# VISUAL LANGUAGES AND LOGIC

VLL 09

Corvallis, Oregon, USA

September 20th, 2009

Editors:

PHILIP COX, ANDREW FISH AND JOHN HOWSE

# Contents

# Preface

This volume contains the proceedings of the Second International Workshop on Visual Languages and Logic (VLL 09), held in Corvallis, Oregon, USA, on the 20th September 2009, as a satellite event of the 2009 IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2009). The First International Workshop on Visual Languages and Logic was held in Coeur d'Aléne, Idaho, USA, on the 23rd September 2007.

The goal of the VLL Workshop series is to bring together researchers to explore the current state of research at the intersection of visual languages and logic, including topics such as: graphical notations for logics (either classical or non-classical, such as first or higher order logic, temporal logic, description logic, independence friendly logic, spatial logic); diagrammatic reasoning; theorem proving; formalisation (syntax, semantics, reasoning rules); expressiveness of visual logics; visual logic programming languages; visual specifcation languages; applications; and tool support for visual logics.

The six papers presented here were each reviewed by three programme committee members, and provide an insight into some of the interesting combinations of logic and visualisation currently being investigated.

As anyone who has organised such a meeting knows, success depends on many people. We wish to thank the members of the Programme Committee, who, despite being given a very short time to complete their tasks, provided prompt and helpful feedback.

Thanks are also due to the VL/HCC 2009 organisers for providing the opportunity to run VLL 09, and for their logistic support, and to the Swedish Institute of Computer Science and Nokia for their sponsorship. Finally, to put on a workshop, one must have papers and speakers. Accordingly, we would like to thank our guest speaker, Frank Ruskey, the authors of the papers included here, and the speakers who will present them.

These proceedings will be published in volume 510 in the CEUR series, published electronically and available online at: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/

Philip Cox[1], Andrew Fish[2]and John Howse[2]
20th September 2009

(1) Dalhousie University, Canada
(2) University of Brighton, UK

**Programme Committee**

- Gerry Allwein, Naval Research Laboratory, USA;
- Omid Banyasad, IBM, Canada;
- Dave Barker-Plummer, Stanford University, USA;
- Paolo Bottoni, Universita di Roma, La Sapienza, Italy;
- Frithjof Dau, University of Wollongong, Australia;
- Mateja Jamnik, University of Cambridge, UK;
- Alexander Knapp, Ludwig-Maximilians Universität, Munich, Germany;
- Bernd Meyer, Monash University, Australia;
- Nathaniel Miller, University of Northern Colorado, USA;
- Mark Minas, Universität der Bundeswehr, Munich, Germany;
- Julia Padberg, Technische Universit¨at Berlin, Germany;
- Ian Pratt-Hartman, University of Manchester, UK;
- Chris Reed, University of Dundee, UK;
- Gem Stapleton, University of Brighton, UK;
- Nik Swoboda, Universidad Politécnica de Madrid, Spain;
- Simon Thompson, University of Kent, UK.

# Venn/Euler diagrams

Frank Ruskey
Department of Computer Science
University of Victoria
Victoria, B.C. V8W 3P6
Canada
ruskey@cs.uvic.ca

**Abstract**

Among the most familiar of all visual aids is the Venn diagram. They were introduced by John Venn as an alternative to the diagrams that Euler used in his "Letters to a German Princess". Euler used his diagrams as a tool for understanding and reasoning about syllogisms.

In this talk we will review the history of Venn/Euler diagrams, discuss some of their essential mathematical properties, some of their many uses, some recent results about them, and some of the fundamental Venn/Euler problems that remain to be resolved.

The talk will be lavishly illustrated and accessible to the non-specialist. Among the specific topics to be covered are: What is NOT known about 3-Venn diagrams(!); What is a minimum area Venn diagram?; How do you draw a Venn diagram symmetrically?; and can Venn diagrams be drawn (symmetrically) on the sphere?

# More-than-coherent logic for operations on images

Paolo Bottoni[1] Anna Labella[1] Stefano Kasangian[2]
[1] Dip. di Informatica, Università di Roma “La Sapienza”
[2]Dip. di Matematica, Università di Milano

**Abstract**

A model of computation on multi-dimensional words, based on the overlapping operation, induces a categorical structure, to which a new type of logic corresponds, whose formulae express properties of computations in a language containing all first order formulae. While the resulting deductive system is strictly less powerful than (intuitionistic) first order logic, it is more powerful than coherent logic. The approach is illustrated through an example of an online game of map-colouring.

**Keywords** Categorical logics, operations on images, overlapping, online coloring.

## 1 Introduction

We investigate the properties of computations with images via the overlapping operation [1], as a particular case of interactive, non-deterministic, computations. To this end, we exploit a categorical structure, called $SymcatB$, which was studied in [11] to provide a formal setting for the study of concurrent processes [16] and bisimulations between them. In particular, $SymcatB$ is the category of symmetric $\mathcal{B}$-categories and $\mathcal{B}$-functors, where $\mathcal{B}$ is a suitable 2-category (see [20]).

In this paper, we observe that a new type of logic can be associated with $SymcatB$, in analogy with the association of (intuitionistic) first order logic with Heyting categories. Indeed, $SymcatB$ is a coherent category [10], a subcategory of which is a Heyting category. We introduce the notion of ”more-than-coherent logic” to describe the logic associated with $SymcatB$. This exploits a full first order language, for which a logic weaker than intuitionistic logic, but stronger than a coherent logic [10], is defined. In particular, negative formulae behave as the intuitionistic ones.

By associating this logic with the categorical structure, we describe and reason on computations with images, in which agents make moves by placing copies of private images, interacting to construct a concurrent state, again defined by an image. Agents can work concurrently on different parts of the (state) image, without having to act sequentially on contiguous zones, but taking turns according to paths on a tree labeled with the names of the executed actions. Overlapping contributions can be managed exploiting partial orderings imposed on the alphabets from which the pictures are formed.

We show how forms of interactive computing can be modelled in this framework (by generalizing the original algebraic structure in [11], where $\mathcal{B}$ was obtained from a meet-semilattice monoid), as contributions can overlap in any order on any area of an image, provided that some conditions are met. This kind of computation is intrinsically non deterministic, as the same apparent result can be obtained with different actions from the involved agents. Hence, future behaviors can be influenced in different ways.

After mentioning related work in Section 2, in Section 3 we introduce the main definitions of pointed image and overlapping operation and propose the main example of collaborative game and a tentative language to speak about it. Section 4 illustrates the monoidal category of trees modeling concurrent computations, in particular computations with images. Section 5 introduces the category of symmetric $\mathcal{B}$-categories in the sense of [20]. Its coherent structure is studied and the existence of a subcategory which is a Heyting category is shown. The resulting logical system is presented in Section 6, and its instantiation to the current problem is proposed in Section 7. Section 8 draws conclusions and points to future work.

## 2 Related work

We consider here two research traditions, related to the study of algebraic properties of languages, notably two-dimensional ones, and to the categorical treatment of process algebras to describe the behaviour of concurrent agents, respectively.

The algebraic characterization of 2-D languages started with the seminal works of Kirsch [12] and Dacey [4], aimed at imposing some form of juxtaposition (reduced to common concatenation in the 1D case) on 2D images. Horizontal and vertical versions of concatenation were proposed, giving rise to generative models combining horizontal and vertical rewriting [19]. However, these versions of concatenation are only partial functions, being applicable only to pictures of compatible sizes. An algebraic characterization has been given as doubly ranked monoids constructed from alphabets of bidimensional elements, in which each individual operation is associative and compatible with each other [7]. Survey of these topics, with particular emphasis on the notion of recognisability of picture languages, can be found in [6].

Attachment positions for contour curves were introduced by Shaw to allow composition of curves so that the head of an element be attached to the tail of another [18]. Pointed drawings are described in [13] by a string of directions and a pair of *departure* and *arrival* positions. Connected pointed figures equipped with concatenation give rise to a finitely generated inverse monoid, the set of generators being constituted of coloured pixels with all possible positions for the arrival and departure points. While this result concerns the descriptions of figures (shapes in black-and-white images), we are interested in images, i.e. finite bidimensional structures.

The (im)possibility of recovering concatenation of images in the 2D world by simply enriching them with attachment points, or by allowing placement only over well-behaved paths, is studied in [3]. The proposal of pointed pictures [1] avoids such limitations, as they include information on attachment points, and allows translations and rotations of the original content of the pictures and of the associated information.

Modeling the behavior of concurrent agents by trees labeled on a monoid of elementary moves is standard practice in concurrent computing [16] and has been interpreted in an enriched categorical context in [11]. There, several cases of the base monoidal structure are considered (free monoid, trace monoid, etc.). In this paper we consider for the first time a monoid which does not satisfy the left cancelation property.

Lawvere's *algebraic theories* [14] are the first categorical description of the models of a theory, while Lawvere and Tierney's elementary topos extends Grothendieck's (geometric) theory of toposes to the realm of logic, relating them to intuitionistic logic. Since then, many categorical structures have been investigated from a logical viewpoint, by associating with them a language and a deductive system. In particular, coherent categories are related to positive first order logic (coherent logic) [10]. Starting from a coherent category containing a Heyting subcategory, we consider a language more expressive and a deductive system stronger than the coherent ones.

The application of the language of more-than-coherent logic to interactive computations, seen as games, can be related to the computability logic proposed by Japaridze [9], who exploits a wider language, called universal language, including different types of existential and universal quantifiers. The universal language can indeed refer to individual moves, whereas we refer to whole computations. However, we can introduce a notion of factorization, to consider computations made of single moves.

# 3 Interactive computations on images

We consider here the problem of describing properties of interactive computations with images, based on a simplified version of the positional overlapping operation introduced in [1]. In this Section, as an instance of such a computation, we consider an interactive game based on the 4-colour problem, i.e. the problem of finding a colouring of a map using up to four colours, so that no two adjacent regions are coloured in the same way. We first introduce definitions and properties for the *overlapping operation*. In particular, overlapping is defined for (n-dimensional) images as an abstraction from many significant pixel operators as well as visual interaction phenomena, in particular in interactive construction of diagrams.

## 3.1 The overlapping operation

**Definition 1 [Meet-semilattice]**
A *meet-semilattice* $\mathbf{L} = (L, \leq, \wedge, \perp)$ is a partial order w.r.t. $\leq$, such that $\wedge$ is the greatest lower bound function and $\perp$ denotes the bottom element.

**Definition 2 [Complete meet-semilattice]**
A *complete meet-semilattice* $\mathbf{L} = (L, \leq, \wedge, \perp)$ is a meet-semilattice, such that for any non-empty family of elements its greatest lower bound exists.

**Remark 1** A complete meet-semilattice has a join for every family that has an upper bound, given by the meet of all upper bounds.

In the rest of the paper, we consider partially ordered finite alphabets, named $V$, which are complete meet-semilattices with bottom element denoted by $\tau$. We can interpret the order relation on $V$ as an information about transparency of a cell. $\tau$ denotes absolute transparency as well as undefinedness of color assignment to a cell.

**Definition 3 [Image and pointed images]**

1. An *image* $\iota$ on an alphabet $V$ is a function $\iota : Z^n \rightarrow V$, where $Z$ is the set of integer numbers and $\iota$ is almost everywhere equal to $\tau$.

2. A *pointed image* $(\iota, \overrightarrow{x_e}, \overrightarrow{x_t})$ is an image with two designated *entry* and *exit* positions.

3. $PI_V$ is the set of pointed images on $V$ where all the images constantly equal to $\tau$ are identified into $\iota_0$, forgetting about their entry and exit positions. We use $PI$ when $V$ can be left understood.

4. A *translation* of a pointed image $(\iota, \overrightarrow{x_e}, \overrightarrow{x_t})$ by $\overrightarrow{k}$, also noted $t_{\overrightarrow{k}}$, is a pointed image $(\iota', \overrightarrow{x'_e}, \overrightarrow{x'_t})$, where $\iota'(\overrightarrow{x}) = \iota(\overrightarrow{x} + \overrightarrow{k})$, $\overrightarrow{x'_e} = \overrightarrow{x_e} + \overrightarrow{k}$, $\overrightarrow{x'_t} = \overrightarrow{x_t} + \overrightarrow{k}$.

The set of pointed images $PI$ is partially ordered: $(\iota, \overrightarrow{x_e}, \overrightarrow{x_t}) \leq (\iota', \overrightarrow{x'_e}, \overrightarrow{x'_t})$ *iff* there exists a translation $(\iota'', \overrightarrow{x''_e}, \overrightarrow{x''_t})$ of $(\iota', \overrightarrow{x'_e}, \overrightarrow{x'_t})$ such that $\iota \leq \iota''$ as functions and $\overrightarrow{x}_e = \overrightarrow{x''_e}, \overrightarrow{x_t} \leq \overrightarrow{x''_t}$, i.e. $(\iota'', \overrightarrow{x''_e}, \overrightarrow{x''_t}) = t_{\overrightarrow{(x'_e - \overrightarrow{x_e})}}((\iota', \overrightarrow{x'_e}, \overrightarrow{x'_t}))$.

For the definitions above, each image induces a semilattice with the completely transparent image at the bottom and the original image at the top.

**Proposition 1** Let $\boldsymbol{\iota_0} = (\iota_0, \vec{0}, \vec{0})$, where $\iota_0$ is the image with all pixels transparent. Then, $(PI, \leq, \boldsymbol{\iota_0})$ is a meet-semilattice.

The family of overlapping operations $\bullet_{op}$ on $PI$, is parametric w.r.t. a binary associative operation $op : V \times V \rightarrow V$: $(\iota, \overrightarrow{x_e}, \overrightarrow{x_t})) \bullet_{op} (\iota', \overrightarrow{x'_e}, \overrightarrow{x'_t})) = (\iota'', \overrightarrow{x_e}, \overrightarrow{x''_t}))$ with $(\iota'', \overrightarrow{x_e}, \overrightarrow{x''_t}) = (\iota, \overrightarrow{x_e}, \overrightarrow{x_t}) \bullet_{op} (\iota''', \overrightarrow{x'''_e}, \overrightarrow{x'''_t})$, where $(\iota''', \overrightarrow{x'''_e}, \overrightarrow{x'''_t}) = t_{(\overrightarrow{x_t} - \overrightarrow{x'_e}))}((\iota', \overrightarrow{x'_e}, \overrightarrow{x'_t})))$.

Figure 1 illustrates three instantiations of the operation for two pointed images, where $e$ and $t$ indicate the entry and exit position, respectively, and $0$ the origin of the coordinates. The operation $\&_1$ preserves the value of the first argument, $\&_2$ the value of the second, and $\&_3$ combines the two values where they are both defined, and preserves the value of the defined image otherwise. Notice that in all three cases, the set of positions of the resulting image is the same, as well as the origin and the entry and exit positions.

## 3.2 A 4-colour based game

Let a collection of maps $\mathcal{K}$ be given to two agents, called the *Drawer* and the *Painter*. The Drawer starts by selecting a subset of regions from one map and challenging the Painter to colour them according to the 4-colour constraint, using overlapping: once a region has been painted, the colour of no cell in the region can be changed. After the Painter has done colouring, the Drawer selects a new set of regions and the game progresses. The Drawer wins if it can propose a region that the Painter is not able to

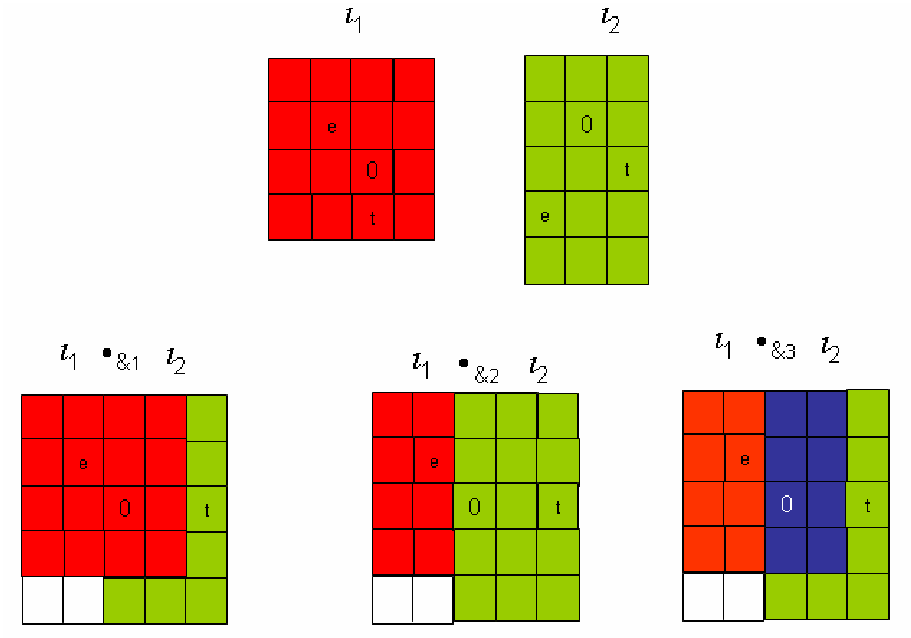


Figure 1: Instantiations of overlapping between pointed images.

colour, while the Painter wins if it achieves a complete colouring of a map. Drawn regions are assigned the colour $\$$, indicating that the region needs to receive its final coloring. We define the alphabets $\mathcal{C}_1 = \{a, b, c, d\}$, $\mathcal{C}_2 = \mathcal{C}_1 \cup \{\$\}$, $\mathcal{C} = \mathcal{C}_2 \cup \{\tau\}$, and the partial order on $\mathcal{C}$ induced by $< = \{(\tau, x) \mid x \in \mathcal{C}_2\} \cup \{(\$, x) \mid x \in \mathcal{C}_1\}$. All the regions are defined as pointed images with entry and exit position in $(1, 1)$, and with all symbols transparent except for those in one of the regions of one map. We denote the set of available regions in a map $k$ as $\mathcal{R}_k = \{1, \ldots, n_k\}$. With $\mathcal{R}$, we indicate the disjoint union of all the regions in $\mathcal{K}$.

Figure 2 shows two maps, each made of six regions, and two colourings satisfying the 4-colour condition. The three central regions are defined by the same subimages in the two maps, but receive different colours due to the 4-colour condition.

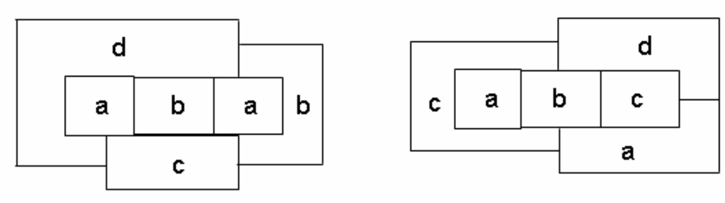


Figure 2: Two maps coloured in the correct way.

We model the colouring of a region $R$ from a map $K$ with a colour $C$ as a move

$(K, R, C) \in \mathcal{K} \times \mathcal{R}_K \times \mathcal{C}$. We call $\mathcal{M}_{[\mathcal{K},\mathcal{R},\mathcal{C}]}$ the resulting alphabet of moves. Where no ambiguity arises, we omit the subscript $\mathcal{K}, \mathcal{R}, \mathcal{C}$ and only refer to $\mathcal{M}$. We partition $\mathcal{M}$ into $\mathcal{M}^d = \mathcal{M}_{\mathcal{K},\mathcal{R},\$}$ and $\mathcal{M}^p = \mathcal{M}_{\mathcal{K},\mathcal{R},\mathcal{C}_1}$, to indicate that the Drawer can only use moves with the \$ colour and the Painter only "real" colours. Moreover, a Painter can perform a move $(k, r, c) \in \mathcal{M}^p$ for some $k$, $r$, and $c$ only if a move $(k', r, \$) \in \mathcal{M}^d$ has been previously performed by the Drawer, where either $k = k'$ or $r \in R_k \cap R_{k'}$. Note that, even if regions are taken from different maps, they are bound to agree on the already performed moves.

A colouring of a map with $n$ regions is a process with moves of the form $(y, R, x)$, and the resulting computation is a word on $\mathcal{M}^*_{[\mathcal{K},\mathcal{R},\mathcal{C}]}$. Actually, the languages of interest here are languages on the *trace monoid* $\mathcal{M}(E) = \mathcal{M}^*_{[\mathcal{K},\mathcal{R},\mathcal{C}]}/\equiv_E$ [15], where the *dependency relation* $E$ is induced by the complement of an independency relation $I$ such that $\forall k \in \mathcal{K}\ \forall r_i, r_j \in \mathcal{R}\ \forall c_h, c_l \in \mathcal{C}_1\ ((k, r_i, c_h), (k, r_j, c_l)) \in I$ and $((k, r_i, \$), (k, r_j, \$)) \in I$.

For a word $\omega \in \mathcal{M}(E)$, we call $\alpha(\omega)$ the set of moves used in $\omega$. In general, one distinguishes Drawer and Painter computations, but we omit the distinction when no ambiguity arises. For a given map $k \in \mathcal{K}$, all computations $\omega$ satisfy $noReplication(\omega) \equiv \forall r \in \mathcal{R}_k, c_h, c_l \in \mathcal{C}[\neg\exists\omega_1, \omega_2(\omega = \omega_1 \bullet \omega_2)[((k, r, c_h) \in \alpha(\omega_1)) \wedge ((k, r, c_l) \in \alpha(\omega_2))]]$, where $\bullet$ denotes concatenation of moves.

Moreover, *admissible* computations for the Painter satisfy predicate $adm_p(\omega) \equiv \forall r, r_i, r_j \in \mathcal{R}_k\ [adj_k(r_i, r_j) \implies ((k, r_i, c) \in \alpha(\omega) \implies \neg(k, r_j, c) \in \alpha(\omega))]$, where $adj_k(r_i, r_j)$ indicates that the two regions $r_i, r_j$ are adjacent in map $k$.

Each agent can perform any legal computation on a map. For the Painter, this corresponds to sequences producing 4-coloured maps, while for the Drawer these are all possible orders of presentations of regions from a map. After a Drawer's move $(k, r, \$)$, the Painter can observe only its projection $(r, \$) \in \mathcal{R} \times \mathcal{C}$ and must replicate with a move $(k', r, c)$ from a computation on a map $k'$ containing $r$. Analogously, the Drawer will observe only the projection $(r, c)$ of such a move. Once the Drawer has selected a map, it has to go on playing with its original choice. In general, $sameMap(\omega) \equiv \forall (k, r, c), (k', r', c') \in \alpha(\omega)[k = k']]$ indicates that a player always selects moves from the same map, while $play_k$ is satisfied if a player's computation satisfies $sameMap$, selecting moves from map $k$.

Of interest here are alternating games, in which the Painter has to colour regions exactly in the order in which they are proposed and progress is made only if the Painter has coloured all of them. We adopt the point of view of an external observer, which regards the game as a single computation $\omega^o$ where moves are successions of projections on $\mathcal{R} \times \mathcal{C}$ of the moves played by the Drawer and the Painter. We introduce a predicate $alt(\omega^o)$ which is satisfied if $\omega^o$ is admissible and, for any prefix $\omega_1$ of $\omega^o$, $\mid \omega_1 \mid_{\{\$\}} \geq \mid \omega_1 \mid_{\mathcal{C}_1}$. Moreover, for a word $\omega^o$ satisfying $alt(\omega^o)$, we introduce the projections $pr^d$ and $pr^p$ providing $\omega^d$ and $\omega^p$, respectively, up to the choice of the map. If the game is played with a fixed number of drawn regions at each turn, we call *alternation step* $x$ this number. Alternating games give rise to words of the form $\omega^o = a^d_{1,1} \dots a^d_{1,x} a^p_{1,1} \dots a^p_{1,x} \dots a^d_{r,1} \dots a^d_{r,x} a^p_{r,1} \dots a^p_{r,x}$. The predicate $alt_x(\omega^o)$ is satisfied if $alt(\omega^o)$ and $\omega^o$ is the unique word describing the alternating game of step $x$ for which $\omega^d = pr^d(\omega^o)$ and $\omega^p = pr^p(\omega^o)$ are the Drawer's and Painter's sequence of

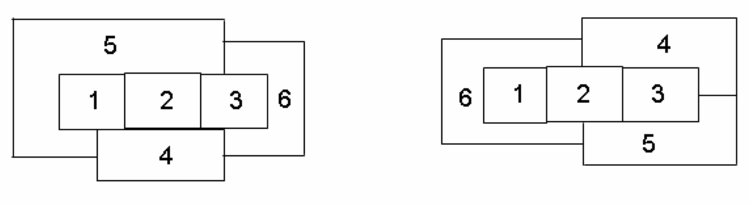


Figure 3: The Drawer's sequences for a game with two maps.

moves, respectively. This corresponds to the case of *online colouring* [8].

Two predicates can be defined on observer computations, denoting success for the Drawer or the Painter, respectively, as follows:

$succ^d(\omega^o) \equiv \exists(r, \$) \in \alpha(pr^d(\omega^o))[\neg\exists c \in \mathcal{C}_1[(r, c) \in \alpha(pr^p(\omega^o))]]$
$succ^p(\omega^o) \equiv \forall(r, \$) \in \alpha(pr^d(\omega^o))[\exists c \in \mathcal{C}_1[(r, c) \in \alpha(pr^p(\omega^o))]]$

Considering the set of maps $\mathcal{K} = \{k_1, k_2\}$ of Figure 3, one verifies Fact 1.

**Fact 1** *The following hold:*

1. $\forall\omega^d \in \mathcal{M}^d(E)\forall\omega^p \in \mathcal{M}^p(E)\forall k \in \mathcal{K}[(play_k(\omega^d) \wedge play_k(\omega^p)) \implies (\exists\omega^o[\omega^d = pr^d(\omega^o) \wedge \omega^p = pr^p(\omega^o) \wedge succ^p(\omega^o)])]$
2. $\forall h, k \in \mathcal{K}[(\neg h = k) \implies (\exists\omega^d \in \mathcal{M}^d(E), \omega^p \in \mathcal{M}^p(E)[(play_h(\omega^d) \wedge play_k(\omega^p)) \implies (\exists\omega^o[\omega^d = pr^d(\omega^o) \wedge \omega^p = pr^p(\omega^o) \wedge succ^d(\omega^o)])])]$

The Drawer's strategy is to select one map in Figure 3 and present the common regions indicated with 1, 2, 3 , in this order. After the Painter has selected the third colour, the Drawer proposes the region identified with 4 in its map. The computation will result in success for the Painter if the two were moving with respect to the same map. Otherwise, the Painter will be forced to choose a colour for which no computation can be successful. From the observer's point of view, iterations of the game in which the first three moves of each player are always the same can result into different games. Such a situation occurs for alternating games of step 1 and 3, but not for any other step.

For any game, the property $succ^p(\omega_1) \wedge perm(\omega_1, \omega_2) \implies succ^p(\omega_2)$ holds, where predicate $perm(\omega_1, \omega_2)$ is satisfied if one word can be obtained from the other by permutation of the positions of the moves.

# 4 Computing with overlapping

We introduce here the categorical structure needed to describe computations occurring on images through the use of the overlapping operation. This structure will result into an instance of a $SymcatB$-category formally defined in Section 5. The language and the logical structure associated with $SymcatB$ in Section 6 will provide the proper solution to the problem of finding a language to speak about collaborative computing with images and characterise its logics.

## 4.1 The monoidal category of trees

In order to describe the possible evolution of a concurrent process from one state to another, one exploits a set of computations labeled with elements from a complete meet-semilattice **L**, representing possible observations of the behavior of an agent (in our leading example the overlapping of visible moves performed by the two players). The elements of **L** give the *extent* of the computation. Since for the observer the process is a non-deterministic one, a further piece of information is needed to identify the degree to which two given computations are indistinguishable to observation; in general, this degree, called *agreement*, will not be maximal. Such a structure gives rise to a *generalized tree* whose paths are computations, glued together via agreement. This kind of definition can produce also pathological trees as, e.g. the empty one or a tree where two paths are completely glued together. Actually, this construction can be carried for any meet-semilattice **L** [11].

**Definition 4 [Trees]**

1. An $L$-tree (or tree) is a triple $(X, e_X, a_X)$ where $X$ is the set of *paths*, the *extent map* $e_X : X \to \mathbf{L}$ is the labeling of the paths and the *agreement* $a_X : X \times X \to \mathbf{L}$ is the gluing between paths, such that, $\forall x, y, z \in X$, it holds that:
   (a) $a_X(x, x) = e_X(x)$
   (b) $a_X(x, y) \leq e_X(x) \wedge e_X(y)$
   (c) $a_X(x, y) \wedge a_X(y, z) \leq a_X(x, z)$
   (d) $a_X(x, y) = a_X(y, x)$

2. An $L$-tree-morphism, or simulation, $f : \mathcal{X} \to \mathcal{Y}$ is a function mapping paths into paths, strictly preserving labeling and non decreasing gluing between them:
   (a) $e_X(x) = e_Y(f(x))$
   (b) $a_X(x, y) \leq a_Y(f(x), f(y))$

3. $L$-trees with their morphisms form the category $Tree_L$, or $Tree$ for short.

Figure 4 shows an example of trees, each with two paths, with equal extent, i.e. the set of words $\{ac, ab\}$, but different agreement, as the agreement between the paths in the left tree is the empty word and that for the right tree is the word labeled $a$.

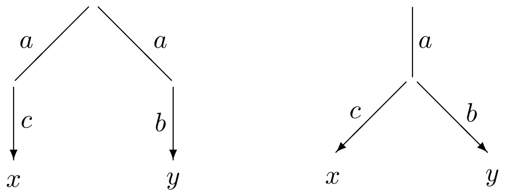


Figure 4: Two Trees.

**Example 1** Let $A^*$ be the free monoid generated by an alphabet $A$. An $A^*$-category $\mathcal{X}$ results in an $A$-labeled tree, with the set of labels ordered according to the prefix relation.

From a monoid on which the prefix relation induces a complete meet-semilattice structure (as for a free monoid), one obtains an instance of the theory developed so far. The same construction can be performed for a trace monoid.

Due to their relevance in our context, we now introduce some further operations on $Tree$, in particular cases. We now consider a meet-semilattice which has also a monoidal structure, according to Definition 5.

**Definition 5 [Meet-semilattice with monoidal structure]**
A meet-semilattice $\mathbf{L} = (L, \leq, \wedge, \bot)$ has a monoidal structure *iff* it is a monoid $(L, \bullet, 1)$ such that the following hold:

- $h \leq h'$ implies $k \bullet h \leq k \bullet h'$ *(right monotonicity)*
- $k \bullet (h \wedge h') = (k \bullet h) \wedge (k \bullet h')$ *(right semidistributivity)*
- $k \leq k \bullet h$ *(non decreasing property)*

**Proposition 2** If $\mathbf{L} = (L, \leq, \wedge, \bot)$ has a monoidal structure $(L, \bullet, 1)$, then $\bot = 1$.

Now we can lift the monoidal structure of $L$ to the level of $Tree_L$.

**Proposition 3** Let $\mathbf{L}$ be a meet-semilattice with a monoidal structure $(L, \bullet, 1)$. Then a tensor product $\otimes : Tree_L \times Tree_L \to Tree_L$ exists, producing a (non symmetric) monoidal category $(Tree_L, \otimes, \mathcal{I})$, where $\mathcal{I}$ is the one-path tree with trivial labeling 1.

**Remark 2** [11] If $\bullet$ is left-cancellative, the tensor $(Tree, \otimes, \mathcal{I})$ is left-closed, i.e. $- \otimes \mathcal{Y}$ has a right adjoint $Tree(\mathcal{Y}, -)$ for every $\mathcal{Y}$. Monoidal left-closedness would allow speaking of the "tree leading from one state to another state", so that, going back in time along a behavior, one could recover uniquely the remaining part of it from a given state. However, this property is very strong and not verified for most examples here.

**Proposition 4** A homomorphism of meet-semilattice monoids $\phi : \mathbf{L}' \to \mathbf{L}$ induces two monoidal functors $\Phi : Tree_{L'} \to Tree_L$ and $\Phi' : Tree_L \to Tree_{L'}$.

**Example 2** Given a set $A$, let $A_\dagger = (A \cup \{\dagger\})$. We now let the free monoids $A_\dagger^*$ and $A^*$ play the roles of $\mathcal{L}'$ and $\mathcal{L}$, respectively, and define a monoidal functor between them by introducing a function on words in $A_\dagger^*$, deleting instances of † as follows:

$$\textsc{del}(s) = \begin{cases} \epsilon & \text{if } s = \epsilon \\ \mu \bullet \textsc{del}(s') & \text{if } s = \mu \bullet s' \text{ and } \mu \neq \dagger \\ \textsc{del}(s') & \text{if } s = \dagger \bullet s' \end{cases}$$

DEL can be extended to a monoidal functor $\Delta$ from trees labelled with $A \cup \{\dagger\}$ to $A$-labelled trees, deleting †'s on paths. On the other hand, according to Proposition 4, given the inclusion homomorphism $i : A^* \to A_\dagger^*$ we obtain also a pair of functors, namely the obvious inclusion functor $INC : Tree_{A^*} \to Tree_{A_\dagger^*}$ and the functor $RES : Tree_{A_\dagger^*} \to Tree_{A^*}$. The latter, given a tree $\mathcal{X}$, erases from it all the paths with labels containing †. This operation corresponds to what in concurrent process algebra is called *restriction*. There is a (strict) mono $res : INC(RES(\mathcal{X})) \rightarrowtail \mathcal{X}$.

### 4.2 The case of overlapping operation

For the overlapping operations defined in Section 3 one proves the following:

**Proposition 5** Let $op : V \times V \rightarrow V$ be an associative monotonic operation with $\tau$ as unit and satisfying Definition 5; then $(PI, \leq, \bullet_{op}, \iota_{\mathbf{0}})$ with $\bullet_{op}$ defined point-wise and coordinate-wise as above, is a meet-semilattice with a monoidal structure.

