least one arrow*

In Fig. 2 we find examples of visual predicates. For instance, the first visual literal represents the 3-ary predicate q, and its square box stands for the set $\{w|q(x, a, w)\}$. The other two visual literals intuitively represent the set of *parents* of *john* and the set of *grandparents* of *someone*.

2.3 Visual Literals

First we define the containment relation between visual terms and visual predicates, and between visual predicates that we use later to define visual literals.

Definition 3 (Containment Relation) *Let VP be the set of visual predicates and VT the set of visual terms. The containment relation ($\sqsubseteq$) is defined as $\sqsubseteq = \sqsubseteq_1 \cup \sqsubseteq_2$ where $\sqsubseteq_1$ and $\sqsubseteq_2$ are:*

1. $\sqsubseteq_1 \subseteq VP \times VP$
   $v_1 \in VP$ is contained in $v_2 \in VP$ ($v_1 \sqsubseteq_1 v_2$) iff the square box of v_1 is contained in the square box of v_2.
2. $\sqsubseteq_2 \subseteq VT \times VP$
   $t \in VT$ is contained in $v \in VP$ ($t \sqsubseteq_2 v$) iff
   - *when t is a circle, the circle is contained in the square box of v.*
   - *when t is a sequence of circles, one of the circles is contained in the square box of v.*
   - *when t is a compound term or a constant, its root rounded box is contained in the square box of v.*

Visual literals are the basic syntactic construction used to build diagrams. They represent a basic fact of our visual language: an inclusion of an element or a set into another set, expressed as a graphical containment. There are two types of visual literals: condition visual literals and conclusion visual literals, to distinguish the role they play in a predicate definition.

Definition 4 (Condition Visual Literal) *A well formed condition visual literal l is composed of*

- *a visual predicate v with its square box drawn using thin lines.*
- *One visual terms t contained in v ($t \sqsubseteq v$).*

Condition visual literals consist always on a containment of one and only one visual term into a visual predicate drawn using thin lines. In Fig. 3 we find examples of condition visual literals. The first visual literal represents the fact that the term *a* satisfies predicate *p*, the second that *john* is a *person*, and the third visual literal represents that *someone* is *parent* of *ann*.

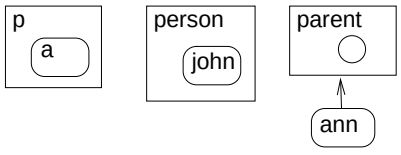


Figure 3: Examples of w.f. Condition Visual Lit.

Definition 5 (Conclusion Visual Literal)
A well formed conclusion visual literal l is composed of

- *a visual predicate g with its square box drawn using thick lines, called the goal visual predicate.*
- *n visual terms t_i contained in g ($t_i \sqsubseteq g$).*
- *m visual predicates v_i contained in g ($v_i \sqsubseteq g$).*

such that $n + m \geq 1$ and there are no other containments other than the ones indicated above.

Conclusion visual literals express the containment of one or more visual terms or visual predicates into another visual predicate (drawn with thick lines). In Fig. 4 we find examples of conclusion visual literals. The first visual literal expresses the fact that *bob* and *ann* are *parents* of *charly*, while the second represents the fact that the *parents* of *bob* are *grandparents* of *charly*.

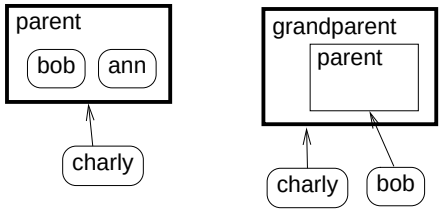


Figure 4: Examples of w.f. Conclusion Visual Lit.

2.4 Diagrams

Diagrams are the basic elements of our visual programs: they correspond to a formula in logic or a clause in textual logic programming. A diagram is defined as a collection of visual literals enclosed in a square box delimiting its syntactic scope. Different literals of the same diagram may share common visual subterms. In fact this diagrammatic notation inherits from DAGs the facility of subterm (subDAG) sharing.

There are two types of diagrams: definition diagrams and query diagrams. A visual program is defined as a collection of definition diagrams. Query diagrams are used to pose queries to these visual programs.

2.4.1 Definition Diagrams

A definition diagram is composed of one conclusion visual literal and (optionally) a set of condition visual literals. The conclusion visual literal contains the visual predicate being defined (drawn with thick lines), while the condition visual literals are the conditions under which the definition is valid. In Section 3 we precisely formalize the semantics of definition diagrams. Let us now formalize the syntax of a definition diagram:

Definition 6 (Definition Diagram) *A well formed definition diagram (d) is composed of:*

1. *a box that circumscribes the syntactic scope of the diagram. I.e. all visual literals of the diagram must be inside this box.*
2. *one conclusion visual literal.*
3. $n \geq 0$ *condition literals.*

such that no containments occur between visual predicates of different visual literals.

In Fig. 5 we find examples of well formed definition diagrams. In the first of them we define predicate q by means of a subset inclusion. Namely we express that $\{w|p(a,w)\} \subseteq \{z|q(b,a,z)\}$, which is equivalent to the implication $p(a,y) \rightarrow q(b,a,y)$. The other two diagrams define *parent* and *grandparent* predicates. The *parent* is defined by giving two elements that belong to the set of parents of *charly*, *bob* and *ann*. The *grandparent* predicate is defined by giving a subset of it, i.e. by stating that the parents of the parents of some person are also grandparents of the same person.

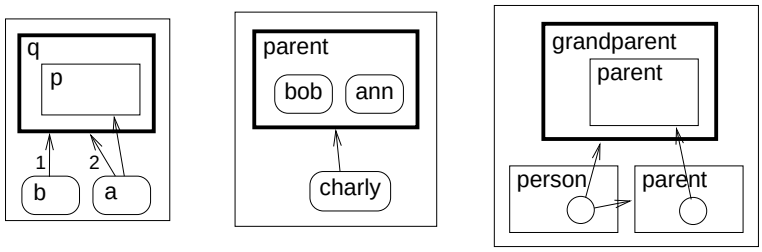


Figure 5: Examples of w.f. Definition Diagrams

2.4.2 Query Diagrams

A query diagram is defined like a definition diagram except that there is no conclusion visual literal. Let us now define the syntax of a query diagram.