One can thus define generalized trees on $\mathbf{PI}$ as illustrated in Section 4.1.

## 5 The category *SymcatB*

We expose here for the sake of completeness the general categorical-theoretic arguments which allow the introduction of the notion of more-than-coherent logic in next section. The reader more interested in applications than in general theories can skip this section or, better, substitute everywhere the locally posetal 2-category $\mathcal{B}$ with a meet-semilattice $\mathbf{L}$ and $SymcatB$ with $Tree_L$. In Section 7 we will see that for our purposes we need this particular instance only.

Given a suitable locally posetal 2-category $\mathcal{B}$ ([20]), one obtains the category $SymcatB$ where for every pair of objects $b$ and $b'$ the hom-set $hom[b, b']$ of morphisms from $b$ to $b'$ is assumed to be cocomplete and *sups* are preserved by composition. We also assume the existence of an operation on hom-sets, behaving as a *meet* w.r.t. the order. If $\mathcal{B}$ is generated as the category of relations on a regular category $\mathbf{B}$, the meet operation is given by the pullback.

**Definition 6 [Symmetric categories and functors]**

1. [20] A symmetric $\mathcal{B}$-category $\mathcal{X} =< X, e_X, a_X >$ is a set $X$ equipped with an *extent* function $e : X \rightarrow B$ and an *agreement* function $a : X \times X \rightarrow Mor(\mathcal{B})$ satisfying: $\forall x, y, z \in X$: 1) $a_X(x, x) = e_X(x)$; 2) $a_X(x, y) \leq e_X(x) \wedge e_X(y)$; 3) $a_X(x, y) \wedge a_X(y, z) \leq a_X(x, z)$; 4) $a_X(x, y) = a_X(y, x)$ where $\wedge$ denotes the meet operation.

2. [20] A $\mathcal{B}$-functor $f : \mathcal{X} \rightarrow \mathcal{Y}$ between two $\mathcal{B}$-categories is a function $f : X \rightarrow Y$, satisfying $\forall x, y \in X$: $e_X(x) = e_Y(f(x))$; $a_X(x, y) \leq a_Y(f(x), f(y))$

3. A $\mathcal{B}$-functor $f$ is called *strict* if the following equation holds: $a_X(x, y) = a_Y(f(x), f(y))$

**Definition 7 [Cartesian and coherent categories]**[10]

1. A cartesian category $C$ is *regular* if it has stable images under pullbacks.

2. A regular category $C$ is *coherent* if it has stable unions under pullbacks.

In the following, we give a series of results needed to introduce more-than-coherent logic in Section 6. As the focus of the paper is on logic rather than on categorical constructions, we omit the proofs.

**Proposition 6** If $\mathcal{B}$ is locally cocomplete, then $SymcatB$ is a coherent category.

**Corollary 1** In $SymcatB$, with every object $\mathcal{X}$, a distributive lattice $(Sub(\mathcal{X}), \cup, \cap)$ is associated, with $\cup$ and $\cap$ the *join* and *meet*, respectively. Given a morphism $f : \mathcal{X} \rightarrow \mathcal{Y}$, an image operator $\Sigma_f : Sub(\mathcal{X}) \rightarrow Sub(\mathcal{Y})$ exists, which is left adjoint to the inverse image operator $f^*$. The whole structure is stable under pullbacks.

We now consider *strict* morphisms in $SymcatB$: strictness is preserved by identity, composition and pullbacks; images are strict if the original morphisms are. We call $sSymcatB$ the subcategory of $SymcatB$ where morphisms are strict. $\mathcal{B}$ can be considered as the terminal object in $SymcatB$ as well as in $sSymcatB$.

**Proposition 7** If $\mathcal{B}$ is locally cocomplete, then $sSymcatB$ is a Heyting category [10].

In other words, every object $\mathcal{X}$ in $SymcatB$ is associated with a pair of categories $Sub(\mathcal{X})$ and $sSub(\mathcal{X})$: the first one with a coherent structure, the second one, using only strict monos, with a Heyting structure. In particular, for any morphism $f : \mathcal{X} \rightarrow \mathcal{Y}$ and any strict mono $m : \mathcal{X}' \rightarrow \mathcal{X}$, we define $\Pi_f X' \equiv \{y \in Y \mid \forall x \in X(f(x) = y \implies x \in X')\}$, which will play the role of universal quantifier (right adjoint to $f^*$). From the existence of a universal quantifier one derives the "negative" operators: $\mathcal{X}' \implies \mathcal{X}'' \equiv \Pi_m(\mathcal{X}' \cap \mathcal{X}'' \rightarrowtail \mathcal{X}')$, where $m : \mathcal{X}' \rightarrowtail \mathcal{X}$ and $\neg\mathcal{X}' \equiv \mathcal{X}' \implies \bot$.

**Proposition 8** For $\mathcal{X}$ in $SymcatB$ we have functors $i : sSub(\mathcal{X}) \rightarrow Sub(\mathcal{X})$ (*inclusion*) and $s : Sub(\mathcal{X}) \rightarrow sSub(\mathcal{X})$, left-inverse left-adjoint to $i$. $s$ has also a left adjoint $j$.

For every sub-object structure in $SymcatB$, one defines all the usual set-theoretical operators including the "negative" ones: negation, implication and universal quantifier. However, these will have the expected adjunction properties for the strict subobjects only. In fact, if we apply the definition of $\Pi_f \mathcal{X}'$ to non strict monos, the result will be a strict one, while the correspondence necessary for the adjunction will work only one-way, i.e., for $f^*(\mathcal{Y}') \rightarrow \mathcal{X}'$, there is a unique $\mathcal{Y}' \rightarrow \Pi_f \mathcal{X}'$, but the opposite is not always true. The same happens for negation and implication. As for the positive operators, they will be the same for both strict and non strict subobjects; only in the case of strict ones will they coincide with those obtained from connectives/quantifiers defined from the subobject classifier.

**Proposition 9** The following hold: 1) A mono (an epi) in $SymcatB$ is regular if and only if it is strict. 2) There is an object $\Omega$ in $SymcatB$ which classifies strict monos.

**Remark 3** In set-theoretical terms, a non-strict mono corresponds to a subobject containing "elements" of an object with both an evaluated membership and an evaluated equality, in both cases possibly non maximal w.r.t. the object. $\Omega$ will classify only strict subobjects, i.e. it will classify subobjects w.r.t. their membership. However, since it ignores non-strict monos, it will be completely indifferent to the actual equality. On the other hand, we are interested in non-strict monos in order to take into account seriously non-determinism in computations; hence we need to consider a variation of both membership and equality.

Actually, under a technical condition, which is satisfied in our case, we can prove that $SymcatB$ contains a topos, composed of skeletal Cauchy complete (see [21]) symmetric $\mathcal{B}$-categories with strict $\mathcal{B}$-functors between them.

# 6 More-than-coherent-logic

Given the categorical structure presented in Section 5 we can now introduce, in a standard way, a new logical system, intermediate between coherent and first order logics.

**Definition 8 [Logics]**

1. A *coherent* logic [10] is a sorted language with formulae built on constants $\top$, $\bot$, connectives $\wedge$ and $\vee$, the existential quantifier $\exists$, the "equality predicate" and whose deductive system comprehends rules $\alpha$), $\beta$), $\gamma$), $\delta$), $\zeta$) and $\theta$) in Table 1.

2. A *first order* logic [10] is a sorted language containing all formulae of the language of coherent logic, plus those built on connectives $\neg$ and $\implies$ and the universal quantifier $\forall$, and whose deductive system comprehends all the inference rules for a coherent logic plus the rules $\epsilon$) and $\eta$) in Table 1 (in this case, rules in $\theta$) are a consequence of the others, in particular of $\epsilon$).

3. A *more-than-coherent* logic is a sorted language containing all the first order logic formulae, and ewhose deductive system comprehends all the inference rules of first order logic, except for the second parts of the $\epsilon$) and $\eta$) rules.

We can interpret terms and formulae of the language in the usual way (see [10]), by fixing a "context", i.e. a finite set of variables containing those appearing as free ones; a term is interpreted as a morphism from the product of the interpretations of the types of the variables in the context to the interpretation of the type of the term, while a formula is interpreted as a subobject of the product of the interpretations of the types of the variables in the context. A sequent between two formulae is satisfied *iff* there is an instance of the order between the corresponding subobjects.

**Theorem 1** If $\mathcal{B}$ is locally cocomplete, $SymcatB$ is a model for a more-than-coherent logic.

**Proof:** As $SymcatB$ is coherent, we interpret its positive logical operators according to the rules for a coherent logic. As for the negative ones, we also interpret them using the $\Pi_\pi$ operator for the universal quantifier, $\Sigma_\pi$ for the existential quantifier, for some suitable projection $\pi$, and the obvious correspondences for the other operators. We are left to prove the validity of the first part of $\epsilon$) and of $\eta$). To this end, one observes that formulae involving negative operators are interpreted in strict subobjects. Hence, if $\phi$ is interpreted in $\mathcal{X}_1$ and $\psi$ in $\mathcal{X}_2$, for the first part of $\eta$) we have: $\mathcal{X}_1 \rightarrow \mathcal{X}_2$ implies $s(\mathcal{X}_1) \rightarrow s(\mathcal{X}_2)$, because $s$ is a functor. The latter implies $s(\mathcal{X}_1) \rightarrow \Pi_\pi(s(\mathcal{X}_2))$, because the first part of $\eta$) holds in a Heyting category; then composing with $\mathcal{X}_1 \rightarrow s(\mathcal{X}_1)$ and using the fact that $\Pi_\pi(s(\mathcal{X}_2)) = \Pi_\pi(\mathcal{X}_2)$, we have $\mathcal{X}_1 \rightarrow \Pi_\pi(\mathcal{X}_2)$. In the same way one can prove the first part of $\epsilon$). □

$$\alpha)\ \text{identity}\ \frac{}{\phi \vdash_{\vec{x}} \phi} \quad \text{substitution}\ \frac{\phi \vdash_{\vec{x}} \psi}{\phi[s/x] \vdash_{\vec{y}} \psi[s/x]} \quad \text{cut}\ \frac{\phi \vdash_{\vec{x}} \psi \ \psi \vdash_{\vec{x}} \chi}{\phi \vdash_{\vec{x}} \chi}$$

$$\beta)\ \text{equality}\ \frac{}{\top \vdash_{\vec{x}} x{=}x} \quad \frac{}{x{=}y \wedge \phi \vdash_{\vec{x}} \phi[y/x]}$$

$$\gamma)\ \text{conjunction}\ \frac{}{\phi \vdash_{\vec{x}} \top} \quad \frac{}{\phi \wedge \psi \vdash_{\vec{x}} \phi} \quad \frac{}{\phi \wedge \psi \vdash_{\vec{x}} \psi} \quad \frac{\phi \vdash_{\vec{x}} \psi \ \phi \vdash_{\vec{x}} \chi}{\phi \vdash_{\vec{x}} \psi \wedge \chi}$$

$$\delta)\ \text{disjunction}\ \frac{}{\bot \vdash_{\vec{x}} \phi} \quad \frac{}{\phi \vdash_{\vec{x}} \phi \vee \psi} \quad \frac{}{\psi \vdash_{\vec{x}} \phi \vee \psi} \quad \frac{\phi \vdash_{\vec{x}} \chi \ \psi \vdash_{\vec{x}} \chi}{\phi \vee \psi \vdash_{\vec{x}} \chi}$$

$$\epsilon)\ \text{implication}\ \frac{\phi \wedge \psi \vdash_{\vec{x}} \chi}{\psi \vdash_{\vec{x}} \phi \implies \chi} \quad \frac{\psi \vdash_{\vec{x}} \phi \implies \chi}{\phi \wedge \psi \vdash_{\vec{x}} \chi}$$

$$\zeta)\ \text{existential quantifier}\ \frac{\phi \vdash_{\vec{x},y} \psi}{\exists y \phi \vdash_{\vec{x}} \psi} \quad \frac{\exists y \phi \vdash_{\vec{x}} \psi}{\phi \vdash_{\vec{x},y} \psi}$$

$$\eta)\ \text{universal quantifier}\ \frac{\phi \vdash_{\vec{x},y} \psi}{\phi \vdash_{\vec{x}} \forall y \psi} \quad \frac{\phi \vdash_{\vec{x}} \forall y \psi}{\phi \vdash_{\vec{x},y} \psi}$$

$$\theta)\ \text{distributivity}\ \frac{}{\phi \wedge (\psi \vee \chi) \vdash_{\vec{x}} (\phi \wedge \psi) \vee (\phi \wedge \chi))} \quad \text{Frobenius}\ \frac{}{\phi \wedge \exists y \psi \vdash_{\vec{x}} \exists y (\phi \wedge \psi)}$$

Table 1: Logical rules

A first order language with all the usual connectives and quantifiers can thus be associated with $SymcatB$. Formulae corresponding to strict subobjects (in particular the negative ones) will enjoy all the rules of a full first order (intuitionistic) logic, while all the other formulae will enjoy all the rules of a coherent logic plus the first parts of $\epsilon$) and $\eta$).

# 7 Formulae on trees

We now give some examples illustrating the use of the language associated with $SymcatB$ with reference to the category $Tree$. In fact, we are able to define a first order language to speak about paths on these trees, i.e. computations in a concurrent framework or, as seen in Section 3, *interactive computations* on images.

A complete meet-semilattice **L** gives rise to a locally-posetal, locally-cocomplete 2-category $\mathcal{L}$, in the way described in Section 5. The category $Tree_L$, of $L$-labeled structured computations and simulations between them, coincides with the category whose objects are symmetric $\mathcal{L}$-categories and whose morphisms are the $\mathcal{L}$-functors between them. The terminal object in $Tree$ is given by **L** itself, thought of as a tree $(L, id_L, \wedge)$. Namely, it has all the elements as paths and the agreement between all of them is the meet.

Operations from Section 4.1 provide some predicates. For example, the mono $res : INC(RES(\mathcal{X})) \to \mathcal{X}$ is an interpretation of $\exists x[P_{\dagger}(x)]$, where $P_{\dagger}(x)$ means that the computation $x$ does not contain an occurrence of the elementary move †. Using negation, one can also express the occurrence of a given elementary move in every computation. Analogously, using the tensor product, one can express the factorization of a computation. In particular, if the tensor product has a right adjoint we are also able to say that "$p$ is a computation from a state $s$ to a state $s'$" using the object $[s', s]$ and the image of its unique morphism in the terminal tree. In this case the corresponding mono is not in general strict.

Strict monomorphisms in $Tree_L$ are injective simulations that strictly preserve agreement. In other words, a strict subtree $\mathcal{X}'$ of a given tree $\mathcal{X}$ contains some of its paths with the same extent and the same agreement as they have in $\mathcal{X}$. If a formula is interpreted in such a kind of subobject, it will behave as a first order formula.

As example of a non 'well behaved" formula, take $\exists x \exists x'[e(f(x)) = del(e(f(x')))]$, where the type of both $x$ and $x'$ has been interpreted in $\mathcal{X}$ and $f$ is a functional symbol interpreted in a non-strict morphism $f : \mathcal{X} \to \mathcal{Y}$[1]. In concurrent process parlance, this means that we have a simulation of the process represented by $\mathcal{X}$ via the process represented by $\mathcal{Y}$, which is more deterministic than $\mathcal{X}$, and we state the existence of two computations in $\mathcal{X}$ which are simulated by computations in $\mathcal{Y}$ with the same extent, up to some extra labels in the second one. Due to non determinism, the subobject of $\mathcal{Y}$ corresponding to this formula is not strict, i.e. we can find two computations satisfying the condition, but their agreement could be smaller than the one they have when they are simulated in $\mathcal{Y}$.

In order to show the limitations of the deductive system of more-than-coherent logic with respect to first-order logic, consider the formula $\forall y[\exists x \exists x'[e(f(x)) = del(e(f(x'))) \wedge a(f(x), y) = \perp]]$, with $x$ typed in $\mathcal{X}$ and $y$ typed in $\mathcal{Y}$, suppose it to be true for the given $\mathcal{X}$ and $\mathcal{Y}$, and that we interpret it w.r.t. the context $\mathcal{Y}$. Then there will be a mono from $\mathcal{Y}$ to its interpretation. If we now remove the universal quantification, the new formula will be still interpreted as subobject of $\mathcal{Y}$, but since it corresponds to a non-strict one there will be no mono from $\mathcal{Y}$ to such a subobject. This fact falsifies the second part of rule $\eta$).

Coming back to our main example in Section 3, we can easily prove:

**Proposition 10** *The following hold:*

1. *$PI$ is a meet-semilattice with monoidal structure on $V$.*

2. *An intrinsic first order language exists, equipped with a more-than-coherent logic, to speak about non deterministic computations with images.*

The same happens in particular for the trace monoid $\mathcal{M}(E)$. As a consequence, once one defines all the terms and predicates mentioned in Section 3.2 using the operations defined on $\mathcal{M}(E)$, all the formulae appearing there are formulae of a first order language equipped with a more-than-coherent logic. This fact shows the use of the proposed language for computing with images, in particular in cooperative or interactive processes, with reference to online coloring.

[1] We will abuse notation a bit here and identify syntactical and semantical symbols.

# 8 Conclusions

Computation on multidimensional words is becoming standard practice for multimedia applications, as well as for representing evolution of distributed states. Ad hoc methods are usually devised for different numbers of dimensions or for modelling semantics.

We have proposed a categorical setting for describing such computations based on the ubiquitous operation of overlapping. This gives an interesting enriched categorical structure, accommodating a new type of logical system, called more-than-coherent logic, with the expressive power of first-order logic, but a weaker deductive system.

This supports reasoning on composition of different contributions where each intermediate state is ”more defined” than the previous one. In particular, such a logical structure can be used to filter out moves which cannot contribute to reaching a desired final state. In [2], filters were used to explore the power of the overlapping operation, allowing the simulation of several rewriting mechanisms. Since more-than-coherent logic naturally emerges from the properties of the overlapping operation, this logic seems to provide a good setting for reasoning about different interactive phenomena involving images, and we plan to investigate its properties more deeply.

The overlapping operation can be parameterized to any number of dimensions and different types of value composition, preserving the required properties for a monoidal structure. Moreover, pointed words introduce a natural notion of synchronization where agents can cooperate in the construction of words defining the result of a computation only on designated positions. The overlapping operation exploited here is a point-wise one. An exploration of the structure underlying other types of operations may uncover different types of logic. As an example, grey-scale image morphology has been related to computations on complete lattices [17] and to fuzzy logic [5]. Moreover, it would be interesting to combine reasoning on computations with reasoning on their results, for example composing filtering on computations and on results.

# References

[1] Bottoni, P. and A. Labella, *Pointed pictures*, JVLC **18** (2007), pp. 523–536.

[2] Bottoni, P. and A. Labella, *Cooperative construction of pointed pictures*, Rom. J. of IST (to appear).

[3] Bottoni, P., G. Mauri and P. Mussio, *From strings to pictures and back*, Rom. J. of IST **6** (2003), pp. 87–104.

[4] Dacey, M. F., *The syntax of a triangle and some other figures.*, Pattern Recognition **2** (1970), pp. 11–31.

[5] Deng, T.-Q. and H. J. A. M. Heijmans, *Grey-scale morphology based on fuzzy logic*, J. of Math. Im. and Vis. **16** (2002), pp. 155–171.

[6] Giammarresi, D. and A. Restivo, *Two-dimensional languages*, , **III**, Springer, 1997 pp. 215–267.

[7] Grammatikopoulou, A., *Prefix picture sets and picture codes*, web.auth.gr/cai05/papers/21.pdf.

[8] Halldórsson, M. M. and M. Szegedy, *Lower bounds for on-line graph coloring*, TCS **130** (1994), pp. 163–174.

[9] Japaridze, G., *Introduction to computability logic*, Annals of Pure and Applied Logic **123** (2003), pp. 1 – 99.

[10] Johnstone, P., "Sketches of an elephant," Oxford Science Publications, 2002.

[11] Kasangian, S. and A. Labella, *Observational trees as models for concurrency*, MSCS **9** (1999), pp. 687–718.

[12] Kirsch, R., *Computer interpretation of english text and picture patterns*, IEEE Trans. EC **13** (1964), pp. 363–376.

[13] Latteux, M., D. Robilliard and D. Simplot, *Figures composées de pixels et monoïde inversif*, Bull. Belg. Math. Soc. **4** (1997), pp. 89–111.

[14] Lawvere, F., *Functorial semantics of algebraic theories*, Proc. Nat. Acad. Sci. U.S.A. **50** (1963), pp. 869–872.

[15] Mazurkiewicz, A. W., *Trace theory*, in: *Advances in Petri Nets*, LNCS **255**, 1986, pp. 279–324.

[16] Milner, R., "Communication and concurrency," Prentice Hall International, 1989.

[17] Ronse, C., *Why mathematical morphology needs complete lattices*, Signal Processing **21** (1990), pp. 129 – 154.

[18] Shaw, A., *The formal picture description scheme as a basis for picture processing systems*, Inf. and Cont. **14** (1969), pp. 9–52.

[19] Siromoney, R. and K. Krithivasan, *Parallel context-free grammars*, Inf. and Con. **24** (1974), pp. 155–162.

[20] Walters, R., *Sheaves and Cauchy-complete categories*, Cahiers de Topologie et Geometrie Diff. **22** (1981), pp. 283–286.

[21] Walters, R., *Sheaves on sites as Cauchy-complete categories*, J. Pure Appl. Algebra **24** (1982), pp. 95–102.

# A logical model query interface[*]

Harald Störrle
Institute of Informatics
University of Munich
Munich, Germany

**Abstract**

This paper presents the Logical Query Facility (LQF), a high level programming interface to query UML models. LQF is a Prolog library built on top of the Model Manipulation Toolkit (MoMaT, cf. [8]). It provides a set of versatile predicates that reflects the notions modelers use when reasoning about their models which makes it easy to formulate queries in a natural way. In order to demonstrate the capabilities of LQF in comparison to OCL, we have implemented it as a plug in to the popular MagicDraw UML CASE tool [3], and evaluated LQF with a benchmark suite of frequent model queries.

# 1 Introduction

## 1.1 Motivation

Over the last decade, model based and model driven development have turned into mainstream approaches in large scale industrial software engineering projects.[1] Visual languages like UML, EPCs, BPMN, DSLs, etc. play a more and more prominent role in such settings, and as a consequence, models have grown much larger (see cf. [9] and Fig. 1).

Another consequence is that more and more people are involved directly in modeling activities. Today, most modelers in large scale projects are not software engineers, but domain experts. In fact, the integration of domain experts is a crucial success factor in medium to large scale software development efforts. Thus, providing an interactive query facility for modelers is dearly needed in many if not all modeling projects.

From experience we know, however, that many modelers are challenged by the complexity of modeling languages already. Often, they can't (or won't) cope with yet another, complicated language for queries (such as OCL or QVT), let alone query APIs. But the query facilities provided by many tools (full-text search and predefined queries)

[*] Thanks to Alexander Knapp for generously sharing his OCL expertise.

[1] Since 2004, the author has participated in two such projects as lead methodologist and modeling coach.

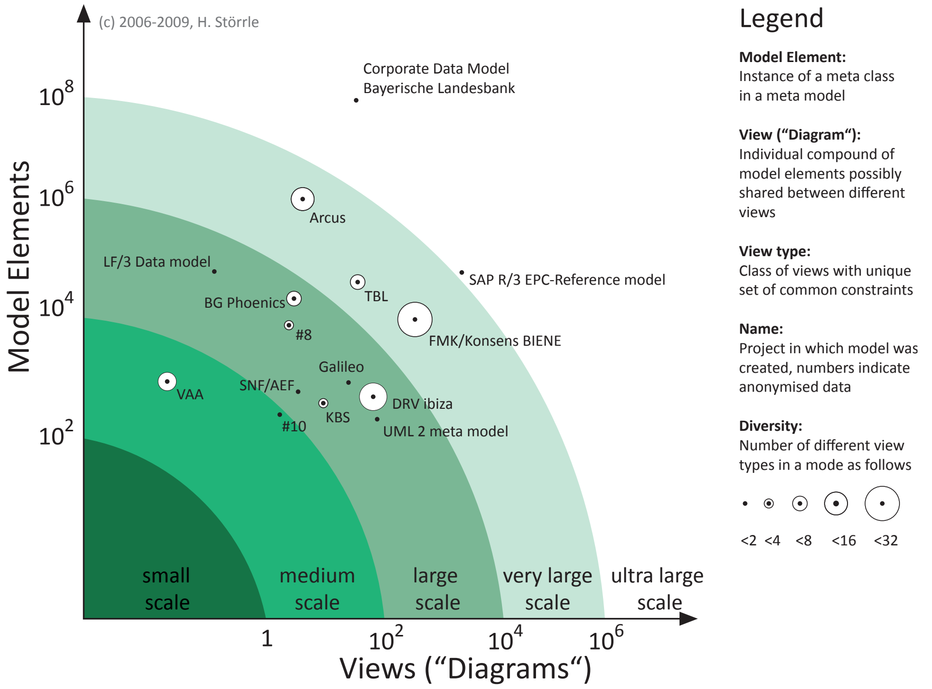


Figure 1: Real life models may become very large (cf. [9]).

are not expressive and flexible enough. This paper reports on our attempt to provide a better query facility which is expressive enough for all queries yet much easier to use.

## 1.2 Related work

Currently, there are four distinct types of UML model query facility: (1) tool specific queries, (2) application programming interfaces, (3) visual query facilities, and (4) abstract query facilities like OCL.

**tool specific facilities** Full text search and predefined queries are easy to use, but very limited in terms of expressiveness. For instance, a text search cannot find model structures or patterns, and sets of predefined queries cannot be easily extended. In our experience from industrial modeling projects, this type of query facility is too limited for many tasks.

**APIs** In contrast, an Application Programming Interface (API) offers complete control over a model and unrestricted expressiveness for querying. However, most CASE tools' APIs are very complex and are built on mainstream programming languages like Java (MagicDraw), C# (Enterprise Architect), or Visual Basic (Rational Rose). So, substantial commitment and effort is required before an end user can use such an API.

**visual queries** There are also visual query facilities like Query Models [6, 7] and VMQL [10]. Unfortunately, the Query Models approach has never been implemented; VMQL *has* been implemented, but there are no evaluations of its practical value yet.

**logic based queries** Today, the Object Constraint Language (OCL, [4]) is the de-facto standard language for complex annotations of UML models (such as consistency conditions, pre- and post-conditions). So, one could say that OCL is the "gold standard" of logic based UML query languages. However, OCL lacks several features essential for querying.

We will analyze OCL's deficiencies for querying in detail in the next section as the starting point for our own work.

## 1.3 Approach

As we have said before, OCL is the de-facto standard for expressing complex properties of UML models but it suffers from several shortcomings as a language for end user model querying. Analyzing these deficiencies will help us define a better query facility.

**Pattern Matching** OCL provides no pattern matching facilities, e. g., name matching using wild cards. For most users concerned with ad hoc queries, the full power of regular expressions are probably not required. Most of the time, it will be sufficient to allow `*` and `?` in names to match any number of characters and a single character, respectively. Defining such a function is very hard with OCL.

**Conceptual Abstraction** When expressing queries in OCL, the modeler needs to navigate through concepts defined in the UML meta model which requires substantial expertise. Also, since the concepts used in the UML meta model have little to do with the notions a modeler uses when reasoning about models, a conceptual mismatch arises that interferes with using OCL.

**Type System** OCL is strongly typed, which many people perceive as disruptive in interactive tasks (such as ad-hoc querying). Moreover, the OCL type system lacks type variables, providing only a limited form of subtyping polymorphism, but no parametric polymorphism (cf. [1]).

**Notation Size & Complexity** OCL has a rich and complex syntax with more than 50 keywords and standard library functions, plus all the usual operators and constants for arithmetics, boolean logic etc., which implies a considerable learning effort for any user.

Summing up, OCL lacks essential query facilities like pattern matching, it fails to provide a useful abstraction layer on top of the meta model, and its syntax and type system are not very helpful either. All in all, its complexity renders it effectively unusable for the average modeler. As an attempt to overcome these limitations, we have designed the Logical Query Facility (LQF) advancing our own prior work (see [8]).

We pursue three goals with LQF. Firstly, LQF should be *universal*, that is, it should allow all types of queries, including full text search. Secondly, LQF should be *expressive*, i. e., as many queries as possible should be expressible in LQF, including those predefined in typical CASE tools. Thirdly, LQF should be *simple*, that is, we aim to make LQF much simpler to use than OCL or an API. To this end, LQF provides a set of predicates that state important model properties in terms modelers are accustomed to rather than in terms of the underlying meta model (as OCL does).

# 2 The Logical Query Facility

In this section we will describe the LQF, the MoMaT framework on which it builds, and the $\text{MQ}_{\text{Logic}}$ tool implementing LQF.

## 2.1 The MoMaT framework

The Model Manipulation Toolkit (MoMaT) is a framework for processing models such as UML models using Prolog. It has been described eg. in [8], and we summarise it here only so that this paper is more self contained.

MoMaT represents model elements as individual facts and models as sets of facts, i. e., a Prolog Database. Consider the example shown in Fig. 3. It shows a simple UML class diagram (top), and its representation as a Prolog module with a set of facts, one for each model element. The blue italic numbers serve as identifiers of model elements. These identifiers are completely arbitrary; any string could be used, or, in fact, the original object identifiers provided for model elements by most contemporary modeling tools. Every fact describing one model element is described using the `me/2` predicate. Fig. 2 shows how the arguments of theis predicate are to be interpreted.

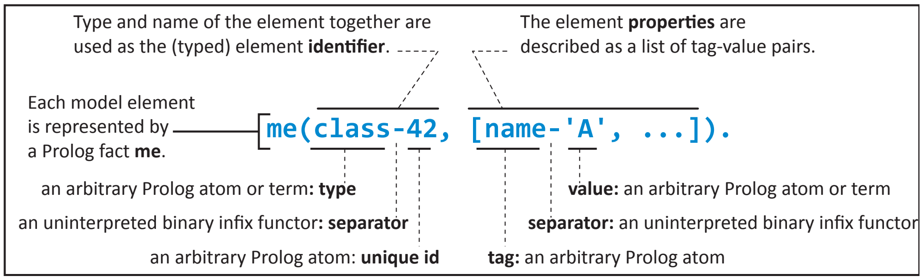

Type and name of the element together are used as the (typed) element **identifier**.
The element **properties** are described as a list of tag-value pairs.
Each model element is represented by a Prolog fact **me**.
me(class-42, [name-'A', ...]).
an arbitrary Prolog atom or term: **type**
an uninterpreted binary infix functor: **separator**
an arbitrary Prolog atom: **unique id**
**value:** an arbitrary Prolog atom or term
**separator:** an uninterpreted binary infix functor
**tag:** an arbitrary Prolog atom

Figure 2: Schematic Prolog representation of a single model element.

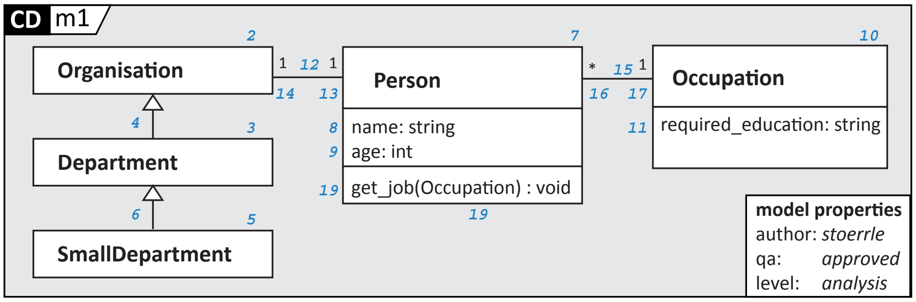
The Prolog representation of models is created automatically by the MX tool (cf. [2]). MX is a standalone tool that processes the files used to store models. Since MX is highly configurable, it can process a very wide range of file formats, that is different versions of UML, XMI, MOF/EMF/ECORE, and different tool manufacturers interpretations of them, but also BPMN/BPEL, and ADL. So far, MX has been used with MagicDraw, EnterpriseArchitect, VisualParadigm, and Adonis. Extending this range is usually a matter of hours. Thus, MX (and MoMat, and LQF) may process a wide variety of models today, and, with a little extra effort, potentially any modeling language.

The Prolog representation shown in Fig. 3 is identical for every source language or file format. The first argument contains the model element type (its meta class, in UML terminology), and and identifier. Both are arbitrary Prolog atoms. The second argument of `me/2` is an unordered list of tag-value pairs, both of which may be arbitrary Prolog expressions, including complex terms, lists, and so on. Note that this representation is purely syntactic: a new modeling language with a different set of concepts (meta classes) is treated just the same and does not require any changes to MoMaT.

This representation alone allows to manipulate models using arbitrary Prolog predicates. For instance, querying for all attributes with type `string` in model `m1` from Fig. 3, we would have to load the output of MX into a Prolog system ("consult the file" in Prolog terminology), and issue a small query at the command line prompt:

```
?- consult('m1').
?- m1:me(property-ID, Attributes),
   memberchk(type-string, Attributes).
```

The query returns all identifiers of model elements of type property (the UML jargon for attribute) in the scope of module `m1`, that have the pair `type-string` among their attributes. In this case, the answer is the set of identifiers 1 and 7. To understand this type of expression, a user needs to know a number of Prolog conventions and syntactic elements.



```
:-module(m1, [me/2,model/2]).
  model(cd-1,           [type-class_diagram, name-m1,language-'UML 2',
                         version-'2.2']).
  me(model-1,           [name-'Data',ownedMember-ids([2,3,5,7,10,12,15]),
                         level-analysis, author-stoerrle, qa-approved]).
  me(class-2,           [name-'Organisation',isAbstract-true,
                         ownedMember-ids([14])]).
  me(class-3,           [name-'Department',ownedMember-ids([4])]).
  me(generalization-4,  [to-id(2),from-id(3),isSubstitutable-true]).
  me(class-5,           [name-'SmallDepartment',ownedMember-ids([6])]).
  me(generalization-6,  [to-id(3),from-id(5),isSubstitutable-true]).
  me(class-7,           [name-'Person',ownedMember-ids([8,9,13,16,19])]).
  me(property-8,        [name-name,visibility-private,
                         multiplicity-interval(0,*),type-string]).
  me(property-9,        [name-age,visibility-private,
                         multiplicity-interval(0,*),type-int]).
  me(class-10,          [name-'Occupation',ownedMember-ids([11,17])]).
  me(property-11,       [name-required_education,visibility-private,
                         multiplicity-interval(0,*),type-string]).
  me(association-12,    [memberEnd-ids([13,14])]).
  me(property-13,       [type-id(2),visibility-private,association-id(12),
                         multiplicity-interval(1,1),type-id(2)]).
  me(property-14,       [type-id(7),visibility-private,association-id(12),
                         multiplicity-interval(1,1),type-id(7)]).
  me(association-15,    [memberEnd-ids([17,16])]).
  me(property-16,       [type-id(10),visibility-private,association-id(15),
                         multiplicity-interval(0,*)]).
  me(property-17,       [type-id(7),visibility-private,association-id(15),
                         multiplicity-interval(1,1)]).
  me(operation-18,      [name-'get_job',ownedMember-ids([19]),
                         visibility-'public']).
  me(parameter-19,      [visibility-'public',type-id(10)]).
```

Figure 3: A simple UML model (top), and its representation in Prolog (bottom). The blue italic numbers serve as identifiers of model elements; for easier reference we have added them in the UML model, close to the respective element. Many of the model element's properties are default values (e. g., visibilities, multiplicities and `isSubstitutable`). The layout of the Prolog representation has been improved for readability. The notation `:-module` is the syntax SWI Prolog uses to define a module.

**Modules** A module in the Prolog system we use is a flat name space. Elements in this name space may be accessed by prefixing a predicate by the module name and a colon.

**Facts** A Prolog fact is an identifier followed by a bracketed sequence of arguments which are separated by commas. A fact is terminated by a full stop. The predicate `-` is defined as an infix operator, so `type-string` really is identical to `-(type, string)`.

**Variables** In Prolog, all identifiers starting with a capital letter are logical variables. The underscore character denotes the anonymous variable.

**Queries** Stating a fact prompts Prolog to try and find a variable binging that makes this fact true relative to the currently known facts.

**Lists** Lists are enclosed in square braces, the list elements are separated by commas.

While this type of access brings the full power of Prolog to UML models, it requires considerable knowledge both of Prolog and the respective modeling language. MoMaT provides an abstraction layer that makes it easier to deal with complex operations on models of different kinds. However, since MoMaT provides the full spectrum of operations, it has proved to be too complex for just querying, and definitely too difficult to learn for the casual user. LQF, on the other hand, provides a restricted and specialised set of operators that makes this possible.

## 2.2 The LQF predicates

The Logical Query Facility (LQF) provides a small set of powerful and generic predicates on top of MoMaT. The LQF predicates capture the properties and relationships of model elements in the terms modelers are accustomed to rather than in terms of the underlying meta model (as OCL does). See Table 1 for a complete reference of the LQF-predicates currently defined. Note that most arguments may be either unbound, bound to items, or bound to sets of items. Predicates from `associated` on also have an additional optional last parameter indicating the number of steps (default is 1).

As a first example, consider again the query we defined in the previous section to determine the string-typed attributes in model m1. Using LQF, this query may be rewritten as

```
exists(property, ID, [type-string])
```

Now consider a more complex example. Assume, we want to check that two model elements $\mathtt{E}_1$ and $\mathtt{E}_2$ are associated. Using OCL this requires us to navigate from $\mathtt{E}_1$ and $\mathtt{E}_2$ to their respective `ownedMembers`, and find an association containing them both. In order to *find* the opposite end of an association partner, a different OCL statement is needed, and in order to get pairs of associated model elements, *yet another* OCL statement is needed.

In contrast, the LQF predicate `associated/2` may be instantiated in all three ways, i. e., with both $\mathtt{E}_1$ and $\mathtt{E}_2$ bound ("check association between them"), with just one of them bound ("get the other end of an association"), or none of them bound ("find associated pairs of elements"). Additionally, the LQF predicate provides an option to check whether the association is indirect, that is, via a given number of steps (including "any"). Also, it is defined on pairs of elements as well as on sets of elements (for n-ary associations). Finally, it may be used for all kinds of model elements, whereas OCL would require one definition for every pair of element types.

Similar options and usage modes are provided by all other LQF predicates. The predicates concerned with relationships also have an additional optional last parameter indicating the length of the path of the relationship type ("steps"), ranging from 1 (default) to * (any number of steps). For instance, `is_a(A,B)` asserts that there is a generalization relationship between model elements `A` and `B`, while `is_a(A, B, 3)` asserts that there is a chain of at most three generalizations between `A` and `B`. Similarly, `is_a(A, B, *)` asserts that there is a chain of generalizations between `A` and `B`, and it may be of arbitrary length.

## 2.3 The MQ$_{\text{Logic}}$ Tool

In order to explore our approach further, we have implemented MQ$_{\text{Logic}}$, a prototype plug in to the popular MagicDraw UML CASE tool (cf. [3]). It uses the MX model converter [2], some of the infrastructure of the MQ model query tool [11], SWI-Prolog and the JPL Java-Prolog-Bridge library (see `www.swi-prolog.org`). The LQF predicates are implemented as a set of SWI-Prolog modules. Fig. 4 shows an overview of the structure of MQ$_{\text{Logic}}$. See Fig. 5 for a screenshot of MQ$_{\text{Logic}}$.

This chart is annotated with the steps involved in creating and executing a query. We will start with the steps marked by white circles.

① First, the user creates or obtains a model and starts the MQ$_{\text{Logic}}$ system from within MagicDraw.

② The model is exported by MagicDraw and stored as a XMI-file in the local file system.

③ Using the MX tool (cf. [2]), the XMI file is converted into a set of prolog facts.

Steps ② and ③ are performed completely automatically. Note that ③ modifies only the format, but leaves the semantic contents of the model completely unchanged. After changes to the model the user must refresh its Prolog representation which repeats steps ② and ③.

In order to execute a LQF query, the following steps must be executed (marked with numbered black circles in Fig. 4).

❶ The user inputs an ordinary Prolog query as plain text to the MQ$_{\text{Logic}}$ input window, using the predicates defined by LQF (see Table 1).

❷ The query is sent as-is to the Prolog engine via the JPL Java-to-Prolog bridge.

❸ The query is executed as-is, dynamically using LQF predicates.

Table 1: The predicates defined by LQF.

exists(TYPE, ID, PROPS)

There is an element of type TYPE identified by ID with the properties listed in PROPS as Key-Value pairs. Note that at least one of the Key or the Value must be instantiated.

sub_type_of(SUPERTYPE, SUBTYPE)

In the underlying modeling language, SUBTYPE is more special than SUPERTYPE.

attribute_of(TYPE, ID, Value)

In the underlying modeling language, SUBTYPE is more special than SUPERTYPE.

name(ID, NAME)

The element identified by ID has the qualified name NAME.

match(VAL, PATTERN)

Value VAL matches the pattern PATTERN (both parameters must be instantiated).

distinct(IDS) All elements in IDS are distinct.

occurs_in(ID, D)

The element identified by ID occurs in the diagram identified by D.

associated(ID-SET)

All elements in ID-SET are part of an nary association, where $n \geq |\text{ID-SET}|$.

rel(ID, ID', RTYPE)

There is a relationship of type RTYPE between the element(s) identified by ID1 and the element(s) identified by ID2. If both ID1 and ID2 are sets, all pairs of identifiers must be in the relationship.

is_a(ID, ID')

There is a generalization relationship between the element(s) identified by ID and the element(s) identified by ID'. If both ID and ID' are sets, all pairs of identifiers must be in the relationship.

depends(ID, ID')

There is a dependency relationship between the element(s) identified by ID and the element(s) identified by ID'. If both ID and ID' are sets, all pairs of identifiers must be in the relationship.

connected(ID, ID')

There is any kind of connection between the element(s) identified by ID and the element(s) identified by ID'. If both ID and ID' are sets, all pairs of identifiers must be connected.

precedes(ID, ID')

There is a sequential ordering relationship between the element(s) identified by ID and the element(s) identified by ID' (e. g., before/after, incoming/outgoing etc.). If both ID and ID' are sets, all pairs of identifiers must be in the relationship.

calls(ID, ID')

There is a calling relationship between the element(s) identified by ID and the element(s) identified by ID'. If both ID and ID' are sets, all pairs of identifiers must be in the relationship.

contains(ID, ID')

There is a whole-part relationship between the element(s) identified by ID and the element(s) identified by ID' (e. g., class/attributes, package/members, state/substate etc.). If both ID and ID' are sets, all pairs of identifiers must be in the relationship.

❹ The results are presented back to the user. Currently, this feedback is restricted to simple values such as (qualified) names of model elements.

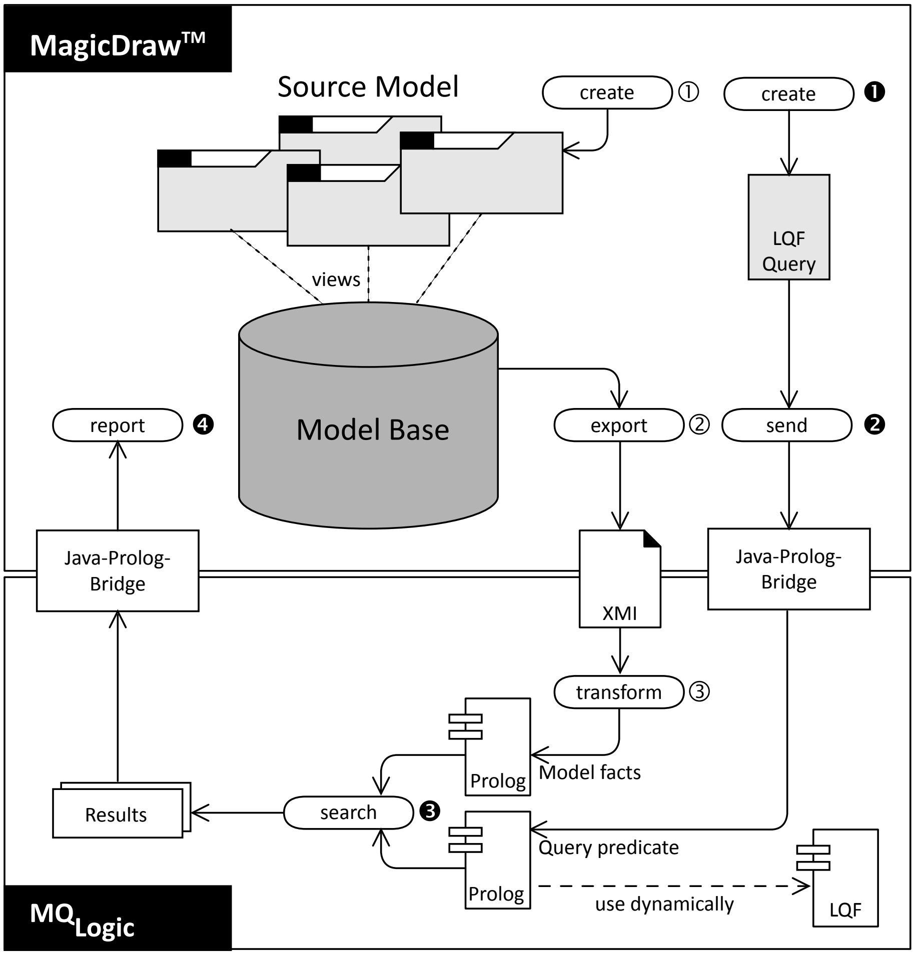
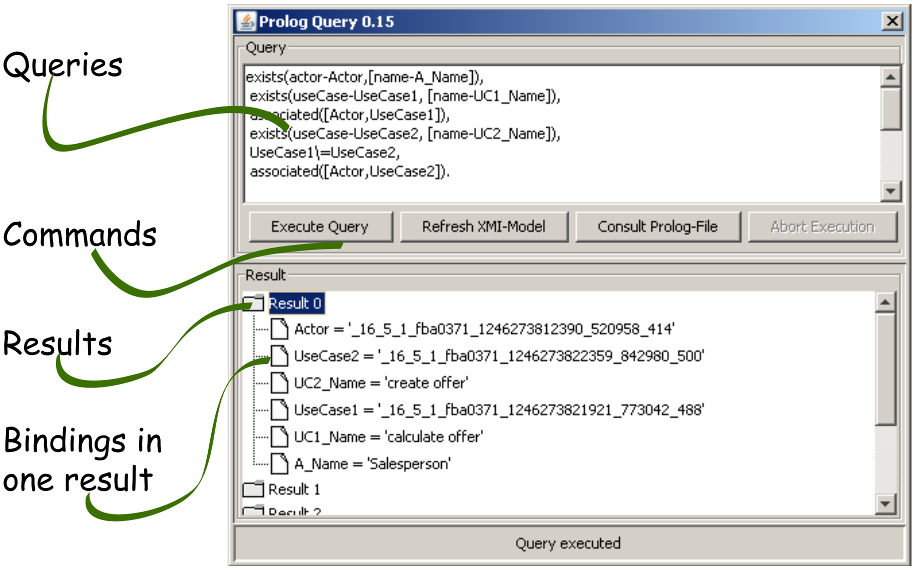


Figure 4: Overview of our prototype implementation of $\text{MQ}_{\text{Logic}}$.

# 3 Evaluation

Since $\text{MQ}_{\text{Logic}}$ allows us to run arbitrary Prolog queries against the model, we may issue every computable query. So, in terms of expressiveness, LQF is equally powerful as any API offered by any UML tool (assuming unrestricted read access to the complete model by the respective API). Similarly, any computable function may theoretically be expressed in OCL, so there is not difference in terms of expressiveness between these



```
exists(actor-Actor,[name-A_Name]),
 exists(useCase-UseCase1, [name-UC1_Name]),
 associated([Actor,UseCase1]),
 exists(useCase-UseCase2, [name-UC2_Name]),
 UseCase1\=UseCase2,
 associated([Actor,UseCase2]).
```

```
Result 0
  Actor = '_16_5_1_fba0371_1246273812390_520958_414'
  UseCase2 = '_16_5_1_fba0371_1246273822359_842980_500'
  UC2_Name = 'create offer'
  UseCase1 = '_16_5_1_fba0371_1246273821921_773042_488'
  UC1_Name = 'calculate offer'
  A_Name = 'Salesperson'
Result 1
Result 2
```

Figure 5: Screenshot of the MQ$_{\text{Logic}}$ prototype running in MagicDraw.

three alternatives. Now, the crucial question is, whether creating and/or understanding queries in LQF is easier than in OCL. In order to find out, we tested the MQ$_{\text{Logic}}$ tool. Over the last years we have collected a suite of the ten most popular queries (beyond full text search) people have wanted to run against their models (see Fig. 6). We will use them as a benchmark to evaluate predefined APIs, LQF and OCL, contrasting how they represent these queries. Due lack of space, we will discuss only the first six queries in this paper.

- text search with pattern matching
- search for particular attribute values
- unconnected nodes/subgraphs
- all transitive super classes
- counting elements of given types
- undefined attributes
- elements of a given type
- structural patterns
- invisible model elements
- references to an element

Figure 6: Some of the most frequent types of queries in industrial modeling projects.

## 3.1 Text search with pattern matching

Probably *the* most frequent query is to do a full text search for a given string over a complete model. Most (though not all) CASE tools offer this functionality. A sample application might be “Find all occurrences of ’foobar’ in any attribute of any model element.” In LQF, this is a rather simple expression.

```
exists(_, Element, [Attr-Val]),
match(Val, '*foobar*').
```

Recall that all variables are written with leading capitals except the underscore which is the anonymous variable. Analogously, we could ask for an element whose name is restricted by a wild card pattern. For instance, when looking for all occurrences of the factory-pattern, we might ask for “All classes whose name ends with ’Factory’”. In LQF, this may be expressed as follows.

```
exists(class, C, [name-N]),
match(N, '*Factory').
```

To our knowledge, such queries can’t be expressed in OCL.

## 3.2 Search for undefined attributes or particular values

One of the most common queries is to ask for “unfinished work”, for instance, any attributes that should be filled but are not. For instance, operations of classes may or may not have a visibility. So, when looking for operations that lack a value for “visibility”, in LQF we would have to say

```
exists(operation, Element, Attributes),
not(member(visibility, Attributes)).
```

while we could not express this in OCL.

Since most tools do not distinguish between attributes that are left empty on purpose and attributes that have not yet been filled, it is common to set attributes of the latter kind with a dummy default value like ’??’ or ’ToDo’, indicating unfinished business (if an automatic default is not available, it may be replaced by manual work). Then, a full text search could find such markers. However, it must also be possible to restrict the search scope and the the text search must be guaranteed to access all fields. Unfortunately, these preconditions are rarely met (we know of no such example). Thus, querying for such values across all types of attributes is a convenient way of checking for unfinished business. To our knowledge, this can’t be expressed in OCL. In LQF, the query would read

```
exists(_, Element, [Attribute-'??']);
exists(_, Element, [Attribute-'ToDo']).
```

### 3.3 All elements of a given type

The first query from our benchmark that may be expressed in OCL is a query for all elements of a given type, say, classes, in a given model. Using LQF, this query could be expressed as `exists(class, C, [])`. In OCL, we would have to use the `allInstances` construct, as in `Class.allInstances()`.

```
context Package
  self.packagedElement->select ( t | t.oclIsTypeOf(Class)).
```

Both queries are of approximately similar complexity, but it is already clear that the second query requires knowledge of the UML meta model (i. e., the meta association `packagedElement`), but also that the OCL syntax is rather complex (i. e., the difference between the `.` and the `->` operator, and the keywords `self`, `select` and `oclIsTypeOf`). This type of query is also easily expressed in many tools' query facilities using predefined queries.

### 3.4 All transitive super classes

Collecting all (transitive) super classes of a class named "Contract" amounts to computing a fixed point, which is a rather challenging task for the ordinary modeler (and for quite a computer science graduate, too). Expressing this in OCL adds an additional level of syntactic complexity, as the following code demonstrates.

```
def:
superClasses_1_1(baseClass: Class) : Set(Class) =
   if self.hasGeneralization()
   then self.generalization.general.
          oclAsType(Class)->asSet())
   else Set{}
   endif

def:
superClasses_n_1(baseClasses: Set(Class)) : Set(Class) =
   baseClasses->forAll( bc | superClasses_1_1(bc) )
     ->flatten()->asSet()

def:
superClasses_n_n(baseClasses: Set(Class)) : Set(Class) =
   let next = superClasses_n_1(baseClasses)
   in if next.equals(baseClasses)
      then return baseClasses
      else return superClasses_n_n(next)
      endif
```

We first define `superClasses_1_1` to compute the set of direct super classes of a single class, the simplest case. In the next step, we lift this function to sets of base classes, defining superClasses_n_1. The flatten operator transforms sets of sets of items into sets of items. Finally, chains of inheritance relationships are computed by superClasses_n_n, which also includes an implicit occurs check. Our query for the super classes of Contract can thus be expressed as follows.

```
let baseClass = self.packagedElement
   ->select( x | x.oclIsTypeOf(Class)
     ->select( x | x.name = 'Contract')
       ->asOrderedSet->at(1)
in superClasses_1_1(baseClass)
```

With LQF, all this complexity is encapsulated in the is_a predicate so the respective query is rather simple.

```
  exists(class, Sub,   [name-'Contract']),
  exists(class, Super, []),
  is_a(Sub, Super, steps=*).
```

There are three reasons for this succinctness. First, the notion of "is a superclass of" used to characterize the query in natural language is present in LQF, but not in OCL. Creating such an abstraction in OCL requires considerable work and expertise. Second, the OCL syntax is rather complex, thus difficult to master. Third, OCL's type system intervenes, forcing us to include type casting operations like asOrderedSet().

Note also, that in the case of OCL, we would have to define similar functions for *every single* type of relationship that may occur transitively. In LQF, on the other hand, the rel predicate covers all type of relationships. Additionally, the most frequent cases (generalization, calling, precedence etc.) are also provided with convenience predicates.

So, while we could hide the complexity of the fixed point computations in OCL behind suitable library functions created by experts, there would have to be a large set of similar functions for different types and usage modes. Six years after the last OCL version was finalized, no such library seems to exist. And even if it did exist, the user still would have to learn a large set of functions with complex syntax.

### 3.5 Structural patterns

Consider next the query for a particular structure, e. g.: "Collect all actors associated to at least two different Use Cases." This query represents a large class of queries for local model structures and are useful for design pattern mining. In OCL, this query may be expressed as follows.

```
context Package
def:
actorUseCaseAssoc(a: Actor, u: UseCase) : Bool =
  let types : set(Element) =
    self.packagedElement->asSet()->
      select(assoc | assoc.isKindOf(Association)).
```

```
          ownedElement.type->asSet().
  in let participants : set(Element) = {a, u}.
    in types.intersection(participants) = participants

def:
actorWithTwoUCs(a: Actor) : Bool =
  self.packagedElement->asSet()-> select(ucs |
     ucs.isKindOf(UseCase))
      ->collect( uc | actorUseCaseAssoc(a, uc))
        ->count() > 1

def:
allActorsWithTwoUCs() : Set(Actor)=
  self.packagedElement->asSet()->
    select(a | a.isKindOf(Actor))
      ->collect( a | actorWithTwoUCs(a))-> asSet()
endpackage
```

In LQF, this query would read as follows (this is also the query we show in Fig. 5).

```
  exists(actor, Actor,[]),
  exists(useCase, UC_1,[]),
  exists(useCase, UC_2,[]),
  distinct([U1, U2]),
  associated([Actor,UC_1]),
  associated([Actor,UC_2]).
```

### 3.6 OCL-APIs

While the OCL as such does not offer much to support querying. In that respect, it is fairly well comparable to MoMaT without LQF as an additional abstraction layer on top of it. It seems that no such query API exists for OCL. In fact, it seems that there are few OCL APIs for whatever purpose publicly available.

One notable exception is the UML, however, which defines 77 auxiliary functions and helpful abbreviations for defining OCL queries. These include a number that may improve writing queries in OCL, for instance

- `allParents()` returning the transitive closure of the Generalization relationship;
- `general` abbreviates `generalization.general`;
- `<EXPR>[<TYPE>]` abbreviates `<EXPR>.oclAsType(<TYPE>)` where `<EXPR>` is any OCL expression and `<TYPE>` is any meta class (type cast in QVT);
- `opposite` abbreviates access to the opposite end of a (binary) association.

This collection of OCL predicates and shorthands is not really an API, it has not been designed to facilitate end user queries. It is just the collection that happened to be helpful when defining the constraints of the UML standard document. So, it is not complete or orthogonal. For instance, there is no predicate for the transitive closure of the *aggregation* relationship, `allParents` lacks an occurs check, there is no predicate to collect all inherited features, and so on. Also, many of the features of LQF like pattern matching, and predicate overloading are not defined. Still, using these auxiliary predicates makes OCL much better usable than pure OCL, as our experiments have shown (see next Section).

# 4 Experimental evaluation of LQF

While we believe our approach is obviously better than OCL, we are biased of course, compromising our judgment. Our claim of superiority is mostly concerned with the usability, most notably the understandability of LQF as a model query language. Obviously, such a claim can only be examined empirically. We have therefore devised a questionnaire with a set of tasks to help answer these questions. A complete account of these experiments, unfortunately, would be beyond the scope of this paper and will be submitted elsewhere. Without going into the details, we only summarize our findings here.

The experiment consisted in a questionnaire where subjects were asked to match queries described in natural language and queries described in OCL and LQF, the latter being our two experimental conditions. In a second task, subjects were asked to judge as correct or not pairs of given matches of a natural language query and a query expressed in OCL or LQF. Next, subjects were asked to compare the time and effort it took them to complete OCL and LQF tasks, and their personal opinion of the understandability of the respective languages. Finally, some of the subjects participated in structured interviews to further elaborate on their experiences and feelings concerning the tasks.

Unsurprisingly, we could demonstrate that subjects made many more mistakes using OCL than they did using LQF, for all tasks, and for all categories of errors. Subjects also consistently judged their effort with OCL tasks much higher than LQF tasks and generally found LQF much better understandable than OCL (which was generally judged as very difficult to understand). These findings were also confirmed by post-experiment interviews. Interestingly, the occupation of the subjects (students, IT professionals, scientists) and their prior knowledge of OCL did not influence these results substantially.

As we have said, none of these findings were surprising, quite the opposite. An interesting phenomenon occurred, however, when adding another experimental condition besides OCL and LQF, namely, OCL plus the convenience functions defined en passant in the UML standard (see [5]). We called this query language “OCL+UML”.

The error rates of OCL+UML were slightly lower than those of LQF, and similarly, the subjective judgments were slightly better. However, when controlling for prior OCL knowledge, the relation between LQF and OCL+UML flipped, both in error rates and judgments. That is: subjects with no prior OCL exposure performed better on LQF than on OCL+UML, and subjects with OCL exposure performed better on OCL+UML than on LQF. In most cases, the exposure was a rather substantial MDA course the students acting as subjects had just finished.

# 5 Discussion

## 5.1 Summary

This paper presents the Logical Query Facility (LQF), a very high-level Prolog API suitable for querying UML models ad-hoc by end-users. We have implemented the $\mathsf{MQ_L}$ tool, a plug in to the popular MagicDraw CASE tool implementing LQF. It allows to access all languages supported by MagicDraw, i.e., all of UML, a variety of UML profiles, and BPMN. Executing a query in $\mathsf{MQ_L}$ amounts to translating a UML model into a Prolog rule base, and executing the LQF-based query predicate on it. LQF builds on the MoMaT system (see [8]). It shares some of the infrastructure of VMQL [10], but follows a distinct approach defining its own language, and providing its own tool.

## 5.2 Contribution

Our approach attempts to achieve *universality*, *expressiveness*, and *simplicity* (cf. Section 1.2). We have evaluated the universality and expressiveness of our approach against these goals by collecting a test suite of common queries and checking that all of these queries can be expressed in LQF. We have evaluated the simplicity of our approach by contrasting the OCL and LQF representations of these queries. It is obvious that LQF expressions are much simpler and shorter than corresponding OCL expressions. We have tried to confirm this finding by a controlled experiment. Although our results seem to confirm our hypothesis, we do not have sufficiently many data points yet to truly support our claim. Further experimentation is clearly called for.

LQF offers two advantages over OCL, today's de-facto standard for querying UML models. First, it shields the modeler from the complexity of the UML meta model so that a modeler may express queries using familiar concepts. Second, it provides a very small, yet powerful interface as all predicates may be used in different usage modes (i. e., different patterns of instantiating parameters). As our experiments have demonstrated, this interface is truly easy to understand.

While we cannot be sure that our sample of queries is truly representative for all application contexts, it surely is sufficient to contrast the different approaches. Obviously, all text based query facilities for visual query languages suffer from the media gap between query and model. To which degree this impedes querying is currently an open question.

## 5.3 Future work

There are a number of promising routes for future work. First of all, LQF lacks means to access the diagrammatic aspect of models, i. e., visual features of diagrams such as relative position, size, and so on. Also, accessing the meta model in the same way as the model would allow parameterization over concepts.

Then, $\mathsf{MQ}_{\mathsf{Logic}}$ is just a prototype. It currently lacks features for visualization of query results, debugging support, and productivity features like syntax highlighting, auto completion and so on.

Finally, the syntax seems to be suboptimal. Whether the improvements come from visual notations like VMQL (cf. [10]) or controlled natural language constructs can only be determined empirically.

# References

[1] Luca Cardelli and Peter Wegner. On Understanding Types Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17(4), December 1985.

[2] Josef Edenhauser. MX – Model Exchange Tool. Master's thesis, Innsbruck University, 2008.

[3] No Magic, Inc. *USERS MANUAL (version 16.5)*, 2009. available online at `http://www.magicdraw.com`.

[4] OMG. UML 2.0 OCL Specification (ptc/03-10-14). Technical report, Object Management Group, October 2003. available at `www.omg.org/docs/ptc/03-10-14.pdf`.

[5] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, V2.2 beta (ptc/08-05-04). Technical report, Object Management Group, May 2008. Available at `www.omg.org`, downloaded on March 6$^{\text{th}}$, 2009.

[6] Dominik Stein, Stefan Hanenberg, and Rainer Unland. Query Models. In Thomas Baar, Alfred Strohmeier, Ana Moreira, and Stephen J. Mellor, editors, *Proc.* 7$^{\text{th}}$ *Intl. Conf. Unified Modeling Language (≪UML≫'04)*, number 3273 in LNCS, pages 98–112. Springer Verlag, 2004.

[7] Dominik Stein, Stefan Hanenberg, and Rainer Unland. On Relationships between Query Models. In A. Hartman and D. Kreische, editors, *Proc. Eur. Conf. Model Driven Architecture – Foundations and Applications (ECMDA-FA 2005)*, number 3748 in LNCS, pages 77–92. Springer Verlag, 2005.

[8] Harald Störrle. A PROLOG-based Approach to Representing and Querying UML Models. In Philip Cox, Andrew Fish, and John Howse, editors, *Intl. Ws. Visual Languages and Logic (VLL'07)*, volume 274 of *CEUR-WS*, pages 71–84. CEUR, 2007. Available at `ftp.informatik.rwthaachen.de/Publications/CEUR-WS`.

[9] Harald Störrle. Large Scale Modeling Efforts. A Survey on Challenges and Best Practices. In *IASTED Intl. Conf. Software Engineering (SE'2007)*, pages 382–389. IASTED, 2007.

[10] Harald Störrle. VMQL: A Generic Visual Model Query Language. In Martin Erwig, Robert DeLine, and Mark Minas, editors, *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'09)*. IEEE Computer Society, 2009. to be published.

[11] Mathias Winder. MQ – Eine visuelle Query-Schnittstelle für Modelle, 2009. Bachelor thesis, Innsbruck University.

# Conservativity for a hierarchy of Euler and Venn reasoning systems

Koji Mineshima, Mitsuhiro Okada, and Ryo Takemura

Department of Philosophy, Keio University,
2-15-45 Mita, Minato-ku, Tokyo 108-8345, Japan.
{minesima,mitsu,takemura}@abelard.flet.keio.ac.jp

**Abstract**

This paper introduces a hierarchy of Euler and Venn diagrammatic reasoning systems in terms of their expressive powers in topological-relation-based formalization. At the bottom of the hierarchy is the Euler diagrammatic system introduced in Mineshima-Okada-Sato-Takemura [13, 12], which is expressive enough to characterize syllogistic reasoning in terms of unification and deletion rules. At the top of the hierarchy is a Venn diagrammatic system such as Swoboda-Allwein's Euler/Venn diagrammatic system [23]. In order to understand the hierarchy uniformly, we introduce an algebraic structure, which also provides another description of our unification rule of Euler diagrams. We prove that each system S' of the hierarchy is conservative over any lower system S with respect to validity—in the sense that S' is an extension of S, and the semantic consequence relations of S and S' are equivalent for diagrams of S. Furthermore, we prove that a region-based Venn diagrammatic system is conservative over our topological-relation-based Euler diagrammatic system with respect to provability.

## 1 Introduction

Euler diagrams were introduced by Euler (1768) [1] to represent logical relations among the terms of a syllogism by topological relations among circles. Given two Euler diagrams which represent the premises of a syllogism, the syllogistic inference can be naturally replaced by the task of manipulating the diagrams, in particular of unifying the diagrams and extracting information from them. For example, the well-known syllogism named "Barbara," i.e., *All $A$ are $B$ and All $B$ are $C$; therefore All $A$ are $C$*, can be represented diagrammatically as in Fig. 1.

Another well-known diagrammatic representation system for syllogistic reasoning is Venn diagrams. In Venn diagrams a novel syntactic device, namely *shading,* to represent emptiness plays a central role in place of the topological relations of Euler diagrams. Because of their expressive power and their uniformity in formalizing the manipulation of combining diagrams simply as the superposition of shadings, Venn diagrams have been very well studied. Cf. Venn-I, II systems of Shin [19], Spider

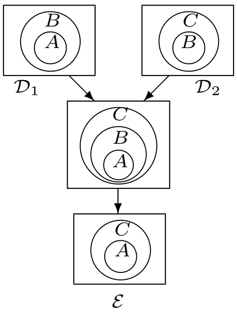


Fig. 1 Barbara with Euler diagrams

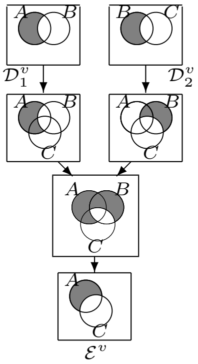


Fig. 2 Barbara with Venn diagrams

diagrams SD1 and SD2 of [9], [14], etc. For a recent survey, see [20]. However, the development of systems of Venn diagrams is obtained at the cost of clarity of Euler diagrams. As Venn [25] himself already pointed out, when more than three circles are involved, Venn diagrams fail in their main purpose of affording intuitive and sensible illustration. (For some discussions on visual disadvantages of Venn diagrams, see [8, 5]. See also [18] for our cognitive psychological experiments comparing linguistic, Euler diagrammatic, and Venn diagrammatic representations.) Recently, *Euler diagrams with shading* were introduced to make up for the shortcoming of Venn diagrams: E.g., Euler/Venn diagrams of [23, 24]; Spider diagrams ESD2 of [14] and SD3 of [10]. However, their abstract syntax and semantics are still defined in terms of regions, where shaded regions of Venn diagrams are considered as “missing” regions. That is, the idea of the *region-based* Euler diagrams is essentially along the same line as Venn diagrams.