Definition 7 (Query Diagram) *A well formed query diagram (q) is composed of:*

1. *a box that circumscribes the syntactic scope of the diagram. I.e. all visual literals of the diagram must be inside this box.*
2. $n \geq 1$ *visual condition literals.*

such that no containments occur between visual predicates of different visual literals.

A query diagram can be seen as an existential query, i.e. a conjunction of inclusions where variables are existentially quantified. In Fig. 6 we find examples of well-formed query diagrams. In the first query diagram we want to know the *parents* of *ann*, namely the in FOPC: $\exists x\ parent(ann, x)$?. The second query diagram asks for *grandparents* of *charly* who are *tall*. Textually this is expressed as the FOPC formula: $\exists x\ tall(x) \wedge grandparent(charly, x)$.

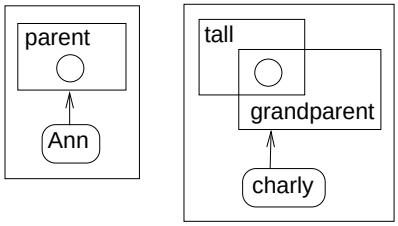


Figure 6: Examples of w.f. Query Diagrams

3 Model Theory

Let us define the models of our diagrams:

Definition 8 (First Order Model) *A model $\mathcal{A}$ is composed of:*

1. *A semantic domain $|\mathcal{A}|$, i.e. the set of elements over which visual terms are mapped to.*
2. *A visual term interpretation function $\phi^{\rho}_{\mathcal{A}}$, that maps every visual term to an element of the semantic domain. $\rho : \mathcal{V} \rightarrow |\mathcal{A}|$ is a mapping from each variable (circle) to an element of the semantic domain (such that all circles of a sequence are mapped to the same element).*
3. *A predicate symbol interpretation function $\varphi_{\mathcal{A}}$ which given a predicate symbol p of arity n returns the set of tuples of elements of the semantic domain that satisfy the predicate ($\varphi_{\mathcal{A}}(p) \subseteq |\mathcal{A}|^n$ where n is the arity of p).*

Let us now define an interpretation function for visual predicates based on the predicate symbol interpretation function ($\varphi_{\mathcal{A}}$):

Definition 9 (Visual Predicate Interpretation) *The visual predicate interpretation function $\psi^{\rho}_{\mathcal{A}}$, that maps every visual predicate into a subset of $|\mathcal{A}|$, is defined as:*

$$\psi^{\rho}_{\mathcal{A}}(v) \;=\; \{z \mid \langle \phi^{\rho}_{\mathcal{A}}(t_1), ..., \phi^{\rho}_{\mathcal{A}}(t_{n-1}), z\rangle \in \varphi_{\mathcal{A}}(p)\}$$

where visual predicate v corresponds to predicate symbol p of arity n ($n \geq 1$), and its arguments are the visual terms $t_1, ..., t_{n-1}$.

Let us now define when $\mathcal{A}$ is a model of a visual literal. $\mathcal{A}$ will be a model of a visual literal when after interpreting the visual terms and visual predicates of the visual literal, the inclusions originated by the graphical containment hold:

Definition 10 (Models of a Cond. Visual Lit.) *$\mathcal{A}$ is a model of a condition visual literal l ($\mathcal{A} \models_{\rho} l$) iff*

$$\phi^{\rho}_{\mathcal{A}}(t) \in \psi^{\rho}_{\mathcal{A}}(v)$$

where v is the visual predicate and t is the visual term contained in v ($t \sqsubseteq v$).

Definition 11 (Models of a Concl. Visual Lit.) *$\mathcal{A}$ is a model of a conclusion visual literal l ($\mathcal{A} \models_{\rho} l$) iff*

$$\forall t_i \;\; \phi^{\rho}_{\mathcal{A}}(t_i) \in \psi^{\rho}_{\mathcal{A}}(g)$$
$$\forall v_j \;\; \psi^{\rho}_{\mathcal{A}}(v_j) \subseteq \psi^{\rho}_{\mathcal{A}}(g)$$

where g is the visual predicate being defined (drawn with thick lines), t_i are the visual terms contained in g ($t_i \sqsubseteq g$) and v_j are the visual predicates contained in g ($v_j \sqsubseteq g$).

Now we define when a first order model $\mathcal{A}$ is a model of a definition diagram. The intuition is that a model $\mathcal{A}$ is a model of a definition diagram if, for all possible variable substitutions, when it is a model of all its condition visual literals then it is also a model of its conclusion visual literal.

Definition 12 (Models of a Definition Diag.) *$\mathcal{A}$ is a model of a definition diagram d (i.e. we say $\mathcal{A} \models d$) iff*

$$\forall \rho \; (\mathcal{A} \models_{\rho} h \;\vee\; \mathcal{A} \not\models_{\rho} l_1 \;\vee\; ... \;\vee\; \mathcal{A} \not\models_{\rho} l_n)$$

where h is the conclusion visual literal and $l_1 ... l_n$ ($n \geq 0$) are the condition visual literals and ρ is a variable substitution function.

Finally we define when $\mathcal{A}$ is a model of a query diagram, the intuition being that $\mathcal{A}$ models of a query diagram when there exists some variable substitution such that it is a model of all its condition visual literals.

Definition 13 (Models of a Query Diagram) *$\mathcal{A}$ is a model of a query diagram q (i.e. we say $\mathcal{A} \models q$) iff*

$$\exists \rho \; (\mathcal{A} \models_{\rho} l_1 \;\wedge\; ... \;\wedge\; \mathcal{A} \models_{\rho} l_n)$$

where h is the conclusion visual literal and $l_1 ... l_n$ ($n \geq 0$) are the condition visual literals and ρ is a variable substitution function.

4 Proof Theory

We are now going to show how to perform visual inferences directly with the diagrams, defining an inference system close to SLD-resolution. Actually we have defined the visual language so that it has an expressive power equivalent to that of Horn clauses, and our inference system will inherit many of the good properties of SLD-resolution.