We may point out the following complications of region-based formalization of diagrams:

1. In region-based diagrams, logical relations among circles are represented not simply by topological relations, but by the use of shading or missing regions, which makes the translations of categorical sentences uncomfortably complex. For example, *All $A$ are $B$* is expressed by a region-based diagram through a translation to the statement *There is nothing which is $A$ but not $B$* as seen in $\mathcal{D}_1^v$ of Fig. 2.

2. The inference rule of *unification*, which plays a central role in Euler diagrammatic reasoning, is defined by way of the superposition of Venn diagrams. For example, when we unify two region-based Euler diagrams as in $\mathcal{D}_1$ and $\mathcal{D}_2$ of Fig. 1, they are first transformed into Venn diagrams $\mathcal{D}_1^v$ and $\mathcal{D}_2^v$ of Fig. 2, respectively; then, by superposing the shaded regions of $\mathcal{D}_1^v$ and $\mathcal{D}_2^v$, and by deleting the circle $B$, the Venn diagram $\mathcal{E}^v$ is obtained, which is transformed into the region-based Euler diagram $\mathcal{E}$. In this way, processes of deriving conclusions are

often made complex, and hence less intuitive, in the region-based framework.

In contrast to the studies in the tradition of region-based diagrams, we proposed a novel approach in [13, 12] to formalize Euler diagrams in terms of topological relations. Our system has the following features and advantages:

1. Our diagrammatic syntax and semantics are defined in terms of *topological relations*, inclusion and exclusion relations, between two diagrammatic objects. This formalization makes the translations of categorical sentences natural and intuitive. Furthermore, our formalization makes it possible to represent a diagram by a simple ordered (or graph-theoretical) structure.

2. Our *unification* of two diagrams is formalized directly in terms of topological relations without making a detour to Venn diagrams. Thus, it can directly capture the inference process as illustrated in Fig. 1. We formalize the unification in the style of Gentzen's natural deduction, a well-known formalization of logical reasoning in symbolic logic, which is intended to be as close as possible to actual reasoning (Gentzen [3]). This makes it possible to compare our Euler diagrammatic inference system directly with natural deduction system. Through such comparison, we can apply well-developed proof-theoretical approaches to diagrammatic reasoning. See [13] for such proof-theoretical analyses.

From a perspective of proof-theory, the contrast between the standpoints of the region-based framework and the topological-relation-based framework can be understood as follows: At the level of representation, the contrast is analogous to the one between disjunctive (dually, conjunctive) normal formulas and implicational formulas; at the level of reasoning, the contrast is analogous to the one between resolution calculus style proofs and natural deduction style proofs.

From a perspective of cognitive psychology, our system is designed not just as an alternative of usual linguistic/symbolic representations; we make the best use of advantages of diagrammatic representations so that inherent definiteness or specificity of diagrams can be exploited in actual reasoning. See [18] for our experimental result, which shows that our Euler diagrams are more effective than Venn diagrams or linguistic representations in syllogism solving tasks.

We roughly review our topological-relation-based Euler diagrammatic representation system EUL in Section 2. (We also review our inference system GDS in Appendix A.) Although EUL is weaker in its expressive power than usual Venn diagrammatic systems (e.g. Shin's Venn-II [19], which is equivalent to the monadic first order logic in its expressive power), EUL is expressive enough to characterize basic logical reasoning such as syllogistic reasoning, see [12]. Our EUL-diagrams can be abstractly seen as algebraic (or graph-theoretical) structure, where inclusion relations between diagrammatic objects are reflexive transitive ordering relations, and exclusion relations are irreflexive symmetric relations. Based on this observation, in Section 3, we introduce EUL-structure, which provides another description and a verification of our unification rule of Appendix A. In Section 4, based on the EUL-structure, we introduce a hierarchy of Euler and Venn diagrammatic reasoning systems as seen in Fig. 3.
The most elementary system EUL considers only circles and points as diagrammatic objects; EUL is extended by considering intersection regions $X \cap Y$, union regions

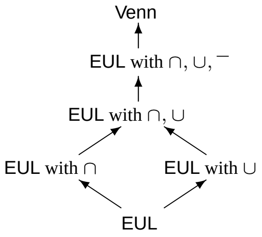


Fig. 3 Hierarchy of Euler and Venn systems

$X \cup Y$, and complement regions $\overline{X}$ as diagrammatic objects, respectively, as well as linking of points; Venn diagrams can be put at the top of the hierarchy of these extended systems. The algebraic structure thus obtained for Venn diagrams is essentially the directed acyclic graph of Swoboda-Allwein [23]. We prove that each system $\mathsf{S}'$ of the hierarchy is conservative over any lower system $\mathsf{S}$ with respect to validity—in the sense that $\mathsf{S}'$ is an extension of $\mathsf{S}$, and the semantic consequence relations of $\mathsf{S}$ and $\mathsf{S}'$ are equivalent for diagrams of $\mathsf{S}$. Moreover, we prove that a region-based Venn diagrammatic system is conservative over our topological-relation-based Euler diagrammatic system with respect to provability. We also give a procedure to transform a topological-relation-based EUL-diagram through an EUL-structure to a semantically equivalent region-based Venn diagram.

# 2 A diagrammatic representation system (EUL) for Euler circles and its set-theoretical semantics

In this section, we review our diagrammatic representation system EUL of [13, 12].

## 2.1 Diagrammatic syntax of EUL

The following definition of diagrams is slightly different from that of [13, 12] in that (1) we regard inclusion relation $\sqsubset$ as reflexive in this paper; (2) we exclude from EUL-diagrams only the empty diagram, on which no topological relation holds.

**Definition 2.1** An EUL**-diagram** is a plane ($\mathbb{R}^2$) with a finite number, at least one, of **named simple closed curves** (denoted by $A, B, C, \dots$) and **named points** (denoted by $a, b, c, \dots$), where each named simple closed curve or named point has a unique and distinct name. EUL-diagrams are denoted by $\mathcal{D}, \mathcal{E}, \mathcal{D}_1, \mathcal{D}_2, \dots$.
An EUL-diagram consisting of at most two objects is called a **minimal diagram**. Minimal diagrams are denoted by $\alpha, \beta, \gamma, \dots$.

In what follows, a named simple closed curve is called a *named circle*. Moreover, named circles and named points are collectively called *objects*, and denoted by $s, t, u, \dots$. We use a rectangle to represent a plane for an EUL-diagram.[1]

[1] Several Euler diagrammatic representation systems impose some additional conditions for well-formed diagrams. E.g., at most two circles meet at a single point, no tangential meetings or concurrency etc. Cf. e.g., [22]. However, for simplicity of the definition, those are all considered to be well-formed in EUL.

We define the following binary topological relations between diagrammatic objects[2]:

**Definition 2.2** **$\mathsf{EUL}$-relations** are the following binary relations between diagrammatic objects:

| | |
|---|---|
| $A \sqsubset B$ | "the interior of $A$ is *inside of* the interior of $B$," |
| $A \mathrel{\vdash\!\dashv} B$ | "the interior of $A$ is *outside of* the interior of $B$," |
| $A \bowtie B$ | "there is an intersection between the interior of $A$ and the interior of $B$," |
| $b \sqsubset A$ | "$b$ is *inside of* the interior of $A$," |
| $b \mathrel{\vdash\!\dashv} A$ | "$b$ is *outside of* the interior of $A$," |
| $a \mathrel{\vdash\!\dashv} b$ | "$a$ is *outside of* $b$ (i.e. $a$ is not located at the point of $b$)." |

We call $\bowtie$-relation *crossing* relation.

$\mathsf{EUL}$-relations $\mathrel{\vdash\!\dashv}$ and $\bowtie$ are symmetric, while $\sqsubset$ is not. In this paper, we consider $\sqsubset$-relation as reflexive by allowing $s \sqsubset s$ for each object $s$.

**Proposition 2.3** *Let $\mathcal{D}$ be an $\mathsf{EUL}$-diagram. For any distinct objects $s$ and $t$ of $\mathcal{D}$, exactly one of the $\mathsf{EUL}$-relations $s \sqsubset t, t \sqsubset s, s \mathrel{\vdash\!\dashv} t, s \bowtie t$ holds.*

Observe that, by Proposition 2.3, for a given $\mathsf{EUL}$-diagram $\mathcal{D}$, the set of $\mathsf{EUL}$-relations holding on $\mathcal{D}$ is uniquely determined. We denote the set by $\mathsf{rel}(\mathcal{D})$. We also denote by $pt(\mathcal{D})$ the set of named points of $\mathcal{D}$, by $cr(\mathcal{D})$ the set of named circles of $\mathcal{D}$, and by $ob(\mathcal{D})$ the set of objects of $\mathcal{D}$.

Although in this section, $ob(\mathcal{D}) = pt(\mathcal{D}) \cup cr(\mathcal{D})$, in Section 4, diagrammatic objects are extended, in addition to named circles and points, by introducing intersection, union, and complement regions respectively.

The following properties, as well as Proposition 2.3, characterize $\mathsf{EUL}$-diagrams.

**Lemma 2.4** *Let $\mathcal{D}$ be an $\mathsf{EUL}$-diagram. Then for any objects (named circles or points) $s, t, u \in ob(\mathcal{D})$, we have the following:*

1. (Transitivity) *If $s \sqsubset t, t \sqsubset u \in \mathsf{rel}(\mathcal{D})$, then $s \sqsubset u \in \mathsf{rel}(\mathcal{D})$.*
2. ($\mathrel{\vdash\!\dashv}$-downward closedness) *If $s \mathrel{\vdash\!\dashv} t, u \sqsubset s \in \mathsf{rel}(\mathcal{D})$, then $u \mathrel{\vdash\!\dashv} t \in \mathsf{rel}(\mathcal{D})$.*
3. (Point determinacy) *For any point $x$ of $\mathcal{D}$, exactly one of $x \sqsubset s$ and $x \mathrel{\vdash\!\dashv} s$ is in $\mathsf{rel}(\mathcal{D})$.*
4. (Point minimality) *For any point $x$ ($\not\equiv s$) of $\mathcal{D}$, $s \sqsubset x \notin \mathsf{rel}(\mathcal{D})$.*

Equivalence between $\mathsf{EUL}$-diagrams is defined as follows. (See [13] for a more detailed explanation.)

**Definition 2.5** When any two objects of the same name appear in different diagrams (planes), we identify them up to isomorphism. Any $\mathsf{EUL}$-diagrams $\mathcal{D}$ and $\mathcal{E}$ such that $ob(\mathcal{D}) = ob(\mathcal{E})$ are **syntactically equivalent** when $\mathsf{rel}(\mathcal{D}) = \mathsf{rel}(\mathcal{E})$.

[2] We follow Gergonne [4] for our notations on topological relations $\sqsubset$ and $\mathrel{\vdash\!\dashv}$.

**Example 2.6 (Equivalence of diagrams)** For example, diagrams $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ of Fig. 4 are equivalent since $\mathsf{rel}(\mathcal{D}_1) = \mathsf{rel}(\mathcal{D}_2) = \mathsf{rel}(\mathcal{D}_3) = \mathsf{rel}(\mathcal{D}_4) = \{A \bowtie B, A \bowtie C, B \bowtie C, a \vdash\dashv A, a \sqsubset B, a \vdash\dashv C\}$. In the description of a set of relations, we usually omit the reflexive relation $s \sqsubset s$ for each object $s$.

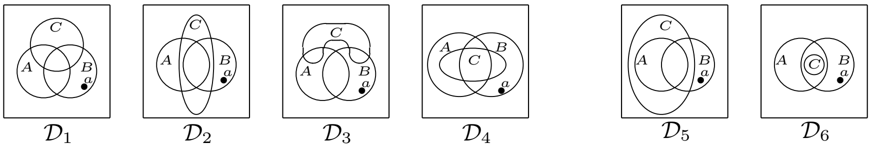


Fig. 4 Equivalence of EUL-diagrams.

On the other hand, $\mathcal{D}_1$ and $\mathcal{D}_5$ (resp. $\mathcal{D}_1$ and $\mathcal{D}_6$) are not equivalent since different EUL-relations hold on them: $A \sqsubset C$ holds on $\mathcal{D}_5$ in place of $A \bowtie C$ of $\mathcal{D}_1$ (resp. $C \sqsubset A$ and $C \sqsubset B$ hold on $\mathcal{D}_6$ in place of $A \bowtie C$ and $C \bowtie B$ of $\mathcal{D}_1$). Cf. Example 4.5 and 4.7 of Section 4, where $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$, and $\mathcal{D}_4$ are distinguished.

Our equation of diagrams may be explained in terms of a kind of "continuous transformation (deformation)" of named circles, which does not change any of the EUL-relations in a diagram. (See [13] for an explanation.)

In what follows, the diagrams which are syntactically equivalent are identified, and they are referred by a single name.

**Remark 2.7 (Expressive power of EUL)** Our equation of diagrams in the basic system EUL may seem to be counterintuitive since seemingly distinctive diagrams $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4$ of Example 2.6 are identified. [3]However, this slightly rough equation makes the description of unification of diagrams much simpler; see Appendix A. Furthermore, it is shown that EUL is expressive enough to characterize basic logical reasoning such as syllogistic reasoning; see [12]. In Section 4, we consider some extensions of EUL, where $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$, and $\mathcal{D}_4$ are distinguished by regarding intersection and union regions respectively as diagrammatic objects. See, in particular, Examples 4.5 and 4.7. Note that, by introducing new diagrammatic objects in a representation system, EUL-relations for these new objects are augmented, so that the system becomes more expressive. At the level of diagrammatic syntax, this means that more fine-grained distinctions between diagrams are made possible.

## 2.2 Set-theoretical semantics of EUL

Our semantics is distinct from usual ones, e.g., [6, 8, 24, 10] in that diagrams are interpreted in terms of binary relations. In order to interpret the EUL-relations $\sqsubset$ and $\vdash\dashv$ uniformly as the subset relation and the disjointness relation, respectively, we regard each point of EUL as a special circle which does not contain, nor cross, any other objects.

[3] This is also pointed out in Fish-Flower [2] as an drawback of the relation-based approach.

**Definition 2.8** A **model** $M$ is a pair $(U, I)$, where $U$ is a non-empty set (the domain of $M$), and $I$ is an interpretation function which assigns to each named circle or point a non-empty subset of $U$ such that $I(a)$ is a singleton for any named point $a$, and $I(a) \neq I(b)$ for any points $a, b$ of distinct names.

Note that we assign a non-empty set to each named circle. This condition is essential for our completeness. See the paragraph on the constraint for consistency in Appendix A and footnote 7 there.

**Definition 2.9** Let $\mathcal{D}$ be an $\mathsf{EUL}$-diagram. $M = (U, I)$ is a **model of** $\mathcal{D}$, written as $M \models \mathcal{D}$, if the following **truth-conditions** (1) and (2) hold: For all objects $s, t$ of $\mathcal{D}$, (1) $I(s) \subseteq I(t)$ if $s \sqsubset t$ holds on $\mathcal{D}$, and (2) $I(s) \cap I(t) = \emptyset$ if $s \vdash\!\dashv t$ holds on $\mathcal{D}$.

Note that when $s$ is a named point $a$, for some $e \in U$, $I(a) = \{e\}$, and the above $I(a) \subseteq I(t)$ of (1) is equivalent to $e \in I(t)$. Similarly, $I(a) \cap I(t) = \emptyset$ of (2) is equivalent to $e \notin I(t)$.

**Remark 2.10 (Semantic interpretation of $\bowtie$-relation)** By Definition 2.9, the $\mathsf{EUL}$-relation $\bowtie$ does not contribute to the truth-condition of $\mathsf{EUL}$-diagrams. Informally speaking, $s \bowtie t$ may be understood as $I(s) \cap I(t) = \emptyset$ *or* $I(s) \cap I(t) \neq \emptyset$, which is true in any model. Cf. also Remark 2.7.

**Definition 2.11** An $\mathsf{EUL}$-diagram $\mathcal{E}$ is a **semantically valid consequence** of $\mathsf{EUL}$-diagrams $\mathcal{D}_1, \ldots, \mathcal{D}_n$, written as $\mathcal{D}_1, \ldots, \mathcal{D}_n \models \mathcal{E}$, when the following holds: For any model $M$, if $M \models \mathcal{D}_1$ and ... and $M \models \mathcal{D}_n$, then $M \models \mathcal{E}$.

See Appendix A and [13] for our Generalized Diagrammatic Syllogistic inference system $\mathsf{GDS}$, whose completeness holds with respect to the semantics of this section.

# 3 EUL-structure

In this section, we introduce an algebraic structure called $\mathsf{EUL}$-structure for $\mathsf{EUL}$-diagrams.

**Definition 3.1** An **$\mathsf{EUL}$-structure** $(D, p(D), \sqsubset, \vdash\!\dashv)$ is a partially ordered structure, where $D$ is a set whose cardinality $\#D \geq 1$, and $p(D) \subseteq D$:

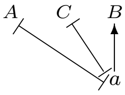
1. $\sqsubset$ is a reflexive transitive ordering relation on $D$.
2. $\vdash\!\dashv$ is an irreflexive symmetric relation on $D$.
3. ($\vdash\!\dashv$-downward closedness) For any $s, t, u \in D$, $s \vdash\!\dashv t$ and $t \sqsupset u$ imply $s \vdash\!\dashv u$.
4. (Point determinacy) For any $s \in D$ and $x \in p(D)$, $x \sqsubset s$ or $x \vdash\!\dashv s$.
5. (Point minimality) For any $s \in D$ and $x \in p(D)$ such that $s \not\equiv x$, $not(s \sqsubset x)$.

Cf. Lemma 2.4. Observe that the above properties (i), (ii), and (iii) imply that, for any distinct pair of elements of $D$, at most one of the relations $\sqsubset$ and $\vdash\dashv$ holds (cf. Proposition 2.3); because if both of them hold, say $s \sqsubset t$ and $s \vdash\dashv t$, the property (iii) implies $s \vdash\dashv s$, which contradicts the irreflexivity of $\vdash\dashv$-relation. [4]

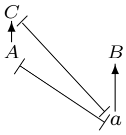
As seen in Section 2.1, given an EUL-diagram $\mathcal{D}$, the set $\mathsf{rel}(\mathcal{D})$ of relations holding on it is uniquely determined by Proposition 2.3. $\mathsf{rel}(\mathcal{D})$ can be regarded as an EUL-structure.

**Proposition 3.2** *Let $\mathcal{D}$ be an* EUL*-diagram. The set of* EUL*-relations* $\mathsf{rel}(\mathcal{D})$ *gives rise to an* EUL*-structure* $(ob(\mathcal{D}), pt(\mathcal{D}), \sqsubset, \vdash\dashv)$.

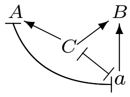
For example, $\mathsf{rel}(\mathcal{D}_1), \mathsf{rel}(\mathcal{D}_5)$ and $\mathsf{rel}(\mathcal{D}_6)$ of Fig. 4 in Example 2.6 are expressed graphically as follows: Here the ordering relations $\sqsubset$ are expressed by $\rightarrow$-edges.



$\mathsf{rel}(\mathcal{D}_1)$



$\mathsf{rel}(\mathcal{D}_5)$



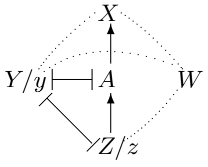
$\mathsf{rel}(\mathcal{D}_6)$

Observe that there is no edge for $\bowtie$-relation.

Now we describe the unification rule of Definition A.1 of Appendix A in terms of a graph-theoretical representation of EUL-diagrams, which may assist with the understanding and motivation of our unification rule.

**Proposition 3.3** *Let $\mathcal{D}$ be an* EUL*-diagram, and $\alpha$ be a minimal diagram. The set of* EUL*-relations* $\mathsf{rel}(\mathcal{D} + \alpha)$*, which is obtained by unifying $\mathcal{D}$ and $\alpha$, gives rise to an* EUL*-structure.*

*Proof.* In order to describe graphically the unification of EUL-diagrams $\mathcal{D}$ and $\alpha$, we focus on the shared object of $\mathcal{D}$ and $\alpha$, say $A$, and express the EUL-structure of $\mathsf{rel}(\mathcal{D})$ as follows:



$\mathsf{rel}(\mathcal{D})$

$\rightarrow$-edge denotes $\sqsubset$-relation

$\vdash\dashv$-edge denotes $\vdash\dashv$-relation

No edge for $\bowtie$-relation

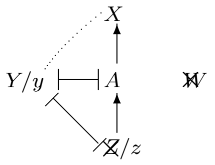
"$\cdots$" denotes one of $\sqsubset, \sqsupset, \vdash\dashv, \bowtie$

The variables $X, Y, Z, W$ (resp. $y, z$) are representative circles (resp. points) which are possibly related to $A$. When it makes no difference whether a possibly related object is circle or point, we denote the object as $Y/y$ (instead of simply writing $s$). Each dotted line between objects expresses that there may be one of the relations $\sqsubset, \sqsupset, \vdash\dashv, \bowtie$ between the objects. Note that there is no edge for each $\bowtie$-relation, as seen between $A$ and $W$. We omit the trivial transitive edge $Z \rightarrow X$ to avoid notational complexity. In the following description of each unification rule for $\mathcal{D}$ and $\alpha$, we give a graphical

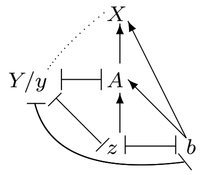
[4] Note that, by the properties (i)–(iii), an EUL-structure $(D, p(D), \sqsubset, \vdash\dashv)$ is an *event structure* of Nielsen-Plotkin-Winskel [15].

representation of the $\mathsf{EUL}$-structures of $\mathsf{rel}(\mathcal{D})$ in the left-hand graph, and $\mathsf{rel}(\mathcal{D}+\alpha)$ in the right-hand graph. We begin with $\mathsf{U3}$-rule since $\mathsf{U1}$ and $\mathsf{U2}$ rules are rather untypical cases:

$\mathsf{U3}$ Under the constraint of $\mathsf{U3}$-rule, there is no circle $Z$ such that $Z \sqsubset A$ holds, and no circle $W$ such that $A \bowtie W$ holds, which is expressed by × in the graph of $\mathsf{rel}(\mathcal{D})$. According to $\mathsf{U3}$-rule of Definition A.1, $\mathsf{rel}(\mathcal{D}+(b \sqsubset A))$ is represented by the graph on the right.



$\mathsf{rel}(\mathcal{D})$

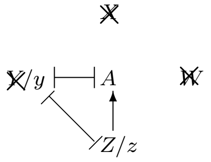


$\mathsf{rel}(\mathcal{D}+(b \sqsubset A))$

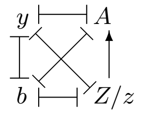
It is easily seen that $\mathsf{rel}(\mathcal{D}+(b \sqsubset A))$ is an $\mathsf{EUL}$-structure: I.e., the augmented edges do not violate the properties of $\mathsf{EUL}$-structure.

Note also that, without the constraint, i.e., if there is a circle $Z$ or $W$ as above, in order to preserve Point determinacy, we should fix a relation between $b$ and $Z$ (resp. $W$) to $\sqsubset$ or $\vdash\!\dashv$. However, neither of them is sound with respect to our formal semantics of $\mathsf{EUL}$.

$\mathsf{U4}$ Under the constraint of $\mathsf{U4}$-rule, there is no circle $X$ such that $A \sqsubset X$ holds, no circle $Y$ such that $A \vdash\!\dashv Y$ holds, and no circle $W$ such that $A \bowtie W$ holds, which is expressed by × in the graph of $\mathsf{rel}(\mathcal{D})$. According to $\mathsf{U4}$-rule of Definition A.1, $\mathsf{rel}(\mathcal{D}+(b \vdash\!\dashv A))$ is represented by the right hand graph below.



$\mathsf{rel}(\mathcal{D})$

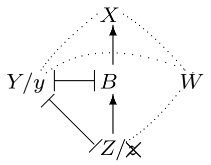


$\mathsf{rel}(\mathcal{D}+(b \vdash\!\dashv A))$

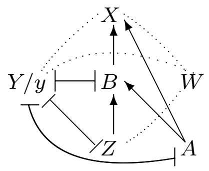
It is easily seen that $\mathsf{rel}(\mathcal{D}+(b \vdash\!\dashv A))$ is an $\mathsf{EUL}$-structure: I.e., the augmented edges do not violate the properties of $\mathsf{EUL}$-structure.

Without the constraint, i.e., if there is a circle $X, Y$ or $W$ as above, in order to preserve Point determinacy, we should fix a relation between $b$ and $X$ (resp. $Y, W$) to $\sqsubset$ or $\vdash\!\dashv$ in $\mathsf{rel}(\mathcal{D}+(b \vdash\!\dashv A))$. However, none of them is sound with respect to our semantics of $\mathsf{EUL}$.

$\mathsf{U5}$ Under the constraint of $\mathsf{U5}$-rule, there is no point $z$ such that $z \sqsubset B$ holds. According to $\mathsf{U5}$-rule of Definition A.1, $\mathsf{rel}(\mathcal{D}+(A \sqsubset B))$ is represented by the right hand graph below.



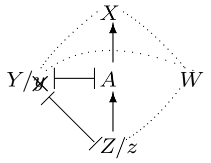
$\mathsf{rel}(\mathcal{D})$



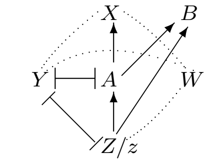
$\mathsf{rel}(\mathcal{D} + (A \sqsubset B))$

Without the constraint, i.e., if there is a point $z$ as above, in order to preserve Point determinacy, we should fix a relation between $z$ and $A$ to $\sqsubset$ or $\vdash\!\dashv$. However, none of them is sound with respect to our semantics of EUL.

U6 Under the constraint of U6-rule, there is no point $y$ such that $y \vdash\!\dashv A$ holds. According to U6-rule of Definition A.1, $\mathsf{rel}(\mathcal{D} + (A \sqsubset B))$ is represented by the right hand graph below.



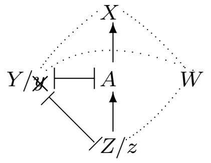
$\mathsf{rel}(\mathcal{D})$



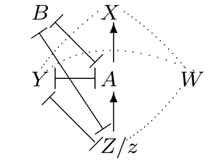
$\mathsf{rel}(\mathcal{D} + (A \sqsubset B))$

Without the constraint, i.e., if there is a point $y$ as above, in order to preserve Point determinacy, we should fix a relation between $y$ and $B$ to $\sqsubset$ or $\vdash\!\dashv$. However, none of them is sound with respect to our semantics of EUL.

U7 Under the constraint of U7-rule, there is no point $y$ such that $y \vdash\!\dashv A$ holds. According to U7-rule of Definition A.1, $\mathsf{rel}(\mathcal{D} + (A \vdash\!\dashv B))$ is represented by the right hand graph below.



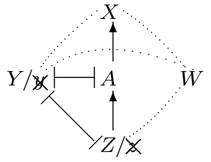
$\mathsf{rel}(\mathcal{D})$



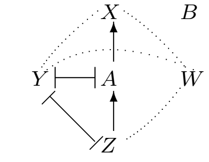
$\mathsf{rel}(\mathcal{D} + (A \vdash\!\dashv B))$

Without the constraint, i.e., if there is a point $y$ as above, in order to preserve Point determinacy, we should fix a relation between $y$ and $B$ to $\sqsubset$ or $\vdash\!\dashv$. However, none of them is sound with respect to our semantics of EUL.

U8 Under the constraint of U8-rule, there is no point in $\mathcal{D}$. According to U8-rule of Definition A.1, $\mathsf{rel}(\mathcal{D} + (A \bowtie B))$ is represented by the right hand graph below.



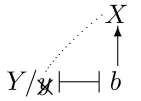
$\mathsf{rel}(\mathcal{D})$



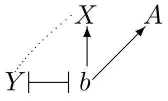
$\mathsf{rel}(\mathcal{D} + (A \bowtie B))$

Without the constraint, i.e., if there is a point $y$ or $z$ as above, in order to preserve Point determinacy, we should fix a relation between $y$ (resp. $z$) and $B$ to $\sqsubset$ or $\vdash\!\dashv$. However, none of them is sound with respect to our semantics of $\mathsf{EUL}$.

U1 Under the constraint of U1-rule, there is no point $y$ in $\mathcal{D}$ other than $b$. According to U1-rule, $\mathsf{rel}(\mathcal{D} + (b \sqsubset A))$ is represented by the right hand graph below.



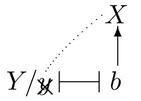
$\mathsf{rel}(\mathcal{D})$



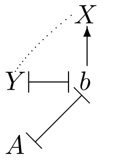
$\mathsf{rel}(\mathcal{D} + (b \sqsubset A))$

Without the constraint, i.e., if there is a point $y$ as above, in order to preserve Point determinacy, we should fix a relation between $y$ and $A$ to $\sqsubset$ or $\vdash\!\dashv$. However, none of them is sound with respect to our semantics of $\mathsf{EUL}$.

U2 Under the constraint of U2-rule, there is no point $y$ in $\mathcal{D}$ other than $b$. According to U2-rule, $\mathsf{rel}(\mathcal{D} + (b \vdash\!\dashv A))$ is represented by the right hand graph below.



$\mathsf{rel}(\mathcal{D})$

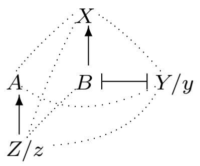


$\mathsf{rel}(\mathcal{D} + (b \vdash\!\dashv A))$

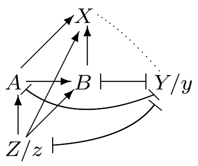
Without the constraint, i.e., if there is a point $y$ as above, in order to preserve Point determinacy, we should fix a relation between $y$ and $A$ to $\sqsubset$ or $\vdash\!\dashv$. However, none of them is sound with respect to our semantics of $\mathsf{EUL}$.

In U9, U10 rules of Definition A.1, the unified diagrams $\mathcal{D}$ and $\alpha$ share two circles, which makes the graphical description of $\mathsf{rel}(\mathcal{D})$ complicated. In order to avoid notational complexity, we omit irrelevant objects and edges, which are retained after the application of U9 and U10 rule, respectively.

U9 Under the constraint of U9-rule, there is no object $s$ such that $s \sqsubset A$ and $s \vdash\!\dashv B$ hold on $\mathcal{D}$, i.e., in the following description of $\mathsf{rel}(\mathcal{D})$, the dotted line between $Y/y$ and $A$ should not be $\rightarrow$, and the dotted line between $Z/z$ and $B$ should not be $\vdash\!\dashv$. According to U9-rule of Definition A.1, $\mathsf{rel}(\mathcal{D} + (A \sqsubset B))$ is represented by the right hand graph below.



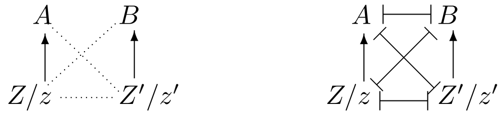
$\mathsf{rel}(\mathcal{D})$



$\mathsf{rel}(\mathcal{D} + (A \sqsubset B))$

Observe that, after the unification, some of the dotted lines of $\mathsf{rel}(\mathcal{D})$ are fixed to $\rightarrow$ or $\vdash\!\dashv$ in $\mathsf{rel}(\mathcal{D}+(A \sqsubset B))$ according to Definition A.1. We need to check that $\mathsf{rel}(\mathcal{D}+(A \sqsubset B))$ is an $\mathsf{EUL}$-structure; for example, if the dotted line between $A$ and $X$ in $\mathsf{rel}(\mathcal{D})$ is $A \vdash\!\dashv X$ (or $A \leftarrow X$), after the application of $\mathsf{U9}$-rule, there are two incompatible edges $\vdash\!\dashv$ (resp. $\leftarrow$) and $\rightarrow$ between $A$ and $X$, which violates the irreflexivity of the $\vdash\!\dashv$-relation of $\mathsf{EUL}$-structure. It is shown that, because of our constraint for $\mathsf{U9}$-rule, the dotted line between $A$ and $X$ is $\bowtie$ (i.e., no edge) or $\rightarrow$. Observe that, if we have $A \vdash\!\dashv X$ in $\mathsf{rel}(\mathcal{D})$, by the $\vdash\!\dashv$-downward closedness of $\mathsf{rel}(\mathcal{D})$, we have $Z/z \vdash\!\dashv B$ in $\mathsf{rel}(\mathcal{D})$, which contradicts the constraint. If we have $A \leftarrow X$ in $\mathsf{rel}(\mathcal{D})$, by the transitivity of $\mathsf{rel}(\mathcal{D})$, we have $A \leftarrow B$ in $\mathsf{rel}(\mathcal{D})$, which contradicts the presupposition of $\mathsf{U9}$-rule, i.e., there is no edge between $A$ and $B$ in $\mathsf{rel}(\mathcal{D})$. Thus the dotted line between $A$ and $X$ should be $\bowtie$ (i.e., no edge) or $\rightarrow$, either of which is compatible with the edge $A \rightarrow X$ in $\mathsf{rel}(\mathcal{D}+(A \sqsubset B))$. Similarly, it is shown that the other dotted lines of $\mathsf{rel}(\mathcal{D})$ are compatible with the edges of $\mathsf{rel}(\mathcal{D}+(A \sqsubset B))$. Then it is easily checked that $\mathsf{rel}(\mathcal{D}+(A \sqsubset B))$ satisfies Definition 3.1, and hence it is an $\mathsf{EUL}$-structure.

U10 Under the constraint of $\mathsf{U10}$-rule, there is no object $s$ such that $s \sqsubset A$ and $s \sqsubset B$ hold on $\mathcal{D}$, i.e., in the following graph of $\mathsf{rel}(\mathcal{D})$, the dotted line between $Z'/z'$ and $A$ (and also between $Z/z$ and $B$) should not be $\rightarrow$. According to $\mathsf{U10}$-rule, $\mathsf{rel}(\mathcal{D}+(A \vdash\!\dashv B))$ is represented by by the right hand graph below.