We want to have an operational semantics easy to understand and close to the intuitive semantics of the diagrams. Therefore the visual inference system we introduce is based on diagram transformations involving containments in a way such that its intrinsic transitivity (the essence of resolution) is obtained by free. We think that our visual language allows a visual operational semantics (or proof theory) that is closer to the language semantics than in the conventional (textual SLD-resolution) case. Other improvements of our diagrammatic approach are that we are able to keep track of the proof and represent various solutions simultaneously in a single diagram.

In Section 4.1 we will define the visual unification procedure. In Section 4.2 we formalize the visual inference rule and introduce the concept of extended query diagram, necessary to perform visual inferences. In Section 4.3 we study how an inference step is performed, formalizing the different diagram transformations that are necessary to perform an inference step.

4.1 Unification

Unification is —like in textual resolution— very important since every visual inference step involves unifying two visual literals: a visual condition literal with a visual conclusion literal.

Definition 14 (Visual Term Unification) *Two visual terms unify when one of the following cases occurs:*

- *Two circles unify and the result is another circle.*
- *A circle and another visual term unify and the result is the visual term. If the circle belongs to a sequence of circles then all the circles are also unified with the visual term.*
- *Two constants unify if their constant symbol is the same.*
- *Two compound visual terms unify iff their root function symbol is the same, and all visual term pairs corresponding to the arguments of both root rounded boxes unify. When there is more than one argument, arrow labels are used to match the arguments. If the arrows are not labeled then an arbitrary order is taken (clock-wise starting at the upper-right corner of the round box).*

Note that since visual terms and visual predicates are based on DAGs, it is not necessary to apply substitutions since variables are already explicitly shared by different visual literals.

Definition 15 (Visual Predicate Unification) *Two visual predicates unify iff their predicate symbol is the same, and all visual term pairs corresponding to the arguments of both predicates unify. When there is more than one argument, arrow labels are used to match the arguments. If the arrows are not labeled then an arbitrary order is taken (clock-wise starting at the upper-right corner of the square box).*

Definition 16 (Visual Literal Unification) *Let l_1 be a condition visual literal (with containment $t \sqsubseteq v$), and l_2 a conclusion visual literal (with containments $t_i \sqsubseteq g, v_j \sqsubseteq g$ where $1 \geq i \geq n$, $1 \geq j \geq m$ and $n + m \geq 1$). l_1 and l_2 unify when:*

1. *the main visual predicates of both literals, v and g, unify.*
2. *One or both of the following cases holds:*
   - *$m \geq 1$. Then new condition visual literals are formed ($t \sqsubseteq v_j$).*
   - *t unifies with some $t_1, ..., t_m$.*

The visual unification of two visual literals embodies two different types of visual reasoning, depending on the type of containment in the conclusion visual literal. Let us show this by means of two examples.

In the example of Fig. 7 the visual predicate corresponding to *parent of charly* is defined by giving two elements contained in it. In this case the unification of the conclusion visual literal defining this visual predicate and the condition visual literal is done by unifying directly the terms contained in both visual literals. Note that the unification of a condition visual literal and a conclusion visual literal may produce a multiple instantiation of a variable. This occurs when the conclusion visual literal has multiple containments and more than one option to instantiate the variable exist. In this situation —as we will show in detail in Section 4.3.1— all different possible instantiations can be put together in the resulting visual literal. In Fig. 7 a multiple instantiation is performed, instantiating a variable with two constants: *bob* and *ann*.

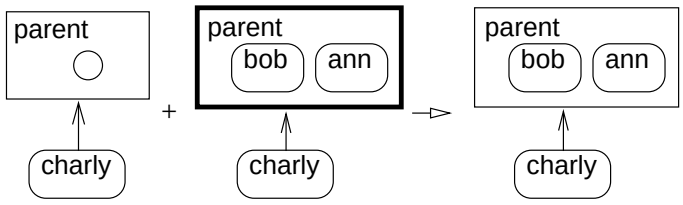


Figure 7: Visual Literal Unification: Visual Term containment

The other possible case is when the visual predicate of the conclusion visual literal is defined by giving a visual predicate contained in it. I.e. the visual predicate is defined by giving a subset of the elements which satisfy the predicate. In Fig. 8 we find an example of this case where the visual predicate *grandparents of charly* is defined by stating that the visual predicate *parents of bob* is contained in it. Thus when we unify both visual literals of Fig. 8 then we create a new visual literal: we try to prove that *john* is a *grandparent of charly* by first proving that is a *parent of bob*. This way of reasoning captures the intrinsic transitivity of set inclusion (and also that of implication).

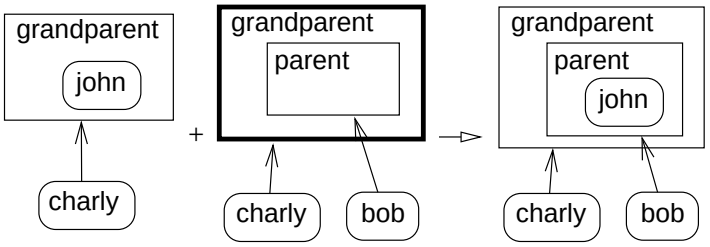


Figure 8: Visual Literal Unification: Visual Predicate containment

4.2 Visual Inference

Now we introduce and formalize the core of the inference system. One visual inference rule is similar to SLD-Resolution: 1) **L**inear – inferences are always performed between a query diagram and a definition diagram, obtaining a new query diagram. 2) **D**efinite – Definition diagrams are equivalent to Horn clauses. 3) **S**election – We do not force any order over the next visual literal of the query diagram to be solved. The main differences with standard SLD-resolution are its diagrammatic nature and the fact that it is possible to keep track of the proof and present multiple solutions in a single answer diagram.

In Section 4.2.1 we formalize the inference rule and show how it works by means of an example. Although the visual inference rule is defined to work with query diagrams, we show how it is possible to extend query diagrams so that 1) we can keep track of the proof within the same query diagram and 2) we can perform multiple instantiations. Finally, in Sections 4.2.2 and 4.2.3 we introduce extended query diagrams and answer diagrams respectively.