$\mathsf{rel}(\mathcal{D})$ $\mathsf{rel}(\mathcal{D}+(A \vdash\!\dashv B))$

We show that there are no incompatible edges in $\mathsf{rel}(\mathcal{D}+(A \vdash\!\dashv B))$. For the dotted line between $Z/z$ and $B$, it is not $\rightarrow$ by the constraint for $\mathsf{U10}$-rule. Furthermore, assume to the contrary that we have $Z/z \leftarrow B$ in $\mathsf{rel}(\mathcal{D})$. Then, by the transitivity of $\mathsf{rel}(\mathcal{D})$, we have $A \leftarrow B$ in $\mathsf{rel}(\mathcal{D})$, which contradicts the presupposition of $\mathsf{U10}$-rule, i.e., there is no edge between $A$ and $B$. Hence the dotted line between $Z/z$ and $B$ should be $\bowtie$ (i.e., no edge) or $\vdash\!\dashv$, either of which is compatible with the edge $Z/z \vdash\!\dashv B$ in $\mathsf{rel}(\mathcal{D}+(A \vdash\!\dashv B))$. Similarly, it is shown that the other two dotted lines of $\mathsf{rel}(\mathcal{D})$ are compatible with the edges of $\mathsf{rel}(\mathcal{D}+(A \vdash\!\dashv B))$. Then it is easily checked that $\mathsf{rel}(\mathcal{D}+(A \vdash\!\dashv B))$ satisfies Definition 3.1, and hence it is an $\mathsf{EUL}$-structure. ■

For a given $\mathsf{EUL}$-structure $(D, p(D), \sqsubset, \vdash\!\dashv)$, it can be shown that there is an $\mathsf{EUL}$-diagram $\mathcal{D}$ such that $\mathsf{rel}(\mathcal{D})$ is equivalent to $(D, p(D), \sqsubset, \vdash\!\dashv)$.

# 4 A hierarchy of EUL-diagrams and Venn diagrams

The representation system $\mathsf{EUL}$ is extended by introducing new diagrammatic objects, intersection, union, and complement regions, respectively. Extended systems are stratified in terms of their expressive powers.

In what follows, we do not mention named points explicitly, since any named point of EUL can be regarded as a special circle, which does not contain, nor cross, any other objects. If we allow a point (as a special circle) to cross other circles, it amounts to adopting linking between points, although it is slightly restricted compared with usual linking as in Shin [19], among others. [5]

We first extend EUL by considering intersection regions as diagrammatic objects. *Regions* of an EUL-diagram are defined recursively as usual, which are closed under intersection, union, and complement, provided that each is non-empty in a diagram. See, e.g., [10].

**Definition 4.1** A non-empty region $r$ of an EUL-diagram $\mathcal{D}$ is an **intersection region** when, for some $\{A_1, \dots, A_n\} \subseteq cr(\mathcal{D})$, $r = \bigcap_{1 \leq i \leq n} in(A_i)$, where $in(A_i)$ is the interior of circle $A_i$. An EUL-**diagrams with intersections** $\mathcal{D}$ is an EUL-diagram where each intersection region $r = \bigcap_{1 \leq i \leq n} in(A_i)$ has the name $\sqcap_{1 \leq i \leq n} A_i$, which is sometimes denoted by $\sqcap A_n$ for short. (In particular when $n = 1$, $\sqcap A_1 = A_1$.)

Note that, in an EUL-diagram with intersections, a region may have two names: For example, when $A \sqsubset B$ holds on $\mathcal{D}$, circle $A$ has another name, $A \sqcap B$.

We define an algebraic structure for EUL-diagrams with intersections.

**Definition 4.2** An **EUL-structure with greatest lower bounds (glbs)** $(D, \sqsubset, \dashv\vdash, \sqcap)$ is an EUL-structure, where for any subset $\{A_1, \dots, A_n\} \subseteq D$ such that $\neg \exists_{1 \leq j,k \leq n} (A_j \dashv\vdash A_k$ holds on $D)$, there is the greatest lower bound $\sqcap_{1 \leq i \leq n} A_i$.

Although we regard named points as special named circles, the operation $\sqcap$ is not applied to them.

**Lemma 4.3** *Let $\mathcal{D}$ be an EUL-diagram with intersections. The set of relations* $\mathsf{rel}(\mathcal{D})$ *gives rise to an EUL-structure with glbs.*

**Lemma 4.4 (EUL $\prec$ EUL+$\sqcap$)** *Let $(D, \sqsubset, \dashv\vdash)$ be an EUL-structure. It is extended, by introducing glbs, to an EUL-structure with glbs $(D^{\sqcap}, \sqsubset, \dashv\vdash, \sqcap)$.*

*Proof.* The domain $D^{\sqcap}$ is defined as follows:

$$D^{\sqcap} := D \cup \{\sqcap_{1 \leq i \leq n} A_i \mid \neg \exists_{1 \leq j,k \leq n} (A_j \dashv\vdash A_k \text{ holds on } D)\}$$

$\sqsubset$ and $\dashv\vdash$ relations on $D$ are preserved on $D^{\sqcap}$ and they are extended for any $\sqcap_{1 \leq i \leq n} A_i \in D^{\sqcap}$ as follows: Let $X, Y \in D^{\sqcap}$.

$$\begin{array}{lll}
\sqcap A_n \sqsubset \sqcap A_n & & \\
X \sqsubset \sqcap A_n & \textit{iff} & X \sqsubset A_i \text{ for all } 1 \leq i \leq n \\
\sqcap A_n \sqsubset X & \textit{iff} & A_i \sqsubset X \text{ for some } 1 \leq i \leq n \\
X \dashv\vdash Y & \textit{iff} & X \sqcap Y \notin D^{\sqcap}
\end{array}$$

[5] We exclude a crossing relation $c \bowtie d$ between distinct named points, since it amounts to $c = d$ *or* $c \neq d$ (cf. Remark 2.10) but we always assume $c \neq d$ in our framework.

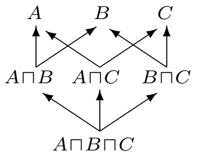
It is immediate that thus constructed $(\mathcal{D}^{\sqcap}, \sqsubset, \vDash\!\!\dashv, \sqcap)$ is an EUL-structure, which satisfies Definition 3.1, and $\sqcap A_n$ is the glb of Definition 4.2. ■

See also Example 4.17.

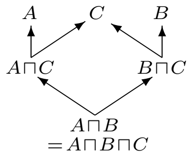
When $\mathcal{D}$ is an EUL-diagram, we denote $\mathcal{D}^{\sqcap}$ an EUL-diagram with intersections whose algebraic structure is constructed from the EUL-structure $\mathsf{rel}(\mathcal{D})$ by Lemma 4.4. We say that the diagram $\mathcal{D}^{\sqcap}$ is **obtained from** $\mathcal{D}$.

By introducing intersection regions as diagrammatic objects, EUL with intersections are more expressive than the basic EUL of Section 2.1. Let us see the following example.

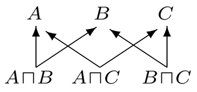
**Example 4.5 (EUL-diagrams with intersections)** The three diagrams $\mathcal{D}_1, \mathcal{D}_2$, and $\mathcal{D}_3$ of Fig. 4 in Example 2.6, which are identified in the original EUL, are distinguished when they are regarded as EUL-diagrams with intersections. The difference among the three diagrams is more clearly seen by drawing their EUL-structures with glbs. (Here, for reasons of simplicity, we omit the point $a$ and abbreviate $\vDash\!\!\dashv$-relation by stipulating that $X \vDash\!\!\dashv Y$ holds when $X \sqcap Y \notin \mathsf{rel}(\mathcal{D}^{\sqcap})$.)



$\mathsf{rel}(\mathcal{D}_1^{\sqcap})$



$\mathsf{rel}(\mathcal{D}_2^{\sqcap})$



$\mathsf{rel}(\mathcal{D}_3^{\sqcap})$

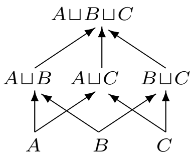
In a similar way as intersections, by considering union regions as diagrammatic objects we have another extension of EUL.

**Definition 4.6** An **EUL-diagrams with unions** $\mathcal{D}$ is an EUL-diagram where each union region $r = \bigcup_{1\leq i\leq n} in(A_i)$ has the name $\sqcup_{1\leq i\leq n} A_i$, provided that it is connected.

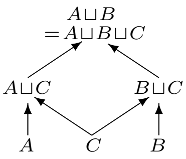
**EUL-structures with least upper bounds (lubs)** for EUL-diagrams with unions are defined in a similar way as EUL-structures with glbs.

EUL with unions is also more expressive than EUL.

**Example 4.7 (EUL-diagrams with unions)** $\mathcal{D}_1$ and $\mathcal{D}_4$ of Fig. 4 in Example 2.6 are distinguished when they are regarded as EUL-diagrams with unions. The EUL-structures with lubs for these two diagrams are represented by the following different structures.



$\mathsf{rel}(\mathcal{D}_1^{\sqcup})$



$\mathsf{rel}(\mathcal{D}_4^{\sqcup})$

**Definition 4.8** An EUL**-diagram with intersections and unions** $\mathcal{D}$ is an EUL-diagram with intersections where union regions also have names.

Note that we only consider intersection (resp. union) regions of circles, and we exclude other regions such as $(A \cap B) \cup (C \cap D)$.

EUL-structure with glbs and lubs are defined by combining EUL-structure with glbs and EUL-structure with lubs.

By considering the complement region of each circle as a diagrammatic object, we further introduce EUL-diagrams with intersections, unions, and complements.

**Definition 4.9** An EUL**-diagram with intersections, unions, and complements** $\mathcal{D}$ is an EUL-diagram with intersections and unions, where each complement $\overline{A}$ of a circle $A$, i.e., the exterior of $A$, has the name $\overline{A}$.

EUL-structures for EUL-diagrams with $\cap, \cup, ^{-}$ are defined as follows.

**Definition 4.10** An EUL**-structure with glbs, lubs, and complements** $(D, \sqsubset, \dashv\vdash, \sqcap, \sqcup, ^{-})$ is an EUL-structure with glbs and lubs $(D, \sqsubset, \dashv\vdash, \sqcap, \sqcup)$ where, for each $A \in D$ which is not of the form $\sqcap C_j$ nor $\sqcup C_j$ ($j \geq 2$), the complement $\overline{A}$ of $A$ is defined in $D$.

Although we regard named points as special named circles, the operations $\sqcap, \sqcup$, and $^{-}$ are not applied to points.

**Lemma 4.11** *Let $\mathcal{D}$ be an* EUL*-diagram with $\cap, \cup, ^{-}$. The set of relations* $\mathsf{rel}(\mathcal{D})$ *gives rise to an* EUL*-structure with glbs, lubs, and complements.*

**Lemma 4.12 (**EUL+$\sqcap$+$\sqcup$ $\prec$ EUL+$\sqcap$+$\sqcup$+$^{-}$**)** *Let* $(D^{\square}, \sqsubset, \dashv\vdash, \sqcap, \sqcup)$ *be an* EUL*-structure with glbs and lubs. It is extended, by introducing complements, to an* EUL*-structure with glbs, lubs, and complements* $(D^{c}, \sqsubset, \dashv\vdash, \sqcap, \sqcup, ^{-})$.

*Proof.* The domain $D^c$ is defined by adding complement $\overline{A}$ for each $A \in D^{\square}$ which is not of the form $\sqcap C_j$ nor $\sqcup C_j$ ($j \geq 2$), and by extending glbs (of the form $(\sqcap B_j) \sqcap (\sqcap\overline{A_i})$) and lubs (of the form $(\sqcup B_j) \sqcup (\sqcup\overline{A_i})$) in a similar way as Lemma 4.4.
$\sqsubset$ and $\dashv\vdash$ relations on $D^{\square}$ are preserved on $D^c$ and they are extended as follows:
For any $A, B \in D^c$ not of the form $\sqcap C_j$ nor $\sqcup C_j$ ($j \geq 2$),

$A \dashv\vdash \overline{A}$
$\overline{A} \sqsubset \overline{B}$ *iff* $B \sqsubset A$ in $D^{\square}$
$A \sqsubset \overline{B}$ and $B \sqsubset \overline{A}$ *iff* $A \dashv\vdash B$ in $D^{\square}$

For any $X, Y \in D^c$ of the form $(\sqcap B_j) \sqcap (\sqcap\overline{A_i})$ (resp. $(\sqcup B_j) \sqcup (\sqcup\overline{A_i})$), $\sqsubset$ and $\dashv\vdash$ relations are extended to be closed under $\sqcap$ and $\sqcup$ in a similar way as Lemma 4.4. ■

See also Example 4.17.

Euler/Venn diagrams of Swoboda-Allwein [23] are obtained by adding shading of minimal regions and linking of points to EUL-diagrams with $\cap, \cup, ^{-}$. [6]

[6] There are some differences between our system and Swoboda-Allwein's system: (i) we allow one circle to cross with another circle any number of times; (ii) we allow union regions as diagrammatic objects, which does not increase expressive power as compared to Swoboda-Allwein's system; (iii) we do not allow a circle to be completely shaded given our definition of semantics, where each circle denotes a non-empty set.

EUL-structures for Euler/Venn diagrams, which we call **Venn-structures**, are the directed acyclic graphs DAGs of Swoboda-Allwein [23].

**Lemma 4.13 (EUL$+\sqcap+\sqcup+^{-} \prec$ Venn)** *Let $(D^c, \sqsubset, \vdash\dashv, \sqcap, \sqcup, ^{-})$ be an EUL-structure with glbs, lubs, and complements. It is extended to a Venn-structure $D^v$ of Swoboda-Allwein [23] by introducing shading and linking.*

When $\mathcal{D}$ is an EUL-diagram, we denote by $\mathcal{D}^v$ (resp. $\mathcal{D}^{\sqcap}, \mathcal{D}^{\sqcup}, \mathcal{D}^{\square}, D^c$) an Euler/Venn diagram (resp. EUL-diagram with intersections, unions, intersections and unions, intersections and unions and complements) whose algebraic structure is constructed from the EUL-structure $\mathsf{rel}(\mathcal{D})$ by Lemma 4.4, 4.12, and 4.13. We say that the diagram $\mathcal{D}^v$ (resp. $\mathcal{D}^{\sqcap}, \mathcal{D}^{\sqcup}, \mathcal{D}^{\square}, D^c$) is **obtained from** $\mathcal{D}$.

Various extensions of EUL introduced so far can be summarized by the following **EUL-hierarchy**:

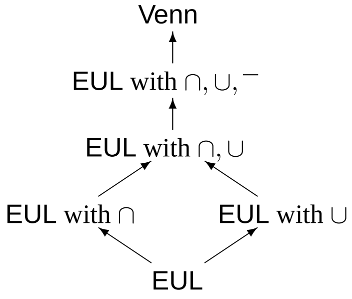


Fig. 5 EUL-hierarchy

Note that the semantics of EUL of Section 2.2 is essentially the same as the semantics of Venn diagrams (e.g. [10, 19]), where the interpretation function $I$ of circles is naturally extended to interpret regions: $I(\sqcap X_i) = \bigcap I(X_i)$, $I(\sqcup X_i) = \bigcup I(X_i)$, and $I(\overline{A}) = U \setminus I(A)$. Note that the denotations of intersections, unions, and complements are not assumed to be non-empty, while those of circles and points are non-empty.

Thus when $\mathcal{D}^*$ is a diagram obtained from an EUL-diagram $\mathcal{D}$ for $* \in \{\sqcap, \sqcup, \square, c, v\}$, $\mathcal{D}$ and $\mathcal{D}^*$ are semantically equivalent since any relation of $\mathcal{D}$ is preserved in $\mathcal{D}^*$ by constructions given in Lemmas 4.4, 4.12, and 4.13:

**Lemma 4.14** *Let $\mathcal{D}$ be an EUL-diagram. For each $* \in \{\sqcap, \sqcup, \square, c, v\}$, let $\mathcal{D}^*$ be a diagram obtained from $\mathcal{D}$. For any model $M$, $M \models \mathcal{D}^*$ if and only if $M \models \mathcal{D}$.*

Based on Lemma 4.14, it is shown that each system of EUL-hierarchy is conservative over any lower system with respect to validity. We denote by $\boldsymbol{\mathcal{D}}$ a sequence of diagrams $\mathcal{D}_1, \ldots, \mathcal{D}_n$.

**Proposition 4.15 (Semantic conservativity)** *Let $S'$ and $S$ be any systems of the EUL-hierarchy such that $S'$ is an extension of $S$. Let $\boldsymbol{\mathcal{D}}, \mathcal{E}$ be diagrams of $S$, and $\boldsymbol{\mathcal{D}}^*, \mathcal{E}^*$ be diagrams of $S'$ obtained from $\boldsymbol{\mathcal{D}}, \mathcal{E}$ for $* \in \{\sqcap, \sqcup, \square, c, v\}$. Then $\boldsymbol{\mathcal{D}}^* \models \mathcal{E}^*$ iff $\boldsymbol{\mathcal{D}} \models \mathcal{E}$.*

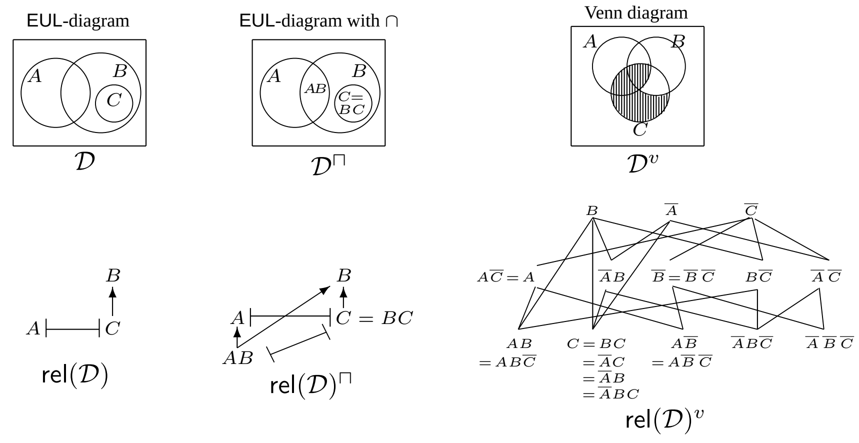
In parallel to the extensions of representation system EUL, we can obtain extended inference systems of GDS of Appendix A. It can be shown that each extended system is a conservative extension of the most elementary GDS with respect to provability. In particular, for Euler/Venn diagrammatic inference system of Swoboda-Allwein [24], we have the following conservativity theorem:

**Theorem 4.16 (Conservativity)** *Let $\boldsymbol{\mathcal{D}}$ and $\mathcal{E}$ be* EUL*-diagrams such that $\boldsymbol{\mathcal{D}}$ has a model. If $\mathcal{E}^v$ is provable from $\boldsymbol{\mathcal{D}}^v$ in Euler/Venn diagrammatic system, then $\mathcal{E}$ is provable from $\boldsymbol{\mathcal{D}}$ in* GDS*.*

*Proof.* Let $\boldsymbol{\mathcal{D}}^v \vdash \mathcal{E}^v$ in Euler/Venn diagrammatic inference system. By soundness (cf. [24]) we have, for any model $M$, $M \models \boldsymbol{\mathcal{D}}^v \Rightarrow M \models \mathcal{E}^v$. Assume $M \models \boldsymbol{\mathcal{D}}$. Then we have $M \models \boldsymbol{\mathcal{D}}^v$ by Lemma 4.14. Thus we have $M \models \mathcal{E}^v$, that is, $M \models \mathcal{E}$. Hence, by the completeness (Theorem A.2) of GDS, we have $\boldsymbol{\mathcal{D}} \vdash \mathcal{E}$ in GDS. ■

The constructions of extensions of EUL-structures given in Lemma 4.4, 4.12, and 4.13 provide a procedure to transform an EUL-diagram to a Venn diagram. Let us see the following example:

**Example 4.17** Let $\mathcal{D}$ be an EUL-diagram such that $\mathsf{rel}(\mathcal{D}) = \{A \bowtie B, A \vdash\!\dashv C, C \sqsubset B\}$. By transforming the EUL-structure $\mathsf{rel}(\mathcal{D})$ through an EUL-structure with glbs $\mathsf{rel}(\mathcal{D})^{\sqcap}$, we obtain a Venn-structure $\mathsf{rel}(\mathcal{D})^v$. In $\mathsf{rel}(\mathcal{D})^{\sqcap}$ and $\mathsf{rel}(\mathcal{D})^v$ below, we omit $\sqcap$ symbol and write $AB$ for $A \sqcap B$. In $\mathsf{rel}(\mathcal{D})^v$, we further omit lubs and $\vdash\!\dashv$-relation, and represent arrows by lines in a hierarchical structure. By extracting minimal unshaded regions ($AB\overline{C}, \overline{A}BC, A\overline{B}\,\overline{C}, \overline{A}B\overline{C}, \overline{A}\,\overline{B}\,\overline{C}$) from $\mathsf{rel}(\mathcal{D})^v$, we obtain a Venn diagram $\mathcal{D}^v$, which is semantically equivalent to the original EUL-diagram $\mathcal{D}$.


EUL-diagram
A
B
C
$\mathcal{D}$
EUL-diagram with ∩
A
AB
B
C=BC
$\mathcal{D}^{\sqcap}$
Venn diagram
A
B
C
$\mathcal{D}^v$
B
A
C
$\mathsf{rel}(\mathcal{D})$
B
A
C = BC
AB
$\mathsf{rel}(\mathcal{D})^{\sqcap}$
B
$\overline{A}$
$\overline{C}$
$A\overline{C} = A$
$\overline{A}B$
$\overline{B} = \overline{B}\,\overline{C}$
$B\overline{C}$
$\overline{A}\,\overline{C}$
$AB = AB\overline{C}$
$C = BC = \overline{A}C = \overline{A}B = \overline{A}BC$
$A\overline{B} = A\overline{B}\,\overline{C}$
$\overline{A}B\overline{C}$
$\overline{A}\,\overline{B}\,\overline{C}$
$\mathsf{rel}(\mathcal{D})^v$

In this paper, we introduced a hierarchy of Euler and Venn diagrammatic reasoning systems in terms of their expressive powers in our topological-relation-based formalization. Because of the space limitation in this paper, we discuss our extensions of EUL mainly at the level of representation and semantics. This is why our conservativity results for these systems (Proposition 4.15) are kept at the level of semantics. We leave our explicit formalization of diagrammatic inference systems for EUL-diagrams with intersections, with unions, with complements, respectively, as future work.

## References

[1] L. Euler, Lettres à une Princesse d'Allemagne sur Divers Sujets de Physique et de Philosophie, Saint-Pétersbourg: De l'Académie des Sciences, 1768. (English Translation by H. Hunter, 1997, *Letters of Euler to a German Princess on Different Subjects in Physics and Philosophy*, Thoemmes Press, 1997.)

[2] A. Fish and J. Flower, Abstractions of Euler Diagrams, Electronic Notes in Theoretical Computer Science, 134, 77-101, 2005.

[3] G. Gentzen, Investigations into logical deduction. In M. E. Szabo, ed., *The collected Papers of Gerhard Gentzen*, 1969. (Originally published as Unter suchungen uber das logische Scliessen, *Mathematische Zetischrift* 39 (1935): 176-210, 405-431.)

[4] J. D. Gergonne, Essai de dialectique rationelle, *Annales de mathématiques pures et appliqées*, 7, 189-228, 1817.

[5] J. Gil, J. Howse, E. Tulchinsky, Positive Semantics of Projections in Venn-Euler Diagrams, *Journal of Visual Languages and Computing*, 13(2), 197-227, 2002.

[6] E. Hammer, *Logic and Visual Information*, CSLI Publications, 1995.

[7] E. Hammer and N. Danner, Towards a model theory of diagrams, in G. Allwein and J. Barwise, eds., *Working Papers on Diagrams and Logic*, Bloomington: Indiana University, 1993.

[8] E. Hammer and S.-J. Shin, Euler's visual logic, *History and Philosophy of Logic*, 19, 1-29, 1998.

[9] J. Howse, F. Molina, J. Taylor, SD2: A Sound and Complete Diagrammatic Reasoning System, *2000 IEEE International Symposium on Visual Languages*, 127-134, 2000.

[10] J. Howse, G. Stapleton, and J. Taylor, Spider Diagrams, *LMS Journal of Computation and Mathematics*, Volume 8, 145-194, London Mathematical Society, 2005.

[11] B. Meyer, Diagrammatic evaluation of visual mathematical notations, in *Diagrammatic Representation and Reasoning*, P. Olivier, M. Anderson, and B. Meyer (Eds.), Springer, 261-277, 2001.

[12] K. Mineshima, M. Okada, Y. Sato, and R. Takemura, Diagrammatic Reasoning System with Euler Circles: Theory and Experiment Design, *Diagrams 08*, LNAI, Vol. 5223, Springer, 188-205, 2008.

[13] K. Mineshima, M. Okada, and R. Takemura, A Diagrammatic Reasoning System with Euler Circles, 2009, to be submitted.

[14] F. Molina, Reasoning with extended Venn-Peirce diagrammatic systems, Ph. D Thesis, University of Brighton, 2001.

[15] M. Nielsen, G. Plotkin, and G. Winskel, Petri Nets, Event Structures and Domains, Part I, *Theoretical Computer Science*, Vol. 13, No. 1, pages 85-108, 1980.

[16] C. S. Peirce, *Collected Papers IV*, Harvard University Press, 1897/1933.

[17] D. Prawitz, *Natural Deduction*, Almqvist & Wiksell, 1965 (Dover, 2006).

[18] Y. Sato, K. Mineshima, R. Takemura, and M. Okada, How can Euler diagrams improve syllogistic reasoning?, poster presented at CogSci 2009, Amsterdam, at the VU University Amsterdam, July 29th to August 1st, 2009.

[19] S.-J. Shin, *The Logical Status of Diagrams*, Cambridge University Press, 1994.

[20] G. Stapleton, A survey of reasoning systems based on Euler diagrams, *Euler 2004*, Electronic Notes in Theoretical Computer Science, 134(1), 127-151, 2005.

[21] G. Stapleton, J. Masthoff, J. Flower, A. Fish, and J. Southern, Automated Theorem Proving in Euler Diagram Systems, *Journal of Automated Reasoning*, volume 39 , issue 4, 431-470, 2007.

[22] G. Stapleton, P. Rodgers, J. Howse, J. Taylor, Properties of Euler diagrams, *ECEASST 7*, 2-16, 2007.

[23] N. Swoboda and G. Allwein, Using DAG transformations to verify Euler/Venn homogeneous and Euler/Venn FOL heterogeneous rules of inference, *Software and System Modeling*, 3(2), 136-149, 2004.

[24] N. Swoboda and G. Allwein, Heterogeneous reasoning with Euler/Venn diagrams containing named constants and FOL, *Euler Diagrams 2004*, ENTCS, Volume 134, Elsevier, 153-187, 2005.

[25] J. Venn, *Symbolic Logic*, Macmillan, 1881.

# A Diagrammatic inference system GDS

In this section, we review Generalized Diagrammatic Syllogistic inference system GDS of [13, 12], which consists of two inference rules: *unification* and *deletion*. In order to motivate our definition of *unification*, let us consider the following question: Given the following diagrams $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$ of Fig. 6, what diagrammatic information on $A, B$ and $c$ can be obtained? (In what follows, in order to avoid notational complexity in a diagram, we express each named point, say $\overset{c}{\bullet}$, simply by its name $c$.) Fig. 6 represents a way of solving the question.

In Fig. 6, at the first step, two diagrams $\mathcal{D}_1$ and $\mathcal{D}_2$ are unified to obtain $\mathcal{D}_1 + \mathcal{D}_2$, where point $c$ in $\mathcal{D}_1$ and $\mathcal{D}_2$ are identified, and $B$ is added to $\mathcal{D}_1$ so that $c$ is inside of $B$ and $B$ overlaps with $A$ without any implication of a relationship between $A$ and $B$. We formalize such cases, where two given diagrams share one object, by U1–U8 rules of group (I) of Definition A.1. At the second step, $\mathcal{D}_1 + \mathcal{D}_2$ is combined with another diagram $\mathcal{D}_3$ to obtain $(\mathcal{D}_1 + \mathcal{D}_2) + \mathcal{D}_3$. Note that the diagrams $\mathcal{D}_1 + \mathcal{D}_2$ and $\mathcal{D}_3$ share two circles $A$ and $B$: $A \bowtie B$ holds on $\mathcal{D}_1 + \mathcal{D}_2$ and $A \sqsubset B$ holds on $\mathcal{D}_3$. Since the semantic information of $A \sqsubset B$ on $\mathcal{D}_3$ is more accurate than that of $A \bowtie B$ on $\mathcal{D}_1 + \mathcal{D}_2$, according to our semantics of EUL (recall that $A \bowtie B$ means just “true” in

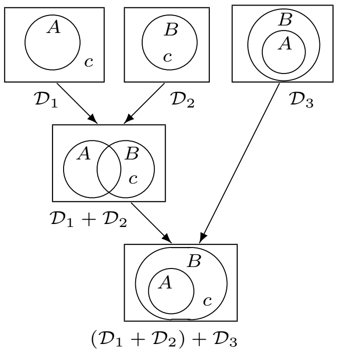

A
c
B
c
B
A
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_3$
A
B
c
$\mathcal{D}_1 + \mathcal{D}_2$
B
A
c
$(\mathcal{D}_1 + \mathcal{D}_2) + \mathcal{D}_3$

Fig. 6 Unification

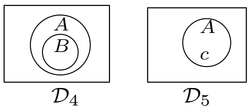


Fig. 7 Indeterminacy

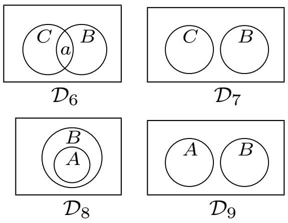


Fig. 8 Inconsistency

our semantics), one keeps the relation $A \sqsubset B$ in the unified diagram $(\mathcal{D}_1 + \mathcal{D}_2) + \mathcal{D}_3$. We formalize such cases, where two given diagrams share two objects, by U9–U10 rules of group (II) of Definition A.1. Observe that the unified diagram $(\mathcal{D}_1 + \mathcal{D}_2) + \mathcal{D}_3$ of Fig. 6 represents the information of these diagrams $\mathcal{D}_1, \mathcal{D}_2$, and $\mathcal{D}_3$, that is, their *conjunction*.

We impose two kinds of constraints on unification. One is the *constraint for determinacy*, which blocks the disjunctive ambiguity with respect to locations of named points. For example, two diagrams $\mathcal{D}_4$ and $\mathcal{D}_5$ in Fig. 7 are not permitted to be unified into one diagram since the location of the point $c$ is not determined (it can be inside $B$ or outside $B$). The other is the *constraint for consistency*, which blocks representing inconsistent information in a single diagram. For example, the diagrams $\mathcal{D}_6$ and $\mathcal{D}_7$ (resp. $\mathcal{D}_8$ and $\mathcal{D}_9$) in Fig. 8 are not permitted to be unified since they contradict each other. Recall that each circle is interpreted by non-empty set in our semantics of Definition 2.8, and hence $\mathcal{D}_8$ and $\mathcal{D}_9$ are also incompatible. [7]

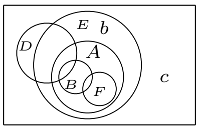
We formalize our unification [8] of two diagrams by restricting one of them to be a *minimal diagram*, except for one rule called the Point Insertion-rule. Our completeness (Theorem A.2) ensures that any diagrams $\mathcal{D}_1, \ldots, \mathcal{D}_n$ may be unified, under the constraints for determinacy and consistency, into one diagram whose semantic information is equivalent to the conjunction of that of $\mathcal{D}_1, \ldots, \mathcal{D}_n$. We give a formal description of inference rules in terms of EUL-relations: Given a diagram $\mathcal{D}$ and a minimal diagram $\alpha$, the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ for the unified diagram $\mathcal{D} + \alpha$ is defined. It is easily checked that the set $\mathsf{rel}(\mathcal{D} + \alpha)$ satisfies the properties of Lemma 2.4 according to our constraints for determinacy and consistency, and hence locations of points are determined in a unified diagram. (See also Section 3, where we give a

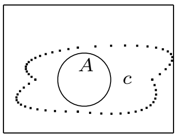
[7] In place of our syntactic constraint, it is possible to allow unification of inconsistent diagrams such as $\mathcal{D}_6$ and $\mathcal{D}_7$ (resp. $\mathcal{D}_8$ and $\mathcal{D}_9$) by extending GDS with an inference rule corresponding to the absurdity rule of Gentzen's natural deduction system: We can infer any diagram from a pair of inconsistent diagrams. (For natural deduction systems, see, for example, [3, 17].) Such a rule is introduced in, for example, [10] for spider diagrams; [7] for Venn diagrams; [23, 24] for Euler/Venn diagrams. However, such a rule requires linguistic symbol, say $\perp$, or some arbitrary convention to represent inconsistency, and hence we prefer our syntactic constraint in our framework of a diagrammatic inference system.

[8] The following definition of inference rules of GDS is slightly different from that of [13, 12] since we regard $\sqsubset$-relation as reflexive relation in this paper.

graph-theoretical representation of unification.)

For a better understanding of our unification rule, we also give a schematic diagrammatic representation and a concrete example of each rule. In the schematic representation of diagrams, to indicate the occurrence of some objects in a context on a diagram, we write the indicated objects explicitly and indicate the context by "dots" as in the diagram to the right below. [9] For example, when we need to indicate only $A$ and $c$ on the left hand diagram, we could write it as shown on the right.





**Definition A.1** **Axiom**, **unification**, and **deletion** of GDS are defined as follows.

**Axiom:**

A1: For any circles $A$ and $B$, any minimal diagram where $A \bowtie B$ holds is an axiom.

A2: Any EUL-diagram which consists only of points is an axiom.

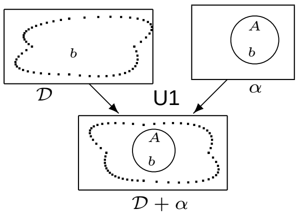
**Unification:** We denote by $\mathcal{D} + \alpha$ the unified diagram of $\mathcal{D}$ with a minimal diagram $\alpha$. $\mathcal{D} + \alpha$ is defined when $\mathcal{D}$ and $\alpha$ share one or two objects. We distinguish the following two cases: (I) When $\mathcal{D}$ and $\alpha$ share one object, they may be unified to $\mathcal{D} + \alpha$ by rules U1–U8 according to the shared object and the relation holding on $\alpha$. Each rule of (I) has a constraint for determinacy. (II) When $\mathcal{D}$ and $\alpha$ share two circles, if the relation which holds on $\alpha$ also holds on $\mathcal{D}$, $\mathcal{D} + \alpha$ is $\mathcal{D}$ itself; otherwise, they may be unified to $\mathcal{D} + \alpha$ by rules U9 or U10 according to the relation holding on $\alpha$. Each rule of (II) has a constraint for consistency. Moreover, there is another unification rule called the Point Insertion-rule (III).

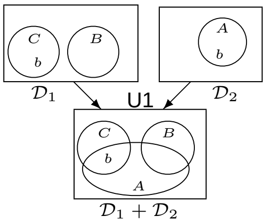
**(I)** The case $\mathcal{D}$ and $\alpha$ share one object:

U1: If $b \sqsubset A$ holds on $\alpha$ and $pt(\mathcal{D}) = \{b\}$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set $\mathsf{rel}(\mathcal{D} + \alpha)$ of relations holding on it is the following:

$$\mathsf{rel}(\mathcal{D}) \cup \{b \sqsubset A\} \cup \{A \bowtie X \mid X \in cr(\mathcal{D})\}$$

U1 is applied as follows:



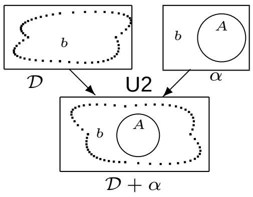


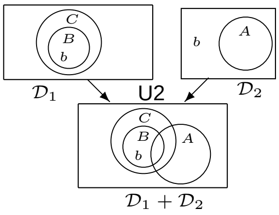
[9] Note that the dots notation is used only for abbreviation of a given diagram. For a formal treatment of such "backgrounds" in a diagram, see, for example, Meyer [11].

U2: If $b \vdash\!\dashv A$ holds on $\alpha$ and $pt(\mathcal{D}) = \{b\}$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set $\mathsf{rel}(\mathcal{D} + \alpha)$ of relations holding on it is the following:

$$\mathsf{rel}(\mathcal{D}) \cup \{b \vdash\!\dashv A\} \cup \{A \bowtie X \mid X \in cr(\mathcal{D})\}$$

U2 is applied as follows:

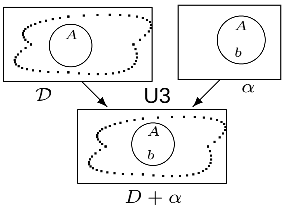


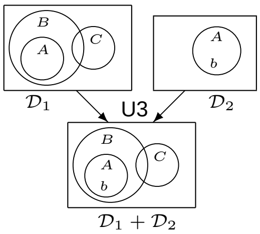


U3: If $b \sqsubset A$ holds on $\alpha$ and $A \in cr(\mathcal{D})$, and if $A \sqsubset X$ or $A \vdash\!\dashv X$ holds for all circle $X$ in $\mathcal{D}$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\mathsf{rel}(\mathcal{D}) \cup \{b \sqsubset X \mid A \sqsubset X \in \mathsf{rel}(\mathcal{D})\} \cup \{b \vdash\!\dashv X \mid A \vdash\!\dashv X \in \mathsf{rel}(\mathcal{D})\}$$
$$\cup \{b \vdash\!\dashv x \mid x \in pt(\mathcal{D})\}$$

U3 is applied as follows:

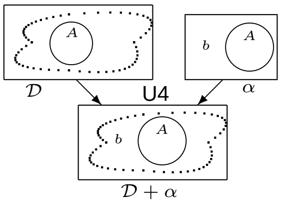


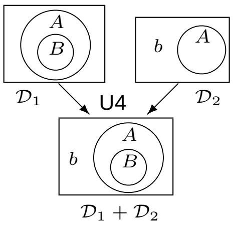


U4: If $b \vdash\!\dashv A$ holds on $\alpha$ and $A \in cr(\mathcal{D})$, and if $X \sqsubset A$ holds for all circle $X$ in $\mathcal{D}$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\mathsf{rel}(\mathcal{D}) \cup \{b \vdash\!\dashv X \mid X \sqsubset A \in \mathsf{rel}(\mathcal{D})\} \cup \{b \vdash\!\dashv x \mid x \in pt(\mathcal{D})\}$$

U4 is applied as follows:

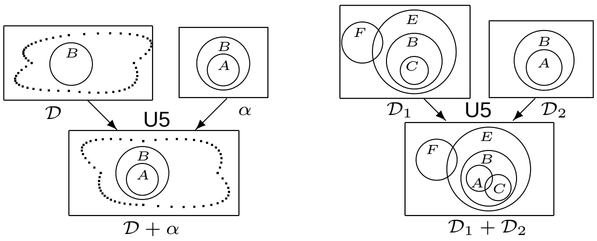




U5: If $A \sqsubset B$ holds on $\alpha$ and $B \in cr(\mathcal{D})$, and if $x \vdash\!\dashv B$ holds for all $x \in pt(\mathcal{D})$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\begin{aligned}\mathsf{rel}(\mathcal{D}) &\cup \{A \sqsubset X \mid B \sqsubset X \in \mathsf{rel}(\mathcal{D})\}\\ &\cup \{A \bowtie X \mid X \sqsubset B \text{ or } X \bowtie B \in \mathsf{rel}(\mathcal{D})\}\\ &\cup \{A \vdash\!\dashv X \mid X \vdash\!\dashv B \in \mathsf{rel}(\mathcal{D})\} \cup \{x \vdash\!\dashv A \mid x \in pt(\mathcal{D})\}\end{aligned}$$

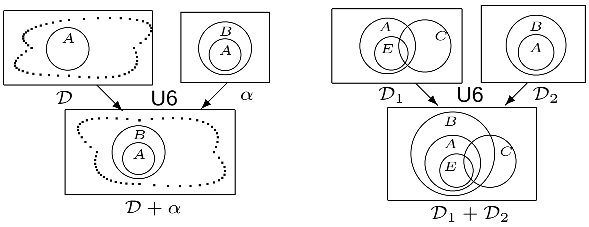
U5 is applied as follows:



U6: If $A \sqsubset B$ holds on $\alpha$ and $A \in cr(\mathcal{D})$, and if $x \sqsubset A$ holds for all $x \in pt(\mathcal{D})$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\begin{aligned}\mathsf{rel}(\mathcal{D}) &\cup \{X \sqsubset B \mid X \sqsubset A \in \mathsf{rel}(\mathcal{D})\} \cup \{x \sqsubset B \mid x \in pt(\mathcal{D})\}\\ &\cup \{X \bowtie B \mid A \sqsubset X \text{ or } A \vdash\!\dashv X \text{ or } A \bowtie X \in \mathsf{rel}(\mathcal{D})\}\end{aligned}$$

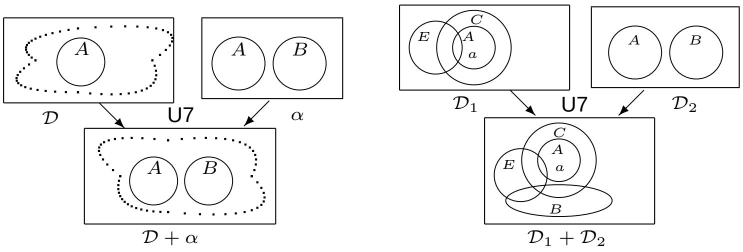
U6 is applied as follows:



U7: If $A \vdash\!\dashv B$ holds on $\alpha$ and $A \in cr(\mathcal{D})$, and if $x \sqsubset A$ holds for all $x \in pt(\mathcal{D})$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\begin{aligned}\mathsf{rel}(\mathcal{D}) &\cup \{X \vdash\!\dashv B \mid X \sqsubset A \in \mathsf{rel}(\mathcal{D})\} \cup \{x \vdash\!\dashv B \mid x \in pt(\mathcal{D})\}\\ &\cup \{X \bowtie B \mid A \sqsubset X \text{ or } A \vdash\!\dashv X \text{ or } A \bowtie X \in \mathsf{rel}(\mathcal{D})\}\end{aligned}$$

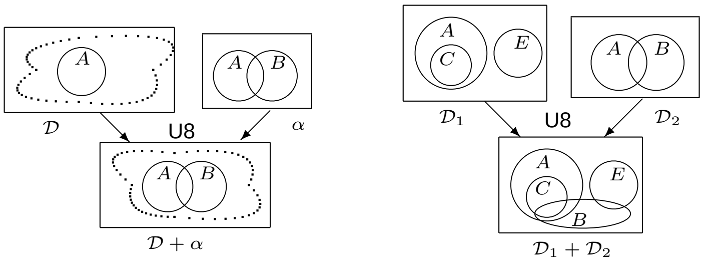
U7 is applied as follows:



U8: If $A \bowtie B$ holds on $\alpha$ and $A \in cr(\mathcal{D})$, and if $pt(\mathcal{D}) = \emptyset$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\mathsf{rel}(\mathcal{D}) \cup \{X \bowtie B \mid X \in cr(\mathcal{D})\}$$

U8 is applied as follows:

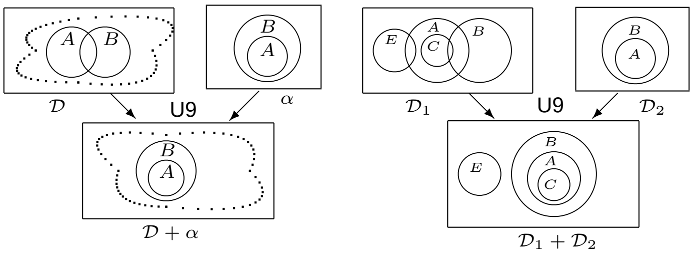


**(II)** When $\mathcal{D}$ and $\alpha$ share two circles, they may be unified to $\mathcal{D} + \alpha$ by the following U9 and U10 rules.

U9: If $A \sqsubset B$ holds on $\alpha$ and $A \bowtie B$ holds on $\mathcal{D}$, and if there is no object $s$ such that $s \sqsubset A$ and $s \mathrel{\vdash\!\dashv} B$ hold on $\mathcal{D}$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\big(\mathsf{rel}(\mathcal{D}) \setminus \{Y \bowtie X \mid Y \sqsubset A \text{ and } B \sqsubset X \in \mathsf{rel}(\mathcal{D})\} \setminus \{X \bowtie Y \mid X \sqsubset A \text{ and } Y \mathrel{\vdash\!\dashv} B \in \mathsf{rel}(\mathcal{D})\}\big)$$
$$\cup \{Y \sqsubset X \mid Y \sqsubset A \text{ and } B \sqsubset X \in \mathsf{rel}(\mathcal{D})\} \cup \{X \mathrel{\vdash\!\dashv} Y \mid X \sqsubset A \text{ and } Y \mathrel{\vdash\!\dashv} B \in \mathsf{rel}(\mathcal{D})\}$$

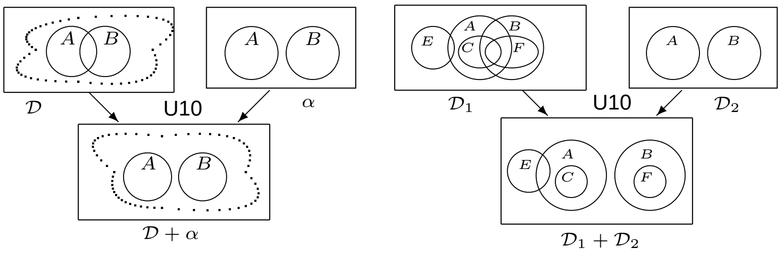
U9 is applied as follows:



U10: If $A \mathrel{\vdash\!\dashv} B$ holds on $\alpha$ and $A \bowtie B$ holds on $\mathcal{D}$, and if there is no object $s$ such that $s \sqsubset A$ and $s \sqsubset B$ hold on $\mathcal{D}$, then $\mathcal{D}$ and $\alpha$ may be unified to a diagram $\mathcal{D} + \alpha$ such that the set of relations $\mathsf{rel}(\mathcal{D} + \alpha)$ is the following:

$$\big(\mathsf{rel}(\mathcal{D}) \setminus \{X \bowtie Y \mid X \sqsubset A \text{ and } Y \sqsubset B \in \mathsf{rel}(\mathcal{D})\}\big) \cup \{X \mathrel{\vdash\!\dashv} Y \mid X \sqsubset A \text{ and } Y \sqsubset B \in \mathsf{rel}(\mathcal{D})\}$$

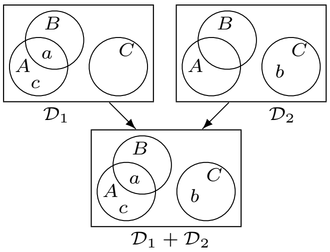
U10 is applied as follows:



**(III)** Point Insertion: If, for any circles $X, Y$ and for any $\Box \in \{\sqsubset, \sqsupset, \vdash\!\dashv, \bowtie\}$, $X \Box Y \in$ $\mathsf{rel}(\mathcal{D}_1)$ *iff* $X \Box Y \in \mathsf{rel}(\mathcal{D}_2)$ holds, and if $pt(\mathcal{D}_2)$ is a singleton $\{b\}$ such that $b \notin pt(\mathcal{D}_1)$, then $\mathcal{D}_1$ and $\mathcal{D}_2$ may be unified to a diagram $\mathcal{D}_1 + \mathcal{D}_2$ such that the set of relations $\mathsf{rel}(\mathcal{D}_1 + \mathcal{D}_2)$ is the following:

$$\mathsf{rel}(\mathcal{D}_1) \cup \mathsf{rel}(\mathcal{D}_2) \cup \{b \vdash\!\dashv x \mid x \in pt(\mathcal{D}_1)\}$$

Point Insertion is applied as follows:



**Deletion:** When $t$ is an object of $\mathcal{D}$, $t$ may be deleted from $\mathcal{D}$ to obtain a diagram $\mathcal{D} - t$ under the constraint that $\mathcal{D} - t$ has at least one objects.

The notion of *diagrammatic proofs (or, d-proofs)* is defined inductively as tree structures consisting of unification and deletion steps. The provability relation between EUL-diagrams is defined as usual. We denote by $\boldsymbol{\mathcal{D}}$ a sequence of diagrams $\mathcal{D}_1, \ldots, \mathcal{D}_n$.

**Theorem A.2 (Soundness and completeness of** GDS **[13])** *Let* $\boldsymbol{\mathcal{D}}, \mathcal{E}$ *be* EUL-*diagrams, and let* $\boldsymbol{\mathcal{D}}$ *have a model.* $\mathcal{E}$ *is a semantically valid consequence of* $\boldsymbol{\mathcal{D}}$ *(*$\boldsymbol{\mathcal{D}} \models \mathcal{E}$*), if, and only if, there is a d-proof of* $\mathcal{E}$ *from* $\boldsymbol{\mathcal{D}}$ *(*$\boldsymbol{\mathcal{D}} \vdash \mathcal{E}$*) in* GDS.

# Transforming Constraint Diagrams

Jim Burton* Gem Stapleton†

Ali Hamie‡
Visual Modelling Group
University of Brighton, Brighton, UK

**Abstract**

Constraint diagrams were proposed by Kent for the purposes of formal software specification in a visual manner. They have recently been formalized and generalized, making them more expressive. This paper presents a collection of transformations that can be applied to the so-called unitary $\alpha$ fragment of constraint diagrams. The transformations can be used to define inference rules in a more succinct manner than in earlier systems. We establish that the transformations are sufficient to transform any given unitary $\alpha$-diagram into any other unitary $\alpha$-diagram. Therefore, they are sufficient for formalizing any inference rules between such diagrams.

## 1 Introduction

Visual languages play an important role in the design and implementation of software. For example, the Unified Modelling Language (UML) [20] is now an industry standard visual notation designed specifically for use by software engineers and is used throughout the software development process, from capturing domain requirements through to implementation. Under some circumstances (such as in a safety critical environment; see, for example, [19]) it is desirable, perhaps even essential, to produce formal models of software. In part, such application areas serve to motivate the need for the precise specification of the UML at both a syntactic and semantic level; the pUML group was set up with this goal in mind [18].

Part of the creation of a formal model is likely to involve specifying constraints such as system invariants and operation contracts which, within the UML, is achieved by using the Object Constraint Language (OCL) [22]. The OCL is the only purely textual part of the UML and, therefore, does not fit with the UML's diagrammatic theme. Building on the formal diagrammatic reasoning systems of Shin [13], Hammer [1] and

* j.burton@brighton.ac.uk

† g.e.stapleton@brighton.ac.uk

‡ a.a.hamie@brighton.ac.uk

others, Kent introduced constraint diagrams [11] which are designed to complement the visual components of the UML and to specify constraints like the (symbolic) OCL. Constraint diagrams can also be used independently of the UML.

In figure 1 there is a constraint diagram which expresses an invariant that we might wish to place on a video rental store system: there is a member that can only borrow films that are in the collections of the stores which they have joined. The semantics of constraint diagrams will be explained more fully later, but the blob acts as an existential quantifier, the arrows allow us to make statements about binary relations and the closed curves represent sets (or classes).

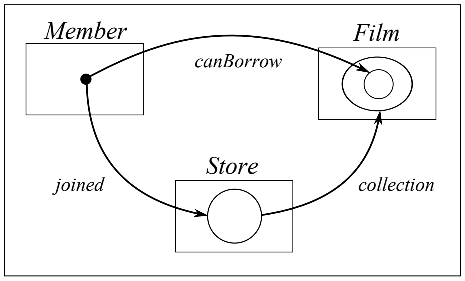


Figure 1: A constraint diagram.

At first glance, constraint diagrams appear intuitive and, perhaps, unambiguous, but it was not until a formalization of their semantics was attempted that a range of ambiguities was noticed [8]. Indeed, only when a formalization was eventually obtained [5] did the complexity of interpreting these diagrams become apparent. Whilst in many examples constraint diagrams are “well-matched to meaning” [9] there are also many situations where their intuitiveness breaks down. This led to the development of generalized constraint diagrams [14]. Both of these constraint diagram notations share a common fragment, which is considered in this paper. For this fragment, we set up a transformation system which forms the basis of a reasoning system for both constraint diagrams and generalized constraint diagrams.

There are various ways in which reasoning will need to be performed when using formal methods. First, there is reasoning about the model; for example, when one wishes to show that the model is consistent or that the post-condition of one operation implies the precondition of another. Secondly, a programmer will need to use some informal reasoning to determine an appropriate implementation that conforms to the specification. Thirdly, at a later stage, one might also wish to formally prove that the implementation does indeed conform to the model. Formal reasoning has been investigated for constraint diagrams [4, 16] but as yet no inference rules have been defined for generalized constraint diagrams.

An aim of this paper is to define a transformation system for so-called unitary $\alpha$-diagrams which can be used to subsequently define inference rules for either constraint diagrams or generalized constraint diagrams. Section 2 provides a brief overview of unitary diagrams. We also present a formalization of the syntax of unitary diagrams in section 2. Our transformations are defined in section 3, focusing separately on those which remove syntax and those which add syntax. Finally, in section 4, we show how the transformations can be used as a basis for inference rules.

# 2 Unitary Diagrams

We follow a typical approach of formally defining the syntax at an abstract level [10]. In this way, we disregard the many aspects of drawn diagrams that are irrelevant to their semantic meaning, such as the shape and relative location of curves. To aid intuition, we include a informal presentation of the concrete syntax, since this is used to guide the work, but all formal aspects are conducted at the abstract level.

## 2.1 Concrete Syntax

The concrete syntax of a visual language defines, in our case, diagrams as drawn images. We proceed to sketch the concrete syntax of so-called unitary constraint diagrams. We make occasional reference to semantics to aid the readers' understanding. For the purposes of this paper, the semantics are not particularly important, which is why we do not include their precise formalization.

Unitary constraint diagrams consist of closed curves (some of which may be labelled) drawn in the plane and which represent sets. The spatial relationships between the curves makes assertions about the relationships between the represented sets. For example, the diagram in figure 1 contains six curves, three of which are labelled. The placement of one curve inside another makes a subset assertion, whilst non-overlapping curves make a disjointness assertion. So, $Member$ and $Film$ are disjoint, for example.

In the regions formed by the curves we can place graphs, whose nodes are either all dots or all asterisks; these graphs are called existential spiders and universal spiders respectively. Existential spiders represent the existence of an element. In figure 1, there is one existential spider that has exactly one node placed inside $Member$. For simplicity of presentation, we will assume there are no universal spiders, although the transformations we define can easily be extended to cope with their inclusion. Similarly, we also assume that the existential spiders are placed in single zones; this constraint to single zones gives what are called $\alpha$-diagrams [16]. The curves in a diagram subdivide the plane into minimal regions: such a region is a connected component of the plane less the images of the curves. In figure 1, there are seven minimal regions. Of particular importance is the notion of a *zone* in a diagram, $d$. A zone is a set of minimal regions that can be described as being inside some (possibly no) curves but outside the rest of the curves in $d$. Semantically, a zone represents the set which is the intersection of the sets represented by the curves it is inside less the union of the sets represented by the curves that it is outside. In figure 1, every minimal region is also a zone and there are no zones that are not also minimal regions. However, this need not be the case: sometimes, zones consist of more than one minimal region and such zones are said to be disconnected. Zones can be shaded. The use of shading places an upper bound on the cardinality of the represented sets: in a shaded zone, all of the elements must be represented by spiders.

Finally, arrows are used to make statements about binary relations: the set of elements (or element) represented by the arrow's source is related to precisely the set of elements represented by the arrow's target under the relation represented by the arrow's label. For example, in figure 1 the arrow labelled *joined* sourced on the existential spider, $e$, asserts that the set of elements to which $e$ is related under the relation *joined* is a subset of *Store.* In addition, if we restrict the domain of *collection* to *Store* then we obtain a subset of *Film* which includes all of the films that can be borrowed by $e$.

So far, we have described unitary diagrams which consist of curves, spiders placed in zones or sets of zones, shading, and arrows. Further examples of unitary diagrams can be seen throughout the paper; we discuss the syntax of $d_1$ when presenting the formalization below. We refer the reader to [5] for further examples and more precise details on the concrete syntax and the semantics of constraint diagrams, and to [14] for similar information on generalized constraint diagrams. For the purposes of this paper, it is the formal, abstract, syntax that is important and the next section includes those details necessary for our transformations to be defined. Unitary diagrams can be joined together using logical connectives, such as $\wedge$, to make compound diagrams; it is unitary diagrams for which we define transformations.

We have placed a restriction on spiders so that they can only have one node, meaning that they are placed inside single zones. This restriction yields $\alpha$-diagrams which form a fragment of (non-unitary) generalized constraint diagrams that is not reduced in expressive power: given any generalized constraint diagram there exists a semantically equivalent diagram that contains only spiders placed inside single zones. However, there are constraint diagrams that are not semantically equivalent to any $\alpha$-diagram, but only if they contain universal spiders. That is given a constraint diagram containing only existential spiders, one can reduce it to an $\alpha$-diagram,as in [16]. We note that excluding universal spiders does decrease the expressive power but, as stated above, our work easily adapts to the case where they are permitted.

## 2.2 Abstract Syntax

Our formal definition of the syntax of unitary diagrams adapts that in [14]. In the abstract syntax we identify labelled curves with their labels; curve labels are drawn from the set $\mathcal{LC}$. Further, at the abstract level, the unlabelled curves are formalized as elements of an arbitrary (but specified) set $\mathcal{UC}$. We consider the elements of $\mathcal{UC}$ to correspond directly to the unlabelled curves of drawn diagrams. In a drawn diagram, a zone can be described by the curves that contain it and the curves that do not contain it. We use this insight to formalize zones at an abstract level.

**Definition 2.1.** *A **zone** is a pair,* $(in, out)$ *where* $in \cap out = \emptyset$ *and* $in \cup out \subseteq \mathcal{LC} \cup \mathcal{UC}$.

The set of all zones is denoted $\mathcal{Z}$. To illustrate the concept, the shaded zone in figure 2 can be described by $z = (\{A\}, \{B, uc\})$ where $uc$ denotes the unlabelled curve. There are two spiders placed in this zone; we cannot formalize a spider by identifying it with the zone in which it is placed. However, this provides the basis of their formalization: a spider will essentially be defined as a number together with a zone. In our example, the two spiders are written as $s_1(z)$ and $s_2(z)$.

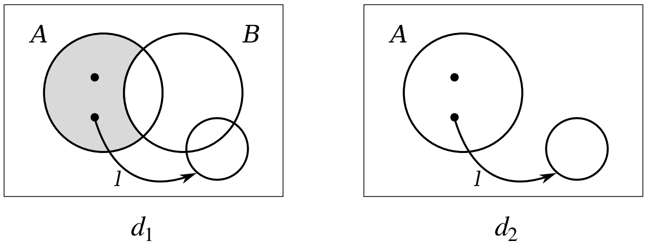


Figure 2: Formalizing the syntax.

**Definition 2.2.** *A **spider** is of the form $s_i(z)$ where $i$ is a natural number and $z$ is a zone. The **habitat** of $s_i(z)$ is $z$ and we say that $s_i(z)$ **inhabits** $z$.*

The set of all spiders is denoted $\mathcal{S}$. We now proceed to formalize arrows. To identify the arrows in a drawn diagram, it is sufficient to state their source and target, together with their label. For example, in figure 2, the arrow can be described by the triple $(l, s_1(z), uc)$ (recall, $uc$ is the unlabelled curve and $z = (\{A\}, \{B, uc\})$). We draw arrow labels from a fixed set $\mathcal{AL}$.

**Definition 2.3.** *An **arrow end** is either a curve drawn from $\mathcal{LC} \cup \mathcal{UC}$ or a spider drawn from $\mathcal{S}$. An **arrow** is an ordered triple $(l, s, t)$ where $l \in \mathcal{AL}$, and $s$ and $t$ are arrow ends called the **source** and **target** respectively.*

**Definition 2.4.** *A **unitary diagram** is a tuple, $d = (Z, Z^*, S, A)$, which satisfies the following:*

1. *$Z = Z(d)$ is a finite set of zones such that for each pair of zones $(in_1, out_1)$ and $(in_2, out_2)$ in $Z(d)$ we have $in_1 \cup out_1 = in_2 \cup out_2$. That is, the zones are all described using the same curves. We define $C(d) = in_1 \cup out_1$.*

2. *$Z^* = Z^*(d)$ is a set of shaded zones such that $Z^*(d) \subseteq Z(d)$. That is, all of the shaded zones are in the diagram.*

3. *$S = S(d)$ is a finite set of spiders such that for each spider $s_i(z) \in S(d)$, $z \in Z(d)$. That is, spiders are placed in zones of the diagram.*

4. *$A = A(d)$ is a set of arrows such that for each arrow $(l, s, t)$ in $A(d)$, $s$ and $t$ are in $S(d) \cup C(d)$. That is, arrows are sourced and targeted on components of the diagram.*

So, $d_1$ in figure 2 is formalized as the tuple $(Z, Z^*, S, A)$ where:

1. $Z$ is comprised of the following zones.

   - $(\{A\}, \{B, uc\})$,
   - $(\{A, B\}, \{uc\})$,
   - $(\{B\}, \{A, uc\})$,

- $(\{B, uc\}, \{A\})$,
- $(\{uc\}, \{A, B\})$,
- $(\emptyset, \{A, B, uc\})$

2. $Z^* = \{(\{A\}, \{B, uc\})\}$,
3. $S = \{s_1(\{A\}, \{B, uc\}), s_2(\{A\}, \{B, uc\})\}$, and
4. $A = \{(l, s_1(\{A\}, \{B, uc\}), uc)\}$.

Semantically, $d_1$ asserts that the set $A - B$ contains at least two elements, $x$ and $y$, through the use of the two existential spiders, the shading asserts that there are no more elements in that set (i.e. $|A - B| = 2$), and that $x$ is related to some set of elements, say $x.l$, under the relation $l$ such that $x.l \cap A = \emptyset$. The diagram $d_2$ makes a weaker statement, asserting that there are at least two elements in $A$, at least one of which is related to some set of elements, under $l$, that is disjoint from $A$. In fact, we can deduce $d_2$ from $d_1$ and, if we had a set of sound and (possibly) complete inference rules then we could prove that $d_2$ does indeed follow semantically from $d_1$.

# 3 Transformations

To facilitate elegant definitions of inference rules for unitary diagrams, we define *diagram transformations*, which are purely syntactic and represent the addition or removal of a piece of syntax. For example, we can remove the curve $B$ from $d_1$ in figure 2, transforming it into $d_2$; this remove curve transformation will be formalized below. The transformations defined will be applicable under specified syntactic conditions, which are not related to sound reasoning, but are intended to merely constrain the transformation to ensure the result of its application is a diagram. The benefit of making transformations which are purely syntactic and unrelated to reasoning is that this facilitates their use in a wide number of (reasoning) contexts.

## 3.1 Transformations that remove syntax

We start with the simplest transformation, that which removes an arrow.

**Transformation 1. Remove arrow**

We can transform a diagram by removing an arrow. In figure 3, the arrow, $a$, labelled $r$ is removed from $d_1$ to give $d_2$.
**Formal definition** Let $d_1$ be a unitary diagram such that there exists an arrow $a$ in $A(d_1)$. The diagram $d_2$ can be obtained from $d_1$ removing $a$ using the remove arrow transformation, denoted $d_1 \xrightarrow{-a} d_2$, where $d_2 = (Z(d_1), Z^*(d_1), S(d_1), A(d_1) - \{a\})$.

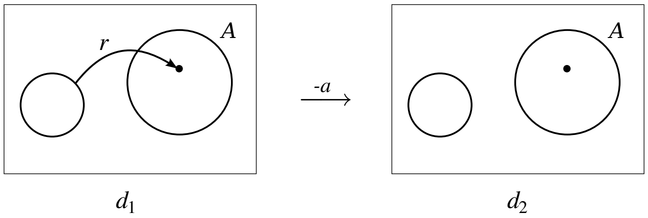


Figure 3: Transforming a diagram by removing an arrow.

**Transformation 2. Remove shading**

We can transform a diagram by removing the shading from a zone. In figure 4, the shading is removed from the zone $(\{B\},\{A\})$ in $d_1$ to give $d_2$.

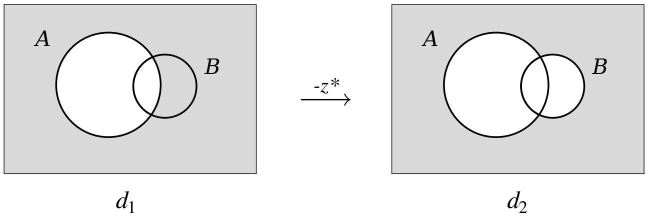


Figure 4: Transforming a diagram by removing shading from a zone.

**Formal definition** Let $d_1$ be a unitary diagram and let $z$ be a zone such that $z \in Z^*(d_1)$. The diagram $d_2$ can be obtained from $d_1$ using the remove shading transformation, denoted $d_1 \xrightarrow{-z^*} d_2$, where $d_2 = (Z(d_1), Z^*(d_1) - \{z\}, S(d_1), A(d_1))$.

**Transformation 3. Remove spider**

Our next transformation removes a spider from a unitary diagram. We need to provide a constraint (i.e. a precondition) on when this transformation can be applied in order to ensure that the result is a diagram. To formally define the remove spider transformation, we need to refer to the set of arrows sourced on, or targeting, a spider. Later, we also need to identify curves that are the source or target of an arrow. Here, we provide some notation that is convenient for identifying these sets.

**Definition 3.1.** *Let $d$ be a unitary diagram and let $s$ be a spider in $S(d)$. The set of arrows which are either sourced or targeted on $s$ in $d$, denoted $A(s,d)$, is*

$$A(s,d) = \{(l,\sigma,\tau) \in A(d) : \sigma = s \vee \tau = s\}.$$

*If an arrow $a$ is sourced or targeted on $s$ then we say $a$* ***touches*** *$s$. Similarly, we define the set of arrows which touch a curve $c$ in a diagram $d$, denoted $A(c,d)$:*

$$A(c,d) = \{(l,\sigma,\tau) \in A(d) : \sigma = c \vee \tau = c\}.$$

We can transform diagrams by removing a spider provided it is not touched by an arrow. In figure 5 $s$ is removed from $d_1$ to give $d_2$.

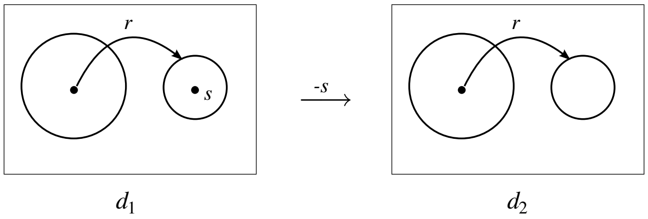


Figure 5: Transforming a diagram by removing a spider.

**Formal definition** Let $d_1$ be a unitary diagram such that there exists a spider $s \in S(d_1)$ which is not touched by any arrow, that is $A(s, d) = \emptyset$. The diagram $d_2$ can be obtained from $d_1$ by removing $s$ under the remove spider transformation, denoted $d_1 \xrightarrow{-s} d_2$, where $d_2 = (Z(d_1), Z^*(d_1), S(d_1) - \{s\}, A(d_1))$.