4.2.1 Visual Inference Rule

First we define the visual inference rule:

Definition 17 (Visual Inference Rule) *Let q be a query diagram with condition visual literals $l_1, ..., l_n$ ($n \geq 1$), and d a definition diagram with conclusion visual literal g and condition visual literals $l'_1, ..., l'_m$ ($m \geq 0$).*

If l_i and g unify then we obtain a new query diagram q' containing the following literals

- $l_1, ..., l_{i-1}, l_{i+1}, ..., l_n$
- $l'_1, ..., l'_m$
- *Any new literal obtained from the unification of l_i and g.*

Let us now see how this rule work by means of an example. In Fig. 9 we develop a visual inference solving the query "$\exists x\ grandparent(charly, x)$?". In step 1 we apply the visual inference rule, unifying the grandparent literal of the query diagram with its counterpart in the definition diagram. This step captures visually the transitivity of the containment relation, and thus we obtain new information for free (i.e. what A. Shimojima calls 'free rides'), i.e. the containment of the variable into the *parent* visual predicate which conforms a new visual literal to be solved. Furthermore, in order to keep track of the proof we do not erase visual predicates and visual terms corresponding to solved visual literals. We keep them in the diagram, using dashed lines to distinguish them from unsolved visual literals. Notice that visual terms shared by two or more visual literals are only marked as solved (with dashed lines) once all visual literals were they appear are solved.

At this point we have two possibilities, depending of which visual literal of the query diagram we chose. We select the visual literal corresponding to *parents of charly* and perform the visual inference step (num. 2). However, since the conclusion visual literal of the definition diagram contains two visual terms, two different instantiations are possible. Here appears one of the advantages of our notation: we can perform a multiple instantiation —as we said before in 4.1— and represent both possible solutions in a unique diagram. Due to this multiple instantiation some visual literals may have to be duplicated in order to properly reflect the different alternatives. In Section 4.3.1 we formalize this duplication. And, furthermore, it is also necessary to annotate the diagram using an AND-OR tree to represent the logical structure of the query.

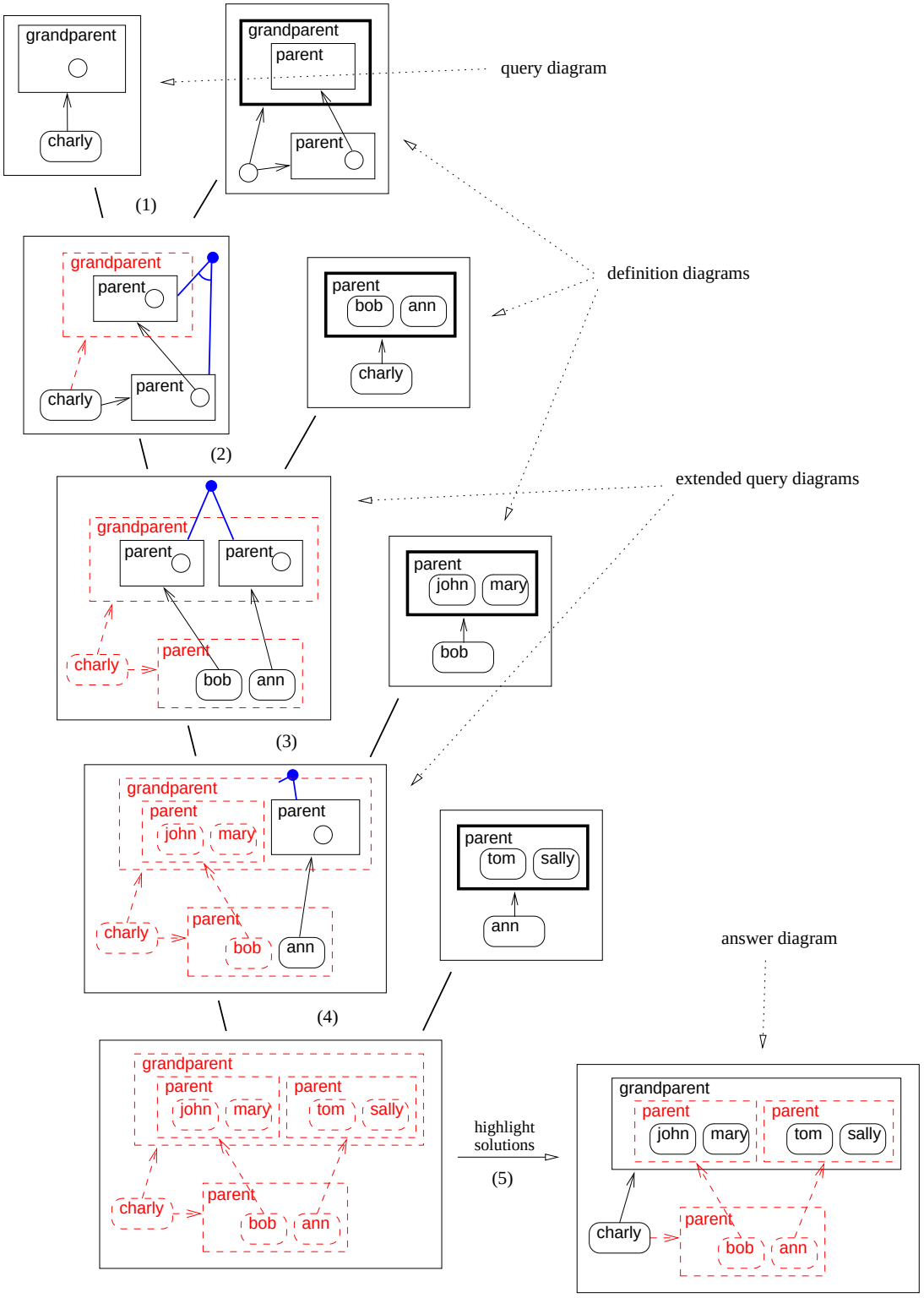


Figure 9: An example of visual inference: solving "$\exists x\ grandparent(charly, x)$?"