**Transformation 4. Remove zone**

We can transform a diagram by removing any zone which is not the habitat of any spider. In figure 6 the zone $(\{A, B\}, \emptyset)$ is removed from $d_1$ to give $d_2$.

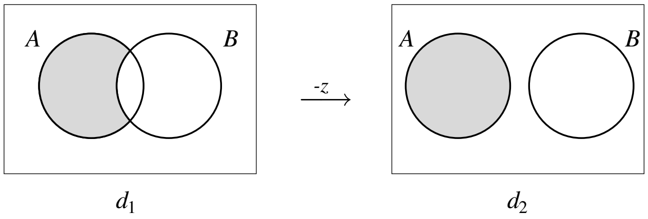


Figure 6: Transforming a diagram by removing a zone.

**Formal definition** Let $d_1$ be a unitary diagram such that there exists a zone $z$ in $Z(d_1)$ which is not the habitat of any spider. Then the diagram $d_2$ can be obtained from $d_1$ by removing $z$ under the remove zone transformation, denoted $d_1 \xrightarrow{-z} d_2$, where $d_2 = (Z(d_1) - \{z\}, Z^*(d_1) - \{z\}, S(d_1), A(d_1))$.

**Transformation 5. Remove curve**

We can remove a curve provided it is not touched by any arrow. In figure 7 the curve labelled $B$ is removed from $d_1$ to give $d_2$.

In diagram $d_1$ in figure 7, the region inside the curve labelled $A$ is partially shaded. We could choose to define the transformation which removes $B$ so that it removes this partial shading or leaves as shaded all zones which were shaded in the original. Actually, there are various choices for how to define a remove curve rule. We want to

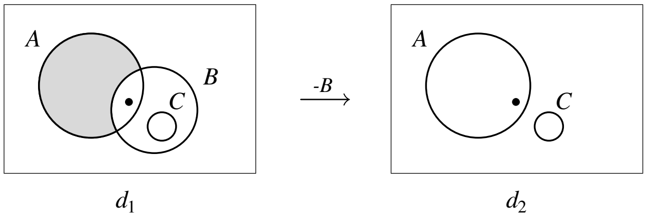


Figure 7: Transforming a diagram by removing a curve.

use the transformations as the basis for (useful) inference rules, so we have chosen to define this transformation in such a manner that it removes partial shading: we know that $B - A$ represents the empty set through the use of this shading and, when deleting $B$, we forget this information. In figure 8, the curve $B$ can be removed, but this time we retain the shading in $A$. We need our formalization to reflect when we must lose shading and when we can retain it; for this purpose we need to appeal to *missing zones*.

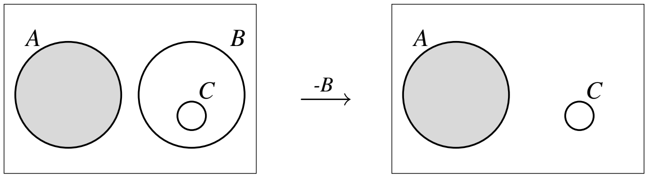


Figure 8: Accounting for missing zones.

**Definition 3.2.** *Let $d$ be a unitary diagram. A zone $z = (in, out)$ where $in \cup out = C(d)$ that is not in $Z(d)$ is said to be* ***missing*** *from $d$. The set of zones missing from $d$ is denoted $MZ(d)$, so $MZ(d) = \{(in, out) \in \mathcal{Z} : in \cup out = C(d)\} - Z(d)$.*

**Formal definition** Let $d_1$ be a unitary diagram and let $c$ be a curve such that $c \in C(d_1)$ and $A(c, d_1) = \emptyset$. The diagram $d_2$ can be obtained from $d_1$ by removing $c$ using the remove curve transformation, denoted $d_1 \xrightarrow{-c} d_2$, where $d_2$ has the following components.

1. $Z(d_2) = \{(in - \{c\}, out - \{c\}) : (in, out) \in Z(d_1)\}$,
2. $Z^*(d_2)$ is the union of the sets of zones $Z_{i,o}$, $Z_{i,m}$, and $Z_{m,o}$ where
   (a) $Z_{i,o}$ is formed by removing $c$ from the shaded zones of $d_1$ that were split by $c$ into two zones (one inside and one outside):

   $$Z_{i,o} = \{(in, out) : (in \cup \{c\}, out) \in Z^*(d_1) \wedge (in, out \cup \{c\}) \in Z^*(d_1)\},$$

   (b) $Z_{i,m}$ is formed by removing $c$ from the shaded zones of $d_1$ that $c$ was entirely within:

   $$Z_{i,m} = \{(in, out) : (in \cup \{c\}, out) \in Z^*(d_1) \wedge (in, out \cup \{c\}) \in MZ(d_1)\},$$

(c) $Z_{m,o}$ is formed by removing $c$ from the shaded zones of $d_1$ that $c$ was entirely outside:

$$Z_{m,o} = \{(in, out) : (in \cup \{c\}, out) \in MZ(d_1) \wedge (in, out \cup \{c\}) \in Z^*(d_1)\},$$

3. $S(d_2) = \{s_i(in - \{c\}, out - \{c\}) : s_i(in, out) \in S(d_1)\}$,
4. $A(d_2) = A(d_1)$.

## 3.2 Transformations that add syntax

The transformations that we define for adding syntax are counterparts of those which remove syntax. For the first two transformations no examples are given since they are very similar to their remove syntax counterparts.

**Transformation 6. Add arrow**

**Formal definition** Let $d_1$ be a unitary diagram and let $(l, s, t)$ be an arrow such that $s, t \in S(d) \cup C(d)$ and $(l, s, t) \notin A(d_1)$. The diagram $d_2$ can be obtained by adding $(l, s, t)$ to $d_1$ using the add arrow transformation, denoted $d_1 \xrightarrow{+a} d_2$, where $d_2 = (Z(d_1), Z^*(d_1), S(d_1), A(d_1) \cup \{(l, s, t)\})$.

**Transformation 7. Add spider**

**Formal definition** Let $d_1$ be a unitary diagram such that there exists a zone $z \in Z(d_1)$ and a spider $s_i(z) \notin S(d_1)$ where $z \in Z(d_1)$. The diagram $d_2$ can be obtained by adding $s$ to $d_1$ using the add spider transformation, denoted $d_1 \xrightarrow{+s} d_2$, where $d_2 = (Z(d_1), Z^*(d_1), S(d_1) \cup \{s\}, A(d_1))$.

**Transformation 8. Add shading**

We can transform a diagram by adding shading to a zone. In figure 9, shading is added to the zone $(\{A, C\}, \{B\})$ in $d_1$ to give $d_2$.

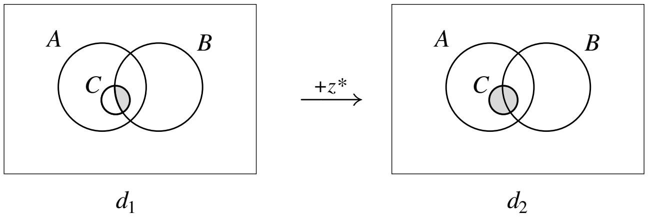


Figure 9: Transforming a diagram by adding shading to a zone.

**Formal definition** Let $d_1$ be a unitary diagram and $z \in Z(d_1) - Z^*(d_1)$. The diagram $d_2$ can be obtained from $d_1$ by adding shading to $z$ using the add shading transformation, denoted $d_1 \xrightarrow{+z^*} d_2$, where $d_2 = (Z(d_1), Z^*(d_1) \cup \{z\}, S(d_1), A(d_1))$.

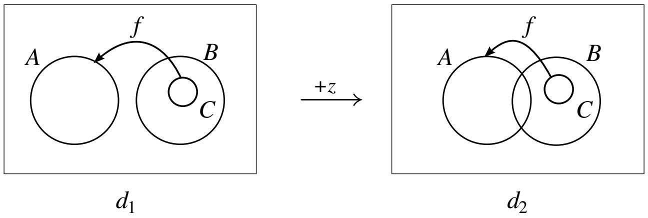


Figure 10: Transforming a diagram by adding a zone.

**Transformation 9. Add zone**

We can transform a diagram by adding a missing zone. Figure 10 shows the addition of the missing zone $(\{A, B\}, \{C\})$ to $d_1$ to give $d_2$.
**Formal definition** Let $d_1$ be a unitary diagram and $z$ be a zone such that $z \in MZ(d_1)$. The diagram $d_2$ can be obtained from $d_1$ by adding $z$ using the add zone transformation, denoted $d_1 \xrightarrow{+z} d_2$, where $d_2 = (Z(d_1) \cup \{z\}, Z^*(d_1), S(d_1), A(d_1))$.

**Transformation 10. Add curve**

There are a number of ways of adding a curve to a diagram: the new curve can be added in such a way that it is entirely outside of all existing curves, or is entirely contained within one other curve, and so on. The relationship between the new curve and the existing curve can be captured by appealing to its relationship with the existing zones: the existing zones are either completely inside the new curve, completely outside the new curve, or split by the new curve. Thus, we parametrise the transformation of adding a curve to a diagram $d_1$ using two subsets of zones which we call $Z_{in}$ and $Z_{out}$, where $Z_{in} \cup Z_{out} = Z(d_1)$; those zones which will fall inside the new curve are in $Z_{in}$, those outside in $Z_{out}$ and those that will be split are in $Z_{in} \cap Z_{out}$. The case of adding a curve which splits every zone in $d_1$, for instance, is that of choosing $Z_{in} = Z_{out} = Z(d_1)$. Figure 11 shows an example of adding a curve with $Z_{in} = \{(\emptyset, \{A, B\})\}$ and $Z_{out} = Z(d_1) - Z_{in}$.

In addition, each spider can be inside or outside the new curve. Thus, we also supply a two-way partition of the spider set, $S_{in}$ and $S_{out}$, allowing us to specify the habitats of the spiders after the curve addition. We must place constraints on the choice of $S_{in}$ and $S_{out}$ to ensure consistency with the manner in which the curve is added. For instance, we cannot place a spider $s_i(z)$ in the set $S_{in}$ if $z \in Z_{out} - Z_{in}$, since the 'new' habitat of the spider will not be present in the diagram after the curve addition. Consequently, we only have a choice about whether a spider, $s_i(z)$, is in $S_{in}$ or $S_{out}$ if $z \in Z_{in} \cap Z_{out}$. Note that this is a syntactic constraint and not related to soundness. An appropriate choice of $Z_{in}$, $Z_{out}$, $S_{in}$ and $S_{out}$ allows the user to add curves in any of the possible ways.

Recall that the set $\mathcal{LC} \cup \mathcal{UC}$ is the abstract set that corresponds to the labelled and unlabelled curves that can appear in any diagram at the concrete syntax level. The set $C(d)$, for any unitary diagram $d$, is a subset of $\mathcal{LC} \cup \mathcal{UC}$.

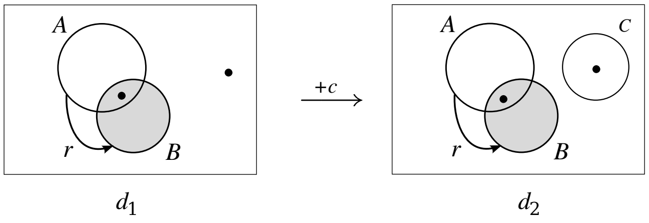


Figure 11: Transforming a diagram by adding a curve.

**Formal definition** Let $d_1$ be a unitary diagram and let $c$ be a curve that is not in $d_1$, that is $c \in (\mathcal{LC} \cup \mathcal{UC}) - C(d)$. Let $Z_{in}$ and $Z_{out}$ be subsets of $Z(d_1)$ such that $Z_{in} \cup Z_{out} = Z(d_1)$. Let $S_{in}$ and $S_{out}$ be a two-way partition of $S(d_1)$ such that

1. for all $s_i(z)$ in $S_{in}$, $z \in Z_{in}$ and
2. for all $s_i(z)$ in $S_{out}$, $z \in Z_{out}$.

The diagram $d_2$ can be obtained by adding the curve $c$ to $d_1$ using the add curve transformation, denoted $d_1 \xrightarrow{+P} d_2$, where $P = (c, Z_{in}, Z_{out}, S_{in}, S_{out})$ and $d_2$ has the following components:

1. $Z(d_2) = Z_{in+c} \cup Z_{out+c}$ where
   (a) $Z_{in+c}$ is formed by adding $c$ to the zones of $d_1$ that we wish to contain $c$ in $d_2$: $Z_{in+c} = \{(in \cup \{c\}, out) : (in, out) \in Z_{in}\}$,
   (b) $Z_{out+c}$ is formed by adding $c$ to the zones of $d_1$ that we wish to exclude $c$ in $d_2$: $Z_{out+c} = \{(in, out \cup \{c\}) : (in, out) \in Z_{out}\}$.
2. $Z^*(d_2) = Z^*_{in+c} \cup Z^*_{out+c}$ where
   (a) $Z^*_{in+c} = \{(in \cup \{c\}, out) : (in, out) \in Z_{in} \cap Z^*(d_1)\}$,
   (b) $Z^*_{out+c} = \{(in, out \cup \{c\}) : (in, out) \in Z_{out} \cap Z^*(d_1)\}$.
3. $S(d_2) = S_{in+c} \cup S_{out+c}$ where
   (a) $S_{in+c} = \{s_i(in \cup \{c\}, out) : s_i(in, out) \in S_{in}\}$,
   (b) $S_{out+c} = \{s_i(in, out \cup \{c\}) : s_i(in, out) \in S_{out}\}$.
4. $A(d_2) = A(d_1)$.

## 3.3 Completeness of the Transformations

The set of transformations defined above is complete because we can use them to transform any unitary diagram into any other unitary diagram, although the resulting system is (intentionally) not sound. The completeness of the transformation system means that these transformations are sufficient for describing a set of sound and complete inference rules for the unitary $\alpha$-diagram fragment of both constraint diagrams and generalized constraint diagrams.

**Theorem 1.** *Let $d_1$ and $d_n$ be unitary diagrams. Then there exists a sequence of diagrams, $(d_1, d_2..., d_n)$ such that for each $i$, where $1 < i \leq n$, the diagram $d_i$ can be obtained from $d_{i-1}$ by the application of one of the above transformations. In other words, the given set of transformations is complete.*

*Sketch.* The transformations which remove syntax can be used repeatedly to transform $d_1$, regardless of its content, into the diagram $(\emptyset, \emptyset, \emptyset, \emptyset)$. The transformations which add syntax can then be used to build $d_2$. □

Although there will often be faster ways to transform one diagram into another, we can rely on the 'brute force' method of removing all diagrammatic elements from $d_1$ to produce the empty diagram then adding the elements of $d_2$. This relies on choosing the right order in which to apply the transformations, depending on their pre-conditions of syntactic well-formedness; for instance, before using remove spider (transformation 3) to remove a spider $s$ which is the source of an arrow $a$ in $d_1$, we must first use remove arrow (transformation 1) to remove $a$ from $d_1$.

# 4 Using Transformations to Define Inference Rules

We are able to use the transformations to define inference rules in a variety of ways. The motivation for using transformations in this way is by analogy with software engineering. Functional and modular abstraction leads to systems with less code duplication and which are easier to understand and maintain. In a similar way, the shorter inference rule definitions that result from abstracting syntactic details into transformations are easier to state, reason about and check for errors. We are able to compose transformations in the definition of rules which make several changes to a diagram in a similar way to composing referentially transparent functions in a functional programming language. To use the transformations when defining inference rules, we may need to place further conditions on when the transformation can be applied to ensure soundness. In the case of the erasure of a spider, such a condition would be that the habitat is not shaded.

We have defined a set of sound inference rules for the unitary $\alpha$ fragment of generalized constraint diagrams, and present three of their definitions below as examples. Work on establishing a complete set for this fragment is ongoing. Four transformations can immediately be used as sound inference rules: remove arrow, remove shading, remove curve and add zone. The other transformations need not result in a semantic consequence of the diagram to which they are applied. Some transformations are used by several rules; for instance, (at least) five of inference rules that we have so far defined use the add arrow transformation. An *add shaded zone* inference rule makes use of two transformations. We have also begun work on defining inference rules whose application results in a compound diagram, and expect a significant proportion of these inference rules to use more than one of the transformations defined here, particularly as compared to the unitary fragment, because the compound inference rules tend to be more complex.

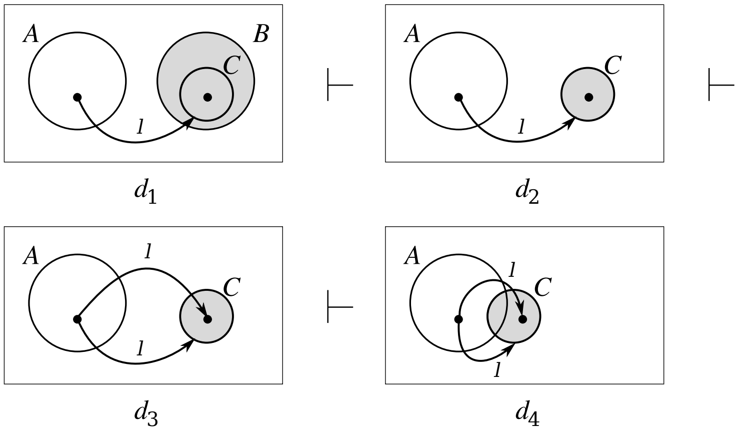


Figure 12: Using transformations to define inference rules.

To illustrate how we use the transformations to define sound inference rules, we consider an example. Figure 12 shows a proof of $d_4$ from $d_1$. First, we apply the remove curve transformation to $d_1$ giving $d_2$. We note that the remove curve transformation can be used directly as an inference rule; that is, applying the remove curve transformation always results in a semantic consequence of the diagram to which the transformation is applied. Next, we apply an add arrow rule to give $d_3$. Unlike the remove curve transformation, we cannot add arrows in arbitrary ways and obtain a semantic consequence. The information provided by the new arrow must be deducible from the information already present in the diagram and, as stated, we have defined a number of rules which add arrows. The rule used to obtain $d_3$ from $d_2$ is called *Add arrow: contour to spider*. Before defining the rule, we define the *empty* curves of a diagram, or those within which every zone is shaded.

**Definition 4.1.** *Let $d$ be a unitary diagram. Define the* ***empty curves*** *of $d$, denoted $EC(d)$, as follows.*

$$EC(d) = \{c \in C(d) : \forall (in, out) \in Z(d)\ c \in in \Rightarrow (in, out) \in Z^*(d)\}.$$

**Definition 4.2.** ***Add arrow: contour to spider.*** *Let $d_1$ be a unitary diagram such that:*

1. *there is an arrow $(l, s, t)$ in $A(d_1)$ such that $t \in EC(d_1)$, and*

2. *there is a spider $\sigma \in S(d_1)$ such that $S(t, d_1) = \{\sigma\}$.*

*Let $d_2$ be the diagram obtained by adding the arrow $(l, s, \sigma)$ to $d_1$ using the add arrow transformation. Then we may replace $d_1$ with $d_2$.*

As an example of the reuse of transformations, we include a second inference rule which adds an arrow to a diagram. Informally, diagram $d_1$ in figure 13 tells us that the sum of the images of the relation $r$ when restricted to the elements of $A$ is the empty set. This information is provided by the arrow labelled $r$ which is sourced on $A$ and targets the shaded and unlabelled curve. It follows that we can add an arrow with the same source and label which targets any other empty curve without changing the meaning of the diagram. The inference rule *Add arrow: empty set* allows us to do this and is used in $d_2$ to add an arrow with the same source and label as the arrow in $d_1$ but which targets $B$.

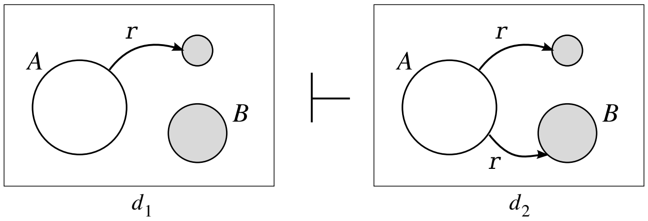


Figure 13: An application of the inference rule *Add arrow: empty set*.

**Definition 4.3.** ***Add arrow: empty set.*** *Let $d_1$ be a unitary diagram which satisfies:*

1. *there is an arrow $(l, s, t)$ in $A(d_1)$ where $t \in EC(d_1)$,*
2. *there is a curve $t_1 \in EC(d_1)$ where $(l, s, t_1) \notin A(d_1)$.*

*Let $d_2$ be the diagram obtained by adding the arrow $(l, s, t_1)$ to $d_1$ using the add arrow transformation. Then we may replace $d_1$ with $d_2$.*

Returning to figure 12, we obtain $d_4$ from $d_3$ using a combination of two transformations: add zone and add shading. The diagram $d_3$ asserts that $A \cap C = \emptyset$ since $A$ and $C$ do not overlap, but we can assert this disjointness using a shaded zone, namely $(\{A, C\}, \emptyset)$, justifying that $d_4$ is a semantic consequence of $d_3$. This intuition is formalized in the following inference rule.

**Definition 4.4.** ***Add shaded zone.*** *Let $d_1$ be a unitary diagram and $z$ be a zone such that $z \in MZ(d_1)$. Let $d_2$ be the diagram obtained by applying the add zone transformation to add $z$ to $d_1$ obtaining $d_1'$, then applying the add shading transformation to shade $z$ in $d_1'$ obtaining $d_2$. Then we can replace $d_1$ by $d_2$.*

# 5 Extending to Compound Diagrams

The defined transformations focus on unitary diagrams. In both constraint diagrams and generalized constraint diagrams, logical operators are used to form so-called compound diagrams, albeit in rather different manners in the two notations. Our transformations can also be used in the context of compound constraint diagrams and generalized constraint diagrams. For instance, figure 14 shows two compound constraint diagrams, the second of which is a consequence of the first. In the first diagram we have two unitary diagrams joined by a $\wedge$ connective and, in one of those diagrams, $d_1$, an arrow labelled $r$ is shown. The diagram $d_2$ includes the source and target of the arrow in $d_1$ but not the arrow itself. Applying the add arrow transformation to add this arrow to $d_2$ is a sound inference step and results in the second compound diagram.

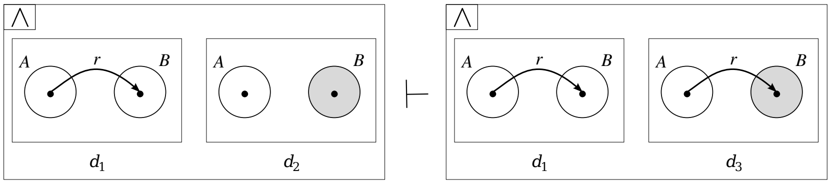


Figure 14: A constraint diagram and the add arrow transformation.

Since non-unitary inference rules sometimes have more complex postconditions, they are more likely to make use of several syntactic transformations than are unitary inference rules. An illustration of this is is given by the rule *excluded middle for zones*, adapted from [16]. This rule states that an unshaded zone either contains exactly the number of spiders depicted, or the zone must contain at least one more spider than depicted. Therefore, the conclusion of the rule is a disjunction of two diagrams, where the add spider transformation had been applied to the first, and the add shading transformation applied to the second. An example is shown in figure 15. The diagrams $d_2$ and $d_3$ are copies of $d_1$ except that transformations have been used to add a spider to zone $\{A, B\}$ in the first and to shade the same zone in the second.

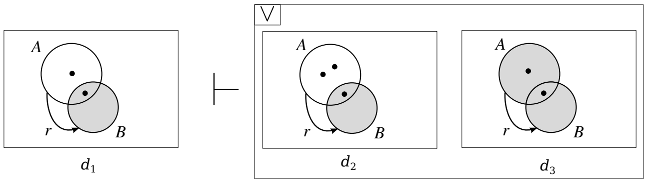


Figure 15: Illustrating *Excluded middle for zones*.

**Definition 5.1.** ***Excluded middle for zones.*** *Let $d_0$ be a unitary constraint diagram, let $z$ be a non-shaded zone in $d_0$, or $z \in Z(d_0) - Z^*(d_0)$, and let $s$ be a spider not in $S(d_0)$. Let $d_1$ be the diagram obtained by using the add spider transformation to add $s$ to $z$ in $d_0$, and let $d_2$ be the diagram obtained by using the add shading transformation to add shading to $z$ in $d_0$. Then $d_0$ can be replaced by $d_1 \vee d_2$.*

# 6 Conclusion

In this paper we have presented a series of transformations that can be applied to unitary diagrams (either constraint diagrams or their generalized form) and established that they are complete. They provide a basis for defining inference rules, as illustrated in section 4, for both constraint diagrams and generalized constraint diagrams. Defining transformations with the right level of generality allows us to use them flexibly in the definition of inference rules. In the future, we plan to use these transformations to build a sound and, ideally, complete reasoning system for generalized constraint diagrams.

We are in the process of creating a proof assistant for reasoning with the abstract syntax of constraint diagrams [2] which is intended to form a flexible basis for graphical tools. The implementation of the tool uses the notion of modular, purely syntactic transformations combined with preconditions to form inference rules in a manner very close to the abstract definitions of the rules. This might in fact be called a "traditional" software engineering solution to the problem of creating such a tool. This close symmetry and the fact that the tool uses a dependently typed language to create types which correspond directly to abstract diagrams, transformations and inference rules, helps when establishing the correctness of the tool.

Also in the context of tool support, significant research has been directed towards the automated generation and layout of Euler diagrams, which form the bases of constraint diagrams, including [3, 6, 15, 21]. Moreover, other work has focused on how to add spiders to the drawn Euler diagrams [12]. Thus, much work has been conducted on how to automatically draw concrete diagrams from their abstract syntax. This diagram drawing functionality provides a basis for making interactive proof assistants and theorem provers accessible to a range of users, not just those familiar, and confident with using, the abstract syntax. Already, fully automated theorem provers have been developed for Euler diagrams [17] and spider diagrams [7]. Thus, whilst significant further work is required to develop tool support for constraint diagrams, there is already a firm basis on which we can build.

**Acknowledgements.** Jim Burton thanks the UK EPSRC for support through a DTA studentship. Gem Stapleton acknowledges support from the UK EPSRC, grant number EP/E011160/1, for the Visualization with Euler diagrams project
(www.eulerdiagrams.com).

# References

[1] Barwise, J. and E. Hammer, *Diagrams and the concept of logical system*, in: G. Allwein and J. Barwise, editors, *Logical Reasoning with Diagrams*, Oxford University Press, 1996 .

[2] Burton, J., *Diagrams and intuitive formal specifications*, in: P. Bottoni, M. B. Rosson and M. Minas, editors, *Visual Languages and Human-Centric Computing*, IEEE (2008), pp. 262–263.

[3] Chow, S. and F. Ruskey, *Drawing area-proportional Venn and Euler diagrams*, in: *Proceedings of Graph Drawing 2003, Perugia, Italy*, LNCS **2912** (2003), pp. 466–477.

[4] Fish, A. and J. Flower, *Investigating reasoning with constraint diagrams*, in: *Visual Language and Formal Methods 2004*, ENTCS **127** (2005), pp. 53–69.

[5] Fish, A., J. Flower and J. Howse, *The semantics of augmented constraint diagrams*, Journal of Visual Languages and Computing **16** (2005), pp. 541–573.

[6] Flower, J. and J. Howse, *Generating Euler diagrams*, in: *Proceedings of 2nd International Conference on the Theory and Application of Diagrams* (2002), pp. 61–75.

[7] Flower, J., J. Masthoff and G. Stapleton, *Generating readable proofs: A heuristic approach to theorem proving with spider diagrams*, in: *Proceedings of 3rd International Conference on the Theory and Application of Diagrams*, LNAI **2980** (2004), pp. 166–181.

[8] Gil, J., J. Howse and S. Kent, *Towards a formalization of constraint diagrams*, in: *Proc IEEE Symposia on Human-Centric Computing (HCC '01), Stresa, Italy* (2001), pp. 72–79.

[9] Gurr, C. and K. Tourlas, *Towards the principled design of software engineering diagrams*, in: *Proceedings of 22nd International Conference on Software Engineering* (2000), pp. 509–518.

[10] Howse, J., F. Molina, S. J. Shin and J. Taylor, *On diagram tokens and types*, in: *Proceedings of 2nd International Conference on the Theory and Application of Diagrams* (2002), pp. 146–160.

[11] Kent, S., *Constraint diagrams: Visualizing invariants in object oriented modelling*, in: *Proceedings of OOPSLA97* (1997), pp. 327–341.

[12] Mutton, P., P. Rodgers and J. Flower, *Drawing graphs in Euler diagrams*, in: *Proceedings of 3rd International Conference on the Theory and Application of Diagrams*, LNAI **2980**, pp. 66–81.

[13] Shin, S. J., "The Logical Status of Diagrams," CUP, 1994.

[14] Stapleton, G. and A. Delaney, *Evaluating and generalizing constraint diagrams*, Journal of Visual Languages and Computing **19** (2008), pp. 499–521.

[15] Stapleton, G., J. Howse, P. Rodgers and L. Zhang, *Generating euler diagrams from existing layouts*, in: *Layout of (Software) Engineering Diagrams*, Electronic Communications of the EASST (2008), pp. 16–31.

[16] Stapleton, G., J. Howse and J. Taylor, *A decidable constraint diagram reasoning system*, Journal of Logic and Computation **15** (2005), pp. 975–1008.

[17] Stapleton, G., J. Masthoff, J. Flower, A. Fish and J. Southern, *Automated theorem proving in Euler diagrams systems*, Journal of Automated Reasoning **39** (2007), pp. 431–470.

[18] The Precise UML Group, *Untitled*, http://www.cs.york.ac.uk/puml/index.html (1997).

[19] UK Ministry of Defence, *The procurement of saftey critical software in defence equipment* (1993).

[20] Unified Modelling Language, *Untitled*, http://www.uml.org/ (2006).

[21] Verroust, A. and M. L. Viaud, *Ensuring the drawability of Euler diagrams for up to eight sets*, in: *Proceedings of 3rd International Conference on the Theory and Application of Diagrams*, LNAI **2980** (2004), pp. 128–141.

[22] Warmer, J. and A. Kleppe, “The Object Constraint Language: Precise Modeling with UML,” Addison-Wesley, 1998.

# A Cognitive Exploration of the “Non-Visual” Nature of Geometric Proofs

Peter W. Coppin[*†] Stephen A. Hockema[‡]
Faculty of Information
University of Toronto

**Abstract**

*Why are Geometric Proofs (Usually) “Non-Visual”?* We asked this question as a way to explore the similarities and differences between diagrams and text (visual thinking versus language thinking). Traditional text-based proofs are considered (by many to be) more rigorous than diagrams alone. In this paper we focus on human perceptual-cognitive characteristics that may encourage textual modes for proofs because of the ergonomic affordances of text relative to diagrams. We suggest that visual-spatial perception of physical objects, where an object is perceived with greater acuity through foveal vision rather than peripheral vision, is similar to attention navigating a conceptual visual-spatial structure. We suggest that attention has foveal-like and peripheral-like characteristics and that textual modes appeal to what we refer to here as foveal-focal attention, an extension of prior work in focused attention.

**Keywords** attention, visual thinking, proof, logic, geometry

## 1 Introduction

*Why are geometric proofs usually “non-visual”?* We asked this question as a way to explore the similarities and differences between diagrams and text (visual thinking versus language thinking [19]). We felt that the examples provided by text-based geometric proofs might be a microcosm for notation use in broader contexts, such as education, a field similarly traditionally dominated by text relative to visual-spatial information [13]. We believe that ongoing research to increase an understanding of the cognitive dimensions of visual-spatial notations relative to text could increase abilities to conceptualize, comprehend, and communicate ideas in education, public policy, and beyond by introducing principled approaches for using ergonomically appropriate notations relative to an intended communication or comprehension purpose (cf. [5, 8]).

[*] E-mail: petercoppin@gmail.com

[†] This material is based upon work supported under a National Science Foundation Graduate Research Fellowship and a University of Toronto Fellowship (during different time periods).

[‡] E-mail: steve.hockema@utoronto.ca

Despite a *prima facie* case that the subject matter of geometry and its underlying theories seems to be about spatial forms and relationships, geometric proofs are most often formally represented to people as text-based descriptions of geometric properties that demonstrate how a geometric relationship is necessarily true as a series of logical relationships. As Tennant [17] described: *"[The diagram] is only an heuristic to prompt certain trains of inference; . . . it is dispensable as a proof-theoretic device; indeed, . . . it has no proper place in a proof as such. For the proof is a syntactic object consisting only of sentences arranged in a finite and inspectable array."* (as quoted in [4])

For example, if block A is under block B and block C is above block B, then logic tells us that block A is below block C. Alternatively, we can easily induce that block A is below block C by observing a diagram, yet the logical text-based proof is considered more rigorous than a diagram [17]. Indeed, for generations, Euclid's *Elements* was considered to be flawed because of its reliance on diagrams. As Mumma [11] described: *"for some of Euclid's steps, the logical form of the preceding sentences is not enough to ground the steps. One must consult the diagram to understand what justifies it."* For this reason it is commonly felt that Euclid *"failed in his efforts to produce (an) exact, full explicit mathematical proof"* [11]. (We will show an example of this below in section 3.1.)