Finally, inference steps 3 and 4, solve the remaining unsolved literals and a solution is produced. In order to correctly visualize the solution an answer diagram is produced by highlighting those visual terms and visual literals originally present in the query.

4.2.2 Extended Query Diagrams

As we have seen the result of a visual inference step is a query diagram enhanced to keep track of already solved boxes and to represent conjunction and disjunction. We call it an *extended query diagram* and is defined as follows:

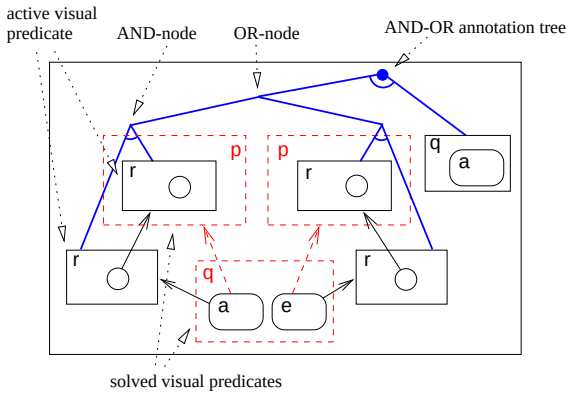


Figure 10: An example of Extended Query Diagram

Definition 18 (Ext. Query Diag.) *An extended query diagram is a query diagram plus*

- *Already solved visual literals, drawn using dashed lines to distinguish them from non-solved literals.*
- *An AND-OR annotation tree whose leaves are all non-solved visual literals. This tree represents the logical structure of the query diagram.*

Usually we will also refer to non-solved visual literals as *active visual literals*. Fig. 10 shows an example of an extended query diagram, indicating its different parts.

4.2.3 Answer Diagrams

The equivalent to the empty clause in standard textual SLD-resolution is an extended query diagram with no active literals left[1]. When such a diagram is obtained the query has been solved and the extended query diagram contains the solutions that are presented using an answer diagram.

Definition 19 (Answer Diagram) *An answer diagram is an extended query diagram with all literals solved and where visual predicates and visual terms present in the original query diagram have been highlighted by drawing them with normal lines instead of dashed lines.*

In Fig. 9, in the inference step num. 5 we obtain the corresponding answer diagram, by highlighting the *grandparent* literal, the *charly* constant, and the four instantiations of the variable originally present in the *grandparent* literal. From this point of view the inference process can be seen as a diagram completion process in which the solutions are successively stored in the extended query diagram while the goals are being solved.

[1]Since the query contains different alternatives it is possible to have a solution of the query before all literals are solved, but we will always consider the case where no more active literals are left, i.e. when all OR-branches (see Section 4.3) are explored

4.3 A Visual Inference Step

A visual inference (solving literal l_k of an extended query diagram) is performed in three steps:

1. Apply the rule as defined in Section 4.2. A multiple instantiation might occur and new literals (from the definition diagram) are added to the extended query diagram. These new literals are not attached to the current AND-OR tree.

2. Reconstruct the AND-OR tree: The new literals introduced are added to the AND-node where the solved literal l_k was attached. If l_k was directly attached to an OR-node, a new AND-node would be created if necessary.

3. If a multiple instantiation has occurred then duplicate visual literals that share variables with the solved literal. We will formally define this duplication process in next Section.

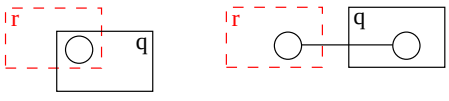
In the next Sections we address the duplication process, the AND-OR tree modifications and also other issues regarding success and failure.

4.3.1 Duplication

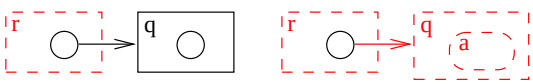
Let us now formally define which part of the diagram is going to be duplicated when a multiple instantiation occurs. We define a duplication function ($\Phi()$) which given the visual literal being solved (i.e. the one where the multiple instantiation has occurred) returns the set of visual literals that are going to be duplicated. This set contains 1) any visual literal enclosing the variable multiply instantiated and 2) any other visual literal sharing a variable with a visual literal already selected for duplication.

Definition 20 (Duplication Function) *The duplication function is defined in two steps. First we calculate the set of visual literals related to the variable multiply instantiated, and second we add recursively the other visual literals to be duplicated. Let l_m be the literal where a multiple instantiation has occurred (corresponding to predicate symbol r) and t_m the term containing the variable:*
***Step 1**:*
$\Phi(l_m) = \{l | l$ *is related to* $t_m\}$
There are three possible cases where a visual literal l (corresponding to predicate symbol q) is related to t_m:

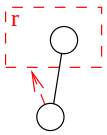
- *$l_m \neq l$, l_m is active and both literals have the same variable contained in their visual predicate*



- *$l_m \neq l$ and l_m has a variable (or a term containing a variable) as argument which is included in the visual predicate of l*



- *$l_m = l$ (therefore $r = q$) and an included variable is at the same time part of an argument*



***Step 2**:*
Repeat $\Phi(l_m) = \Phi(l_m) \cup \{l_1\}$ until no more visual literals are added, where a visual

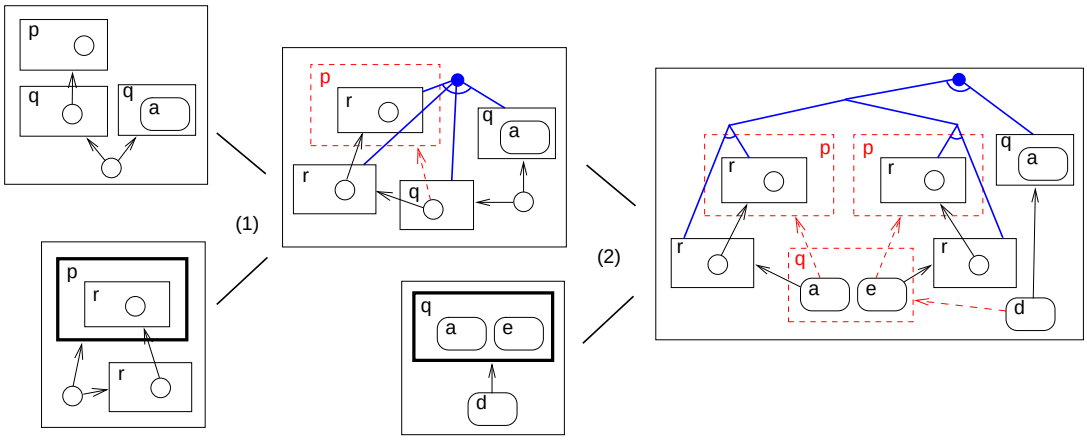
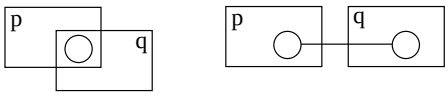


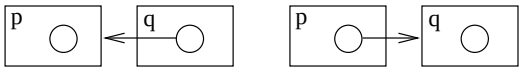
Figure 11: Example of Visual Inference (1)

literal l_1 (corresponding to predicate symbol p) is related to a visual literal $l_2 \in \Phi(l_m)$ (corresponding to predicate symbol p) and one of the following cases holds:

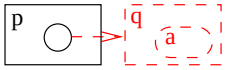
- *$l_1 \neq l_2$, l_1 and l_2 are active, and both literals have the same variable (or term with a variable) contained in their visual predicates*



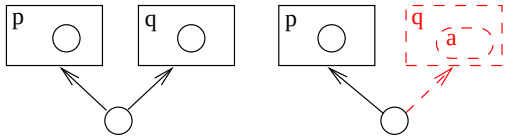
- *$l_1 \neq l_2$, l_1 and l_2 are active, and one of the literals has a variable (or term with a variable) as argument of the visual predicate which is contained in the visual predicate of the other visual literal*



- *$l_1 \neq l_2$, l_2 is active, l_1 is a solved literal and l_2 has a variable (or term with a variable) as argument of the visual predicate which is contained in the visual predicate of l*



- *$l_1 \neq l_2$, l_2 is active and both visual literals share the same variable as an argument of their visual predicates*



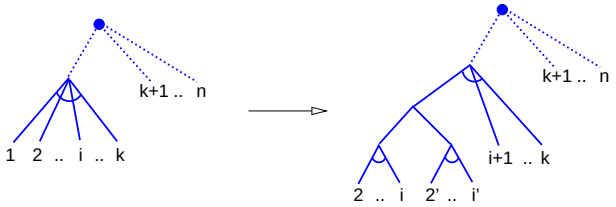
By definition of Φ, the set of visual literals to be duplicated after a multiple instantiation occurred in visual literal l_m ($\Phi(l_m)$), do not share any variable with any visual literal which does not belong to $\Phi(l_m)$. Therefore, the duplication process is performed by simply duplicating all visual literals in $\Phi(l_m)$. In order to illustrate this we use an example (see Figs. 11, 12 and 13) to help explain how these diagram transformations work. In Fig. 11 we find the initial query diagram, representing the following existential query expressed in FOPC: "$\exists x, y, z\ p(x, y) \wedge q(z, x) \wedge q(z, a)$?"

First an AND-OR tree with a single AND-node where all active literals are attached is added to the diagram showing the logical structure of the query. Then the first inference step is performed, involving the visual literal corresponding to the p predicate. Notice that new boxes introduced by the inference step are attached to the AND-node. Inference step num. 2 introduces two different alternatives into the query by performing a multiple instantiation. As a result of this multiple instantiation part of the visual literals of the diagram have to be duplicated and an OR-node is introduced in the diagram representing this two alternatives to solve the query.

We now formalize the transformation of the AND-OR tree after the duplication. First of all, we need to prove that the set of visual literals to be duplicated always belong to the same OR-branch, i.e. they are all attached to the same AND-node:

Lemma 1 *Two literals l_1 and l_2 such that $l_1, l_2 \in \Phi(l_m)$ (for some l_m) are never going to be under different branches of an OR-node.*
Proof. *The only way OR-nodes are introduced is by duplicating part of the diagram when a multiple instantiation occurs in a visual literal l_m. When we duplicate part of the diagram we duplicate all visual literals belonging to $\Phi(l_m)$, therefore it is not possible for two boxes to be related and in two different branches.*

Let us now see how the new AND-OR tree is obtained. Suppose we have a diagram with literals $l_1, ...l_n$, with a multiple instantiation occurred in l_1. Let us also suppose that $\Phi(l_1) = \{l_2, ...l_i\}$. The transformation of the AND-OR tree is the following:



Where the tree before duplication contains all literals of $\Phi(l_1)$ under the same node, and the resulting tree has been obtained by duplicating literals $l_2, ...l_i$ (obtaining $l_2, ...l_i$) and marking literal l_1 as solved The duplication process must also respect other existing containments involving already solved literals (i.e. dashed-lines visual predicates).

4.3.2 Success and Failure

An important difference between our visual inference process and a standard SLD-resolution inference process is that we may have more than one path to solve the query explicitly represented within a single query diagram. Therefore we also need to consider how the diagram is transformed when a visual literal cannot be solved.

Every time a visual literal is solved, its visual predicates (now marked as solved and drawn using dashed lines) are associated to the node of the AND-OR tree where the visual literal was attached. When a node succeeds, i.e. all its visual literals are solved, then its associated dashed visual predicates are copied to its ancestor node in the tree.

- When a branch of an *AND-node* fails, the whole *AND-node* fails, and all the active literals attached to it are also deleted. Solved dashed visual predicates associated to the AND-node are also deleted.
- When a branch of an OR-node succeeds, then the OR-node is marked as solved, in order to 'remember' that even in the case that other branches fail, the disjunction represented by the node has already been proved.
- When a branch of an OR-node fails, and the OR-node has other active literals or has already been marked as solved, then only this branch is deleted. If it is the last branch of the OR-node and the node was not marked as solved then the whole OR-node fails.
- When the root node of the tree (either an AND-node or an OR-node) fails then the whole query diagram fails.

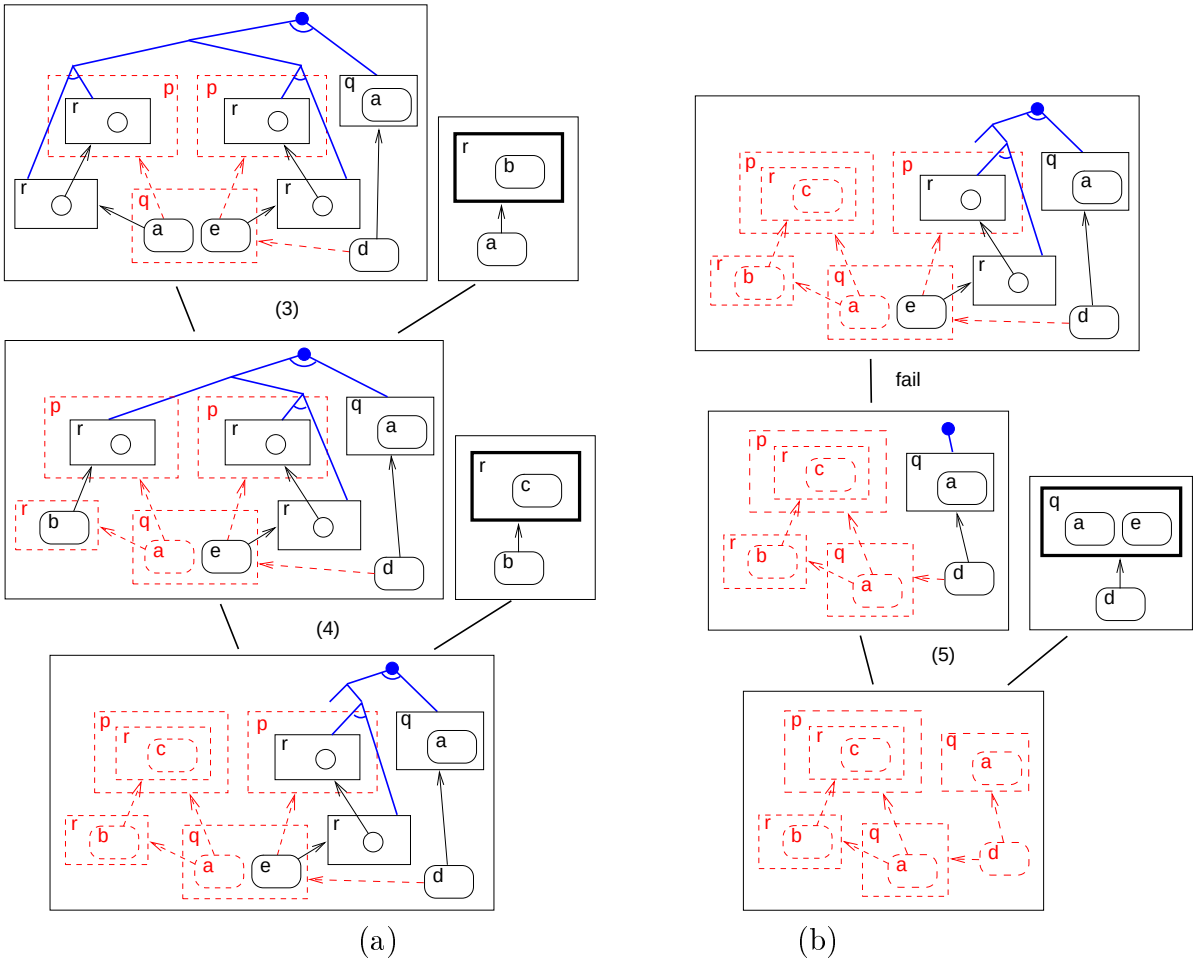


Figure 12: Example Visual Inference (2)

Let us now see the rest of the example introduced in Fig. 11. Next two inference steps (num. 3 and 4 in Fig. 12a) do not introduce further alternatives into the query. Instead, after these two steps a whole branch of the OR-node has been solved and therefore we mark this OR-node as already solved. We do so by leaving an unattached branch.

In Figure 12b the two last visual inference steps are performed. First we realize that one of the literals of the remaining branch of the OR-node cannot be solved, and therefore all active visual literals and dashed visual predicates associated to this node are deleted. Finally in step num. 5 we solve the last literal of the query diagram obtaining a solution.

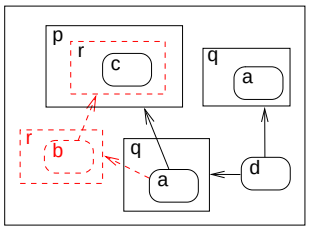


Figure 13: Example (3)

An answer diagram is obtained in Fig. 13 by highlighting the visual predicates and visual terms present in the original query, or originated by instantiation of variables present in the original query.

5 Related Work

The starting point of the work of our research group in the visual languages area was diagrammatic reasoning as presented by Jon Barwise's group in Hyperproof [4] and specially in [9]. However our goals differ from those of the diagrammatic reasoning community.

Our use of Venn/Euler diagrams is centered on its computational aspects. We want to focus on simple diagrams with a clear computational interpretation, avoiding as many logical symbols as possible. There exist other visual declarative programming languages like CUBE [12, 13], VEX [7], SPARCL [21, 22], VPP [14], VLP [11] and GrafOLog [8], but none of them uses sets, Venn/Euler diagrams and graphical containment as its foundations. The graphical schemes representing conceptual models in [6], do not attempt a formal and systematic visual representation of deductive rules. However they are an inspiration for our future work. The *existential graphs* of Charles S. Peirce (see [18, 9]), a full First-Order-Predicate-Logic diagrammatic reasoning system, are of great interest and a source of inspiration of our research; together with John Sowa's *conceptual graphs* (see [19, 20]) modeled after Peirce's diagrammatic approaches to predicate logic. Our approach differs from that of conceptual graphs mainly on the type of visual representations used.

6 Conclusions and Future Work

In this paper we have presented the current state of our research on a visual logic programming language, focusing on its visual inference system. The operational semantics presented in [17] was incomplete and did not take into account the difficulties arising when representing several alternatives to solve a query in a unique diagram, and, when keeping track of the proof that is built in the inference process. In [3] we improved the operational semantics presented in [17] to deal with these problems, and presented it in an informal way by means of an example. This paper is our first attempt to fully formalize this visual inference system.

Immediate work is going to focus on proving the completeness of the inference system presented in this paper —soundness is obvious—, and enhancing it in order to cope with other features of the visual language as it was originally introduced in [16], like for instance predicate composition. We are also interested in studying how the use of a visual syntax changes the formal properties of the operational semantics. We want to know if there are other advantages —apart from the obvious ones— of using visual syntax in declarative programming languages.

Logic-based formalisms are widely used in many areas and we believe that our approach to the visualization of a subset of First Order Logic may be successfully applied to some of them. Two applications of this visual logic programming have been explored until now. In [2] we studied the use of this visual language within formal specification and in [15] we studied its application to the databases field, specifically to the use of visual schemas in deductive databases. In the future, we plan to study other possible applications of the language.

We are currently implementing the language and expect to have a first prototype soon, to be able to perform empirical testing of our language. The implementation is done combining Java and Sicstus Prolog, and comprises an interpreter and a syntax-directed diagram editor.

Acknowledgments

Both authors have been partially supported by the MODELOGOS project TIC97-0579-C02-01 funded by the CICYT. Part of the work has been done while the first author was visiting the Visual Inference Laboratory of the Indiana University supported by a doctoral grant of the *Direcció General de Recerca* (*Generalitat de Catalunya*).

References

[1] Jaume Agustí, Jordi Puigsegur, and Dave Robertson. A Visual Syntax for Logic and Logic Programming. *Journal of Visual Languages and Computing*, 1998. To appear. Available at `http://www.iiia.csic.es/~jpf/papers.html#JVLC`.

[2] Jaume Agustí, Jordi Puigsegur, and W. Marco Schorlemmer. Towards Specifying with Inclusions. *Mathware and Soft Computing*, 4(3):281–297, 1997. Available at `http://www.iiia.csic.es/~jpf-/papers.html#mathware`.

[3] Jaume Agustí, Jordi Puigsegur, and W. Marco Schorlemmer. Query Answering by means of Diagram Transformations. In *Proc. Conference on Flexible Query Answering Systems*, 1998. Available at `http://www.iiia.csic.es/~jpf/conferences.html#FQAS`.

[4] Jon Barwise and John Etchemendy. *Hyperprooff*. CSLI Publications, Stanford, 1993.

[5] Jon Barwise and John Etchemendy. Heterogeneous Logic. In *Diagrammatic Logics: Cognitive and Computational Perspectives*. AAAI Press and MIT Press, 1995.

[6] M. Borman, J.A. Bubenko, P. Johannensson, and B. Wangler. *Conceptual Modelling*. Prentice Hall, 1997.

[7] Wayne Citrin, Richard Hall, and Benjamin Zorn. Programming with Visual Expressions. In *Proceedings of the 11th IEEE Symposium on Visual Languages*, Darmstadt, Germany, September 1995. IEEE Computer Society Press.

[8] Jean-Luc Guèrin and Paul Y. Gloess. GrafOLog: A Visual Language for a Logic with Objects. *Journal of Visual Languages and Computing*, 4:301–324, 1993.

[9] Eric Hammer. *Logic and Visual Information*. Studies in Logic, Language and Computation. CSLI and FoLLI, Stanford, CA, 1995.

[10] David Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[11] Didier Ladret and Michel Rueher. VLP: a Visual Logic Programming Language. *Journal of Visual Languages and Computing*, 2:163–188, 1991.

[12] Mark Alexander Najork. *Programming in Three Dimensions*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1994.

[13] Mark Alexander Najork. Programming in Three Dimensions. *Journal of Visual Languages and Computing*, 7:219–242, 1996.

[14] L.F. Pau and H. Olason. Visual Logic Programming. *Journal of Visual Languages and Computing*, 2:3–15, 1991.

[15] Jordi Puigsegur, Jaume Agustí, and Joan-Antoni Pastor. Towards Visual Schemas in Deductive Databases. Research report, Computer Science Department, Technical University of Catalonia, 1998.

[16] Jordi Puigsegur, Jaume Agustí, and Dave Robertson. A Visual Logic Programming Language. In *Proc. of the 12th IEEE Symposium on Visual Languages*, Boulder, Colorado, September 1996. Available at `http://www.iiia.csic.es/~jpf/conferences.html#VL96`.

[17] Jordi Puigsegur, W. Marco Schorlemmer, and Jaume Agustí. From Queries to Answers in Visual Logic Programming. In *Proc. of the 13th IEEE Symposium on Visual Languages*, Capri, Italy, September 1997. Available at `http://www.iiia.csic.es/~jpf/conferences.html#VL97`.

[18] Don D. Roberts. *The Existential Graphs of Charles S. Peirce*. Mouton and co., The Hague, 1973.

[19] John F. Sowa. *Conceptual Structures. Information Processing in Mind and Machine*. Addison Wesley, 1984.

[20] John F. Sowa. Relating Diagrams to Logic. In Guy W. Mineau, Bernard Moulin, and John F. Sowa, editors, *Conceptual Graphs for Knowledge Representation, Proc. of the First Int. Conf. on Conceptual Structures, ICCS'93, Quebec City, Canada*, Lecture Notes in Artificial Intelligence (699). Springer Verlag, Berlin, 1993.

[21] Lindsey Spratt. *Seeing the Logic of Programming with Sets*. PhD thesis, University of Kansas, 1997.

[22] Lindsey Spratt and Allen Ambler. A Visual Logic Programming languages based on Sets and Partitioning constraints. In *Proceedings of the 9th IEEE Symposium on Visual Languages*, Bergen, Norway, September 1993. IEEE Computer Society Press.