Barwise and Etchemedy [4] began to question the assumption that diagrams were less rigorous than (non-diagrammatic) proofs, bolstering their perspective by using evidence from cognitive psychology [9] that showed how maps enable problem solving more effectively in certain situations. Nonetheless, text-based "language thinking" and algebraic notations remain the dominant mode relative to diagrams throughout mathematics [6]. We asked: *"could there be ergonomic properties afforded by text relative to diagrams that encourages textual modes for proofs?"*

An inversion of our question might be phrased as follows: *what do the representational modes of sequential symbolic proofs relative to diagrams reflect about human cognition?* Though others [1, 6, 12] have explored related questions and have described the dominance of text over visual-spatial representations through historical explanations, we focus our attention on human perceptual-cognitive characteristics that may encourage proofs (and similar materials) to evolve towards text-based modes relative to visual-spatial modes because of the ergonomic affordances of text relative to diagrams.[1]

## 1.1 Related Work

Relative to well developed studies of language in linguistics and related fields, studies of formal visual representations are sporadic and fall across less connected fields [1, 4, 13]. Very little prior work was found that addressed our specific question (especially from a perceptual-cognitive perspective), however, some work with results that can be adapted to explore our question follow:

[1] We also narrow our focus to the presentation of proofs in their final form, as opposed to also considering the discovery and construction process of proving geometric theorems. While the two are obvious related, we believe there is enough to say about the former here that can stand on its own without considering the latter.

Coming from an information processing (cognitive psychology) perspective, Larkin and Simon [10] sought to explore the differences between information as diagrams versus sentences, concluding that sentences embody the characteristic of being indexed on a list, with each element "adjacent" only to the next element in the list. In contrast, diagrams are indexed by location on a plane, many elements may share the same location, and each element may be adjacent to any number of other elements; in this way, Larkin and Simon propose that diagrams may be more useful than sentences for solving certain kinds of problems because they can support more efficient computational processes. (Larkin and Simon include human neurological processes when they use the word "computational".) They also noted that this efficiency depends on the design of the diagram and the ability of the user to interpret the diagram.

Approaching the issue as mathematicians and logicians, Barwise and Etchemendy [4] begin their work by noting that in the field of mathematics and logic, diagrams are not considered valid parts of a proof, and are present only as a heuristic aid (Barwise and Etchemendy [4] citing Tennant [17]). A major purpose of [4] is to overturn this thinking, bolstering their case by citing cognitive psychologist Kosslyn [9], who used maps to justify visual presentations as valid problem solving tools, also making the point that sentences or visual representations offer advantages or disadvantages based on the purpose of the task at hand.

Barwise and Etchemendy conclude (like Larkin and Simon in [10]) by describing advantages offered by diagrams that are not offered by sentences, and by doing so offer a suite of differences between sentences and diagrams that extend [10]. For example, with diagrams relationships are often implicit, whereas with sentences, even the most trivial consequences must be inferred explicitly (as demonstrated in the introductory example). Additionally, they point out that a picture or diagram can support "countless facts" (by this they mean that a plurality of sentences can be constructed from a diagram).

More recently, Mumma approached the question of why geoemtric proofs are language-based by examining the so-called "flaws" in Euclid's proofs where he made use of diagrams and then seeking to provide a rigorous diagramatic foundation for these proofs [12]. In so doing, Mumma proposed three interrelated factors why Euclid's reliance on diagrams in his proofs is regarded as non-rigorous These were what he referred to as:

- the generality problem – proofs are meant to be more general than the *particular* instance in a diagram but how should we generalize from a particular diagram to a more general case?

- the modern mathematical understanding of continuity – diagrams may lead to simplistic and invalid assumptions about the continuity of lines, e.g. with respect to the existence of intersection points

- the modern axiomatic method, which requires that *all* axioms and deductive steps be explicitly specified, and raises suspicion about the assumptions embedded in diagrams.

Thus, like Barwise and Etchemendy, Mumma locates the answer in the subject matter and norms of the field and then attempts to argue that the above three problems can be overcome in a more carefully specified hybrid system. We will argue below that there is also another component to the answer based on basic facts about human cognition.

Along these lines, Shimojima and Katigiri [16] sought to support Barwise and Etchemendy [4] by gaining empirical evidence to show how diagrams reduce inferential load by drawing on newer discoveries in cognitive science such as Ballard et al.'s [2] theory of "deictic indices." Deictic indices are mental pointers to particular objects in external space. The theory suggests that through an attentional mechanism, people can maintain a small pool of such indices at once and can easily direct focal (mental) attention or gaze to any of these indices. In Ballard's et al.'s [2] words *"pointing movements are used to bind objects in the world to cognitive programs."*

Shimojima and Katigiri [16] describe how these indices can be used to keep track of (mental) "non-physical drawings" they may construct while doing inferences about diagrams. They suggest that reasoners can then navigate their attention through these "non-physical" (mental) drawings during reasoning tasks and that these non-physical drawings are assisted by real drawings (diagrams) and that this assistance reduces "inferential load" during reasoning tasks.

Thus, although none of these works specifically address our question about why proofs are "non-visual," we gain important insight regarding the differences between text and diagrams that can guide our inquiry. The thread running through each prior work listed above is that text/language/prose guides attention in ways that are different from visual-spatial representations. The next section will attempt to demonstrate this more explicitly.

## 1.2 Integrating and Extrapolating from Prior Research: Language Appears to Guide Attention through Visual-Spatial Structures

From this loose collection of interrelated work, we can extrapolate some general principles regarding the cognitive dimensions of illustrations relative to text.

Larkin and Simon suggested in [10] that a cognitive dimension of sentences is their list-like structure, in that each item on the list is only adjacent to the item before or after it on the list. In contrast, items in a diagram are adjacent to many items on a list. This view is synergistic with Barwise and Etchemendy, who suggested that a picture or diagram can support "countless facts" (by this they mean that a plurality of sentences can be constructed from a diagram). In other words, many sentences could be created by linking together elements in a diagram into a sentence (list-like structure).

In this way, we extrapolate that a list-like structure (as suggested by Larkin and Simon) can be inferred or induced from a diagram. Each sentence inferred from a diagram is like a path that guides attention through visual-spatial relationships in a diagram. This extrapolation is demonstrated by Shimojima and Katigiri's eye tracking study in [16] that showed how reasoners mentally guide their attention through a "non-physical drawing." They suggest that actual drawings thus support non-physical (mental) drawings, thus reducing inferential load. To summarize, it appears that sentences guide attentional paths through both physical and non-physical (mental) visual-spatial structures. Further, it appears that Shimojima and Katigiri demonstrated that rational language/propositional logic guides attention and motor movements (through eye fixations) through non-physical visual-spatial representations.

### 1.3 Why does “Language Thinking feel More Precise than Visual Thinking”?

At this point in the paper, we are almost ready to suggest a contributing reason to the answer of why “geometric proofs are text-based.” We have suggested that sentences guide attention through visual-spatial structures. However, a question remains: *Why does navigating attention through a visual spatial structure guided by rational language (propositional logic) feel more “rigorous” than experiencing it as a diagram?* As described above, in [12] Mumma proposed a 3-part answer as to why the diagrams are considered less rigorous by the field in general. Here we narrow the question to focus on cognitive dimensions of individual mathematicians.

We propose that detailed scrutiny of visual-spatial structures (and perhaps concepts in general) requires what we will refer to as “higher resolution” foveal-like attention, even if those visual-spatial structures are conceptual (not directly sensed). We suggest that visual-spatial perception in the physical world, where an object is perceived with greater acuity through foveal vision rather than peripheral vision, is similar to attention navigating a conceptual visual-spatial structure. *We suggest that attention has foveal-like and peripheral-like characteristics.* Linkages to traditional (and synergistic) ideas of attention (e.g., [18]) will be described later in this paper.

To explain how navigating non-physical (mental) drawings may have dimensions that mimic visual-spatial perception of the external world through foveal and peripheral vision, we can turn to Barsalou’s [3] theory of perceptual symbol systems where concepts are based on inherently modal neural patterns rooted in direct sensory experience. As Barsalou [3] describes:

> During perceptual experience, association areas in the brain capture bottom-up patterns of activation in sensory-motor areas.
>
> In a top-down manner, association areas partially reactivate sensory-motor areas to implement perceptual symbols.
>
> The storage and reactivation of perceptual symbols operates at the level of perceptual components not at the level of holistic perceptual experiences [3].

In other words, an experience of a geometric visual-spatial structure is inherently modal in that, if experienced through the eyes and visual cortex (for example), the memory of that experience would reflect the experiential mode (i.e. visual versus auditory experience). This means that concepts that emerge from the neurological patterns created from sensory (modal) experience reflect the characteristics of the experiential mode. This means that an experience of a visual-spatial structure that emerges as a concept uses much of the same neurological machinery used to perceive (experience) the visual-spatial structure.

This relationship between foveal attention and perceptual symbols is the basis for our theory detailed in the next section.

# 2 Theory

*Why are geometric proofs (usually) "non-visual?"* We propose that perceptual architectures associated with foveal (sharper, center view, but narrower field of view [FOV]), and peripheral (outside of the center view, less sharp, but a wider FOV) vision found in the human eye, in V1, and the rest of the visual cortex extend into the "deepest levels" of human cognition and are reflected both in conceptual structures and the architecture of attention that "probes" those conceptual structures. In this paper, foveal attention is analogous to (and parallels) foveal vision. Likewise, "peripheral perceptual-cognitive attention" (shortened to peripheral attention [PA] for the rest of this paper) is analogous to peripheral vision.

We suggest that foveal attention may by synonymous with "focused attention" as proposed by Treisman [18], who suggested *"attention must be directed serially to each stimulus in a display whenever conjunctions of more than one separable feature are needed to characterize or distinguish the possible objects presented."* By separable feature, she means primitives such as basic shapes, objects, and colors prior to integration into a conceptual "whole." Furthermore, Treisman uses a metaphor that easily maps to our description of foveal attention:

> Visual attention, like a spotlight or zoom lens, can be used over a small area with high resolution or spread over a wider area with some loss of detail. (Treisman [18] citing Eriksen and Hoffman [7])

We can extend the analogy in the present context to suggest that attention can either be narrowed to focus on a single feature, when we need to see what other features are present and form an object, or distributed over a whole group of items which share a relevant feature [18].

This *"narrowing of the spotlight"* is synonymous with what we mean by foveal attention. Relative to peripheral attention, we suggest that foveal attention is more precise and can detect more detail, paralleling Treisman's spotlight/zoom lens metaphor. Similar to how the eye must explore areas broader than the narrow FOV of foveal vision via a sequence of saccades, we suggest that focal attention must also sequentially walk through mental visual-spatial structures. However, perhaps differently from low-level saccades, we suggest that *language and language-thinking guides* attention through such structures in order to build more precise holistic ideas.

By building on Barsalou's notion that concepts arise from neural patterns that are rooted in modal experiences, and our own speculation (extrapolating from Triesman) that foveal-focal attention may have a limited "FOV," several explanations for why proofs are usually text-based and propositional are proposed:

***Why proofs are often sequential:*** Like foveal vision that must saccade to different parts of a visual-spatial structure, we suggest that foveal-focal attention must also "saccade" to different parts of a conceptual visual-spatial structure due to foveal-focal attention's narrower FOV (rather than experiencing / attending to the whole structure at once ).[2]

[2] It should be emphasized that we are using the notion of saccade here metaphorically; sequential movement of attention to various parts of a structure will probably bear no resemblence to the way an eye actually

***Why a diagram usually cannot constitute a convincing "holistic" proof:*** The need for symbols to fall within the narrow FOV of foveal-focal attention means that diagrams, and the spatial relationships they embody, usually cannot be taken in (i.e. attended to) all at once. So they should instead be processed in a way that allows linkages between earlier perceptual memories and later percepts.

***Why text is effective for proofs:*** External symbolic representations such as text are designed such that each symbol can sequentially fall within the narrow FOV of foveal vision [15] and therefore, foveal attention and foveal-focal attention.

***Why propositional logic is used for proofs:*** Propositional structures in proofs may provide symbolic "short-cuts," serving as stand-ins for visual-spatial relationships that cannot all simultaneously be in the limited FOV of foveal attention. For example: The statement *"if C is below B"* references a perceptual symbol constructed from a previously considered image, and the statement *"if B is below A"* also references a perceptual symbol constructed from a previously considered image. The statement *"therefore C is below A"* references the two previous symbols in order to support construction of a new (mental) image that can serve as the basis for a new perceptual symbol (and that can be used in future propositional statements).

To summarize our theory; we suggest that a visual-spatial structure, such as a geometric structure (irregardless of whether it is presented as a diagrammatic representation) is often beyond the "FOV" of foveal-focal attention. The purpose of sequential symbolic representations such as text, organized as propositional statements, is to guide foveal-focal attention through a sequence of patterns in order to create perceptual symbols that are amenable to analytical neurological machinery.

Hence, in addition to being due to the norms of the field of mathematics, as well as many other social and mathematically technical reasons that have been proposed, we argue the answer to our initial question is *also* related to basic facts about how human cognitition works.

# 3 Thought Experiment

A thought experiment may help clarify the role of the less-diagrammatic notation style of propositional logic used in a geometric proof by imagining what a notation style would look like that was designed to guide narrow FOV foveal-focal attentional processes.

First, a notation style where symbols would fit the narrow FOV of foveal vision would seem to be appropriate, although it is perhaps not the only style that could work. For example, with a better understanding of processes linking perception and attention, we might be able to use more "bandwidth" in parallel by providing just the right cues to guide attention through a diagram (and hence, reduce or even avoid the need for "symbols" altogether). Yet text appears to be an example of a notation system naturally suited to the ergonomics of foveal vision [14].

saccades, such as to its ballistic nature for example. Further, the "sequence" implied by the word "sequential" here is not necessarily imply a *particular* ordering, especially not one that might correspond with the actual sequence of eye saccades. We assume that many cognitive and pragmatic factors play into determining in what order structures must be attended.

Second, geometric structures expressed through the notation system would need to be "serialized" as chunks/strings since more complex visual-spatial structures (i.e. diagrams associated with non-trivial geometric proofs) will presumably require a FOV that fall outside of the foveal attentional units of a notation system ergonomically designed for foveal vision [14].

Third, the notation style would need to deliver those serialized chunks/strings in ways that would direct foveal attention in specifically ordered trajectories and patterns to build a network of foveal-focal perceptual symbols in order to construct and mentally navigate a visual-spatial conceptual structure that falls outside of the "FOV" of foveal-focal attention.

This is because sequential patterns of foveal-focal "saccades" support the creation of perceptual symbols that can be referenced in later attentional "saccades." This would be a hierarchical structure of previously attended foveal-focal mental images (perceptual symbols) where latter parts of the conceptual structure reference previously attended foveal-focal mental images.

In other words, a notation system that was custom designed to guide foveal-focal attention through a visual-spatial conceptual structure would resemble the ergonomics of sequential symbolic (i.e. text based) proofs consisting of serialized sequential symbols (i.e. descriptions), and support the embedding of perceptual symbols, constructed from previously experienced foveal mental images, into other mental images and perceptual symbols (i.e. propositional logic).

## 3.1 Example

A more specific example may reveal the ergonomic characteristics and constraints described above. Proposition 35 from Book I of Euclid's Elements is a classic example that we suggest demonstrates the way that text appears to focus foveal-focal attention. It is also a useful example in that it is a hybrid proof, as will be described below, relying on both text and diagram.

Proposition 35 is that parallelograms that are on the same base and in the same parallels equal one another. Euclid's proof proceeds as follows:

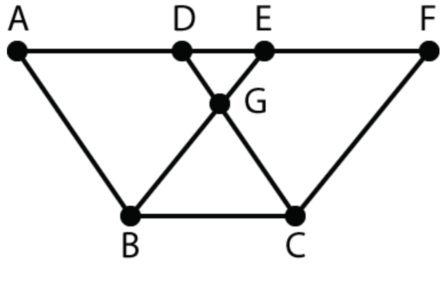


Figure 1: Diagram used in proof of Proposition 35

**Proof.**

(i) Let ABCD, EBCF be parallelograms on the same base BC and in the same parallels AF, BC.

(ii) Since ABCD is parallelogram, AD equals BC (Proposition 34). Similarly, EF equals BC.

(iii) Thus, AD equals EF. (Common Notion 1)

(iv) Equals added to equals are equal, so AE equals DF. (Common Notion 2)

(v) Again, since ABCD is a parallelogram, AB equals DC (Proposition 34) and angle EAB equals angle FDC (Proposition 29).

(vi) By side angle side congruence, triangle EAB equals triangle FDC (Proposition 4). Subtracting triangle EDG from both, we have that the trapezium ABGD equals the trapezium EGCF (Common Notion 3).

(vii) Adding triangle GBC to both, we have that ABCD equals EBCF (Common Notion 2);

□

Note how the text references aspects of the visual-spatial concept in "chunks," revealing the visual-spatial concepts serially, possibly ergonomically optimized for foveal-focal attentional processes. For example, the first line introduces the symbol "$ABCD$", which, in the presence of the figure, directs attention sequentially through the verticies A-B-C-D and then to the parallelogram as a whole, separable from the rest of the figure. However, the figure is not necessary for this step, for ABCD could also serve the role of a simple "word" (term) in the logical proof without actually *referring* to the geometric figure at all. Further, for everything specified early on in the proof—symbols and relationships—each line can be derived from the previous without making reference to the figure at all. Although the figure can still play a helpful illustrative role, it does not play a role in sanctioning particular inference steps. Indeed, as Mumma describes in [12], this is the case all the way up to step $iv$.[3]

For our purposes, the key point here is that in its linguistic form, devoid of the figure, the above proof is amenable to cognitive processes, and this is revealed in aspects of its design. The structure of the language guides the reader through the proof in "bite-sized" chunks – both in the the breakdown as to what constitutes an individual *step*, and the number of symbols involved in each step – that can be linked together and composed to gradually build up to the conclusion.

However, things get more complicated in steps $iv$ through $vii$, in part, because the author of the proof (Euclid via translation) seems to be making the assumption that we will be using the figure to provide interpretation for the text statements, and thus both allow the engagement of our natural perceptual-cognitive chunking abilities and relieve

[3] Step $iv$ contains a so-called "flaw" in that in order to determine what the "equals" are that have been added to AD and EF, one needs to realize that DE is a common segment shared by both AE and DF. The step relies on an understanding that DE has been "added" to both AD and EF and it is reflexively self-equal. However, while it is true, it was nowhere stated in steps $i$-$iii$ that DE is a shared segment, i.e., that D lies on segment AE and that E lies on segment DF. In the above proof, this is knowledge that must be gleaned from the figure.

some of our memory burdens by using the figure as external memory. The proof can be rewritten more "rigorously" to eliminate the need to refer to the figure and to rely on text alone, but this presumably will also involve a simplification of steps $iv$ through $vii$ by breaking them down into more, explicit steps that each demand less cognitive work for the reader.

But could we go the other way? Could we prove the same thing by relying even *more* on diagrams and using much less text? If so, what would it take? In the next subsections we will attempt to do so in order to illustrate our thoery as to how deductive proofs require the explicit sequential guidance of attention through symbols, and demonstarte the difficulties that arise when we attempt to employ diagrams for this purpose.

## 3.2 Towards a more Visual Version

In the last subsection, we saw an example of a geometry proof that, while not entirely avoiding reliance on a diagram, was primarily text-based. In light of our argument above (that this is partly because text is better suited to guide foveal-focal attention sequentially through the appropriate perceptual symbols), here we seek to illustrate this point by describing the results of our attempt at a visual proof of Proposition 35. How could this be achieved without the aid of text? Or in other words, how can one draw attention without using text?

The most straightforward approach here might be to translate each line of text from the proof into a diagram seeking to achieve the same thing. In so doing, one might seek to direct attention to different parts or aspects of the diagram using graphical techniques such as highlighting with color, luminance (value), shading, or adding arrows (to name a few techniques). So, for example, to translate the step that establishes that AD equals EF (step $iii$), we might highlight both of those seqments in red and add some sort of connection beteween the two to denote equivalence. Further, we would need to somehow add the *justification* for this, somehow refering to Common Notion 1.

After we attempted to do this in a systematic way, several things become apparent. First, the guidance of attention within a diagram seemed harder to control relative to text, and even somewhat arbitrary (somewhat like "black magic") to a practitioner primarily trained to express ideas through text. The guidance of attention with a diagram seems to be something that requires a solid scientific understanding of how diagrams are visually processed and/or considerable artistic sophistication and skill. Additionally, relative to our experience with text, it was harder to be precise – to the "right" degree of precision – in a diagram. For example, how can we call attention to just the vertices, say, of a parallelogram, without also calling attention to the parallelogram itself via gestalt principles of perception? Similarly, as Mumma described in [12], it was harder to make general points when dealing with specific diagrams, and there seemed to be the potential to be misled by superfluous details in the diagrams.

So, a straightforward translation of a text-based proof into a visual one seemed to presents several difficulties. Yet there may be other approaches to visual proof that are not biased by the texual starting point, perhaps natively taking advantage of unique affordances of diagrams. We speculate about this in the next subsection.

An open question is how text and diagrams can best be used *together* to complement each other's strengths in a a hybrid approach. As noted, Euclid's proof was not entirely text based; the diagram was required for it to go through. It is instructive to consider the role the diagram played in this case. In our theory, a primary function of the diagram is to provide the basis for perceptual symbols to which the text can then refer and "navigate", i.e. guide attention through. We unpack this more in the next paragraph.

As an example of how the diagram works together with the text, consider step $vi$ of the proof:

- By side angle side congruence, triangle EAB equals triangle FDC (Proposition 4). Subtracting triangle EDG from both, we have that the trapezium ABGD equals the trapezium EGCF (Common Notion 3).

First, note that the diagram serves to confirm the applicability of Common Notion 3 in a way similar to how the diagram was used to justify step $iv$. While the text here could be augmented to ostensibly stand on its own, it is still compelling to look at the trapezia ABGD and EGCF in the figure to at least *confirm that the text was properly understood*. (A reader can "double-check" that they did the subtraction in the right way by checking that the trapezia they obtained were indeed describable as ABGD and EGCF, that is, by unpacking the symbols—e.g., ABGD to A-B-G-D—to confirm the correct vertices were involved.) Next, note that the first sentence not only establishes an equivalence relationship, but also serves to focus attention on the two relevant triangles, EAB and FDC, as quasi-independent parts of the diagram, which then makes them available to mental manipulation (and thus, subsequent use in the text). The operation of subtracting triangle EDG from both EAB and FDC requires manipulating it as a "non-rivalrous" (reusable) symbolic unit. Further, as was previously discussed, [16] has provided evidence of how the diagram is used when mentally reasoning about such operations, with deictic indices being allocated to diagrammatic symbols from a limited pool that constrains the number of symbols to which we can simultaneouly attend. Thus, the mental subtraction requires manipulating holistic triangles as symbols – each as *one* thing – as opposed to a loosely coupled collections of their component vertices and edge segments which would quickly exhaust attentional resources. The diagram helps to reify these component parts into unitary symbolic wholes thanks to gestalt principles of perceptual grouping.

So one of the primary roles of the diagram seems to be to serve as the basis for perceptual symbols that can play a role in *compositional* manipulations, analogous to how words are composed via grammar in language. That is, the diagram, when it is being used effectively in a hybrid text-diagram proof system, still seems to be guiding attention in ways similar to text (sybolically and sequentially). Yet, diagrammatic symbols may afford more than textual symbols in that they can be unpacked and "inspected" when necesary, as the trapezia were when verifying the subtraction. This warrants more research into the affordances of diagrams relative to text in the context of proofs, as opposed to in more general contexts.

## 3.3 Other Hybrid Systems

In the last subection, we started by considering how we might constuct a visual proof from a primarily text-based one. But starting with text might bias us away from a proof strategy that might better take advantage of the affordances of diagrams. What if we were to start from scratch?

Ware points out in [19] that one of the more effective techniques used in HCI and Information Visualization to draw attention is motion. This implies that animations might be a useful tool in geometric proofs. However, since traditional (paper oriented) document formats do not ordinarily support dynamic animations, for the purposes of this paper a comic strip-like story-board will depict such a possible interaction by way of example. Figure 2 shows a possible story-board (or comic strip), corresponding to a potential animation sequence, for the proof of Proposition 35. This is just a tentative sketch to demonstrate the idea; it is has not been refined nor tested with mathematicians, nor has research been done on the proper balance between textual and non-textual elements in such a medium.

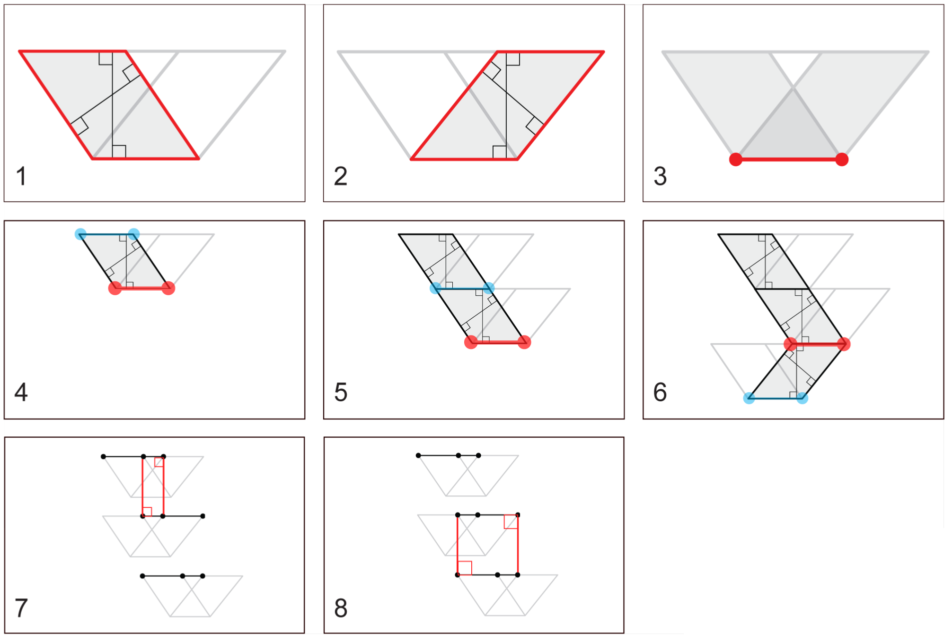


Figure 2: A possible "storyboard" (or "comics") hybrid proof

Storyboards (or comics), in their own right, also present a potentially useful hybrid approach for directing attention during a visual proof. In our theory, the storyboard has a useful property for proofs: the boxes help guide attention, much like the proof steps and symbols in a text-based proof. Further, the domain of comics provides many techniques to further refine the guidance of attention (such as cutaway shots) and help with the problems described above that arise when trying to use diagrams in proofs. For example, there are standard comics techniques to “zoom” in and out, both temporally and spatially, across panels, which may be useful in addressing the problems with precision.

One final point of interest here that emerged while creating this example is that this form of reasoning “felt” more inductive than deductive. Our research seeks to increase the understanding of how different media types might be better suited for diferent sorts of inference. Understanding ergonomic factors that enable a principled approach to attention guidance is an area of active future research for us.

# 4 Conclusion and Implications

We feel that demonstrating how propositional logic guides foveal-focal attention through non-physical drawings has implications far beyond notation techniques for geometric proofs; the issues surrounding the text-based nature of geometric proofs are a microcosm for issues facing other materials, such as textbooks, because many educational concepts that are also visual-spatial in nature use illustrations in a supporting role [13]. We feel that understanding how notation styles direct attention can enable the creation of materials suited for different purposes and for different kinds of learners.

For example, the effectiveness of the Barwise and Etchemedy vision of using visual materials for logical problem solving would be a function of how well the materials support the guiding of foveal-focal attention in specific patterns. Indeed, many of their ideas were rooted in classroom multimedia experiences geared for the teaching of logic. Using today’s technology of the Web and multimedia (i.e. rollovers, mouse events, etc.), the sequential symbolic characteristics of comics, and beyond, many options exist to direct foveal-focal attention using visual materials in ways that may serve the attentional purposes provided by text-based proofs and text heavy materials. We feel that such techniques could be extended for other information presentations as well, finding uses in education, public policy, business, engineering, and beyond.

However, we suggest here that prose may direct attention in ways that may not be possible through visual-spatial notations such as illustrations and diagrams. Because text and other sequential symbolic “language” systems inherently require learning, they may by their very nature “bypass” or “overcome” low level pattern detection neurological machinery in order to trigger “top down” processes that amplify attentional neurological machinery in order to focus attention in specific ways on visual-spatial diagrams or conceptual structures. A more intuitive notation system that relied more on “natural,” “hard-wired.” or “gestalt-like” abilities might strengthen bottom-up patterns instead, at the expense of the intended abstract reasoning encouraged by such materials. Instilling neural patterns that overcome lower level neurological machinery may be the very nature of some aspects of education. At the same time, visual thinking has been proposed as a key part of “transformational thinking,” and many great discoveries occurred through insights with a strong visual-spatial component [13] and by individuals who do not learn well through textual modes [15].

Thus, many open questions remain about the ergonomic properties of textual and diagrammatic modes.

Importantly, a better, more rigorous understanding of the explicit and implicit guidance of attention is a necessary step, and a direction for our current and future research.

## References

[1] Arnheim, R., "Visual thinking," University of California Press, 2004.

[2] Ballard, D. H., M. M. Hayhoe, P. K. Pook and R. P. N. Rao, *Deictic codes for the embodiment of cognition*, BEHAVIORAL AND BRAIN SCIENCES **20** (1995), pp. 723—767.
URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.3813`

[3] Barsalou, L. W., *Perceptions of perceptual symbols*, Behavioral and Brain Sciences **22** (1999), pp. 637–660.
URL `http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=31859`

[4] Barwise, J. and J. Etchemendy, "Visual information and valid reasoning," Mathematical Association of America, 1991 pp. 9–24.
URL `http://portal.acm.org/citation.cfm?id=115667`

[5] Blackwell, A. F., *Ten years of cognitive dimensions in visual languages and computing: Guest editor's introduction to special issue*, Journal of Visual Languages & Computing **17** (2006), pp. 285–287.
URL `http://www.sciencedirect.com/science/article/B6WMM-4K5JBS2-1/2/d76e6d2adf8fbe31d1dcecf8ddfb7fca`

[6] Brown, J. R., "Philosophy of mathematics," Routledge, 1999, 215 pp.

[7] Eriksen, C. W., *Temporal and spatial characteristics of selective encoding*, Perception and Psychophysics (1980).

[8] Green, T. and M. Petre, *Usability analysis of visual programming environments: A 'cognitive dimensions' framework*, Journal of Visual Languages and Computing **7** (1996), pp. 131–174.
URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-0030167097&partnerID=40`

[9] Kosslyn, S. M., "Image and mind," Harvard Univ Pr, 1980.

[10] Larkin, J. H. and H. A. Simon, *Why a diagram is (sometimes) worth ten thousand words*, Cognitive science **11** (1987), pp. 65–100.

[11] Mumma, J., "Ensuring Generality in Euclids Diagrammatic Arguments," 2008 pp. 222–235.
URL `http://dx.doi.org/10.1007/978-3-540-87730-1_21`

[12] Mumma, J., *Proofs, pictures, and Euclid*, Synthese (2009, in press). URL `http://www.contrib.andrew.cmu.edu/~jmumma/list.html`

[13] Ramadas, J., *Visual and spatial modes in science learning*, International Journal of Science Education **31** (2009), pp. 301–318.

[14] Rayner, K. and J. H. Bertera, *Reading without a fovea*, Science **206** (1979), pp. 468–469, ArticleType: primary_article / Full publication date: Oct. 26, 1979 / Copyright  1979 American Association for the Advancement of Science. URL `http://www.jstor.org/stable/1749329`

[15] Schneps, M. H., L. T. Rose and K. W. Fischer, *Visual learning and the brain: Implications for dyslexia*, Mind, Brain, and Education **1** (2007), pp. 128–139. URL `http://dx.doi.org/10.1111/j.1751-228X.2007.00013.x`

[16] Shimojima, A. and Y. Katagiri, “An Eye-Tracking Study of Exploitations of Spatial Constraints in Diagrammatic Reasoning,” 2008 pp. 74–88. URL `http://dx.doi.org/10.1007/978-3-540-87730-1_10`

[17] Tennant, N., *The withering away of formal semantics?*, Mind & Language **1** (1986), pp. 302–318. URL `http://dx.doi.org/10.1111/j.1468-0017.1986.tb00328.x`

[18] Treisman, A. M. and G. Gelade, *A feature-integration theory of attention*, Cognitive psychology **12** (1980), pp. 97–136.

[19] Ware, C., “Visual Thinking: For Design,” Morgan Kaufmann, 2008.

# Implementing an Animated Visual $\lambda$-Calculus

Torsten Strobl* Mark Minas†

Computer Science Department
Universität der Bundeswehr München
85577 Neubiberg, Germany

**Abstract**

$\lambda$-calculus is a well-known formal system for investigating computability, recursion, functional programming, etc. Reduction rules define its semantics. Several visual representations have been proposed and used for making $\lambda$-calculus more comprehensible, easier to teach, or simply more fun to use. *Alligator Eggs* [14] is an example of such a playful representation where abstraction is represented by alligators and variables by alligator eggs. *Alligator Eggs* is also an animated visual language where eating alligators correspond to function application and hatching eggs to variable substitution. This paper shows how *Alligator Eggs* can be implemented as an animated system using the diagram editor generator DiaMeta. Boolean logic is used as a running example where alligator families model boolean terms. The generated animated system allows for an animated illustration of boolean evaluation.

## 1 Introduction

$\lambda$-calculus by A. Church [2] is a well-known formal system for investigating computability, recursion, functional programming, etc. It has originally been invented as an abstract computing approach, and was also the origin of functional programming by inspiring Lisp. But one can also use it for representing and evaluating boolean logic.

$\lambda$-calculus is based on expressions, which can be transformed by so-called reductions to other semantically equivalent expressions. However, the textual representation of $\lambda$-calculus expressions is not very intuitive since (a) this representation of expressions is difficult to grasp, and (b) the sequence of reductions does not immediately show what is going on. This paper improves this representation with respect to these issues (a) and (b). As a solution, we propose to utilize a visual representation of $\lambda$-calculus expressions. We apply animation for making immediately clear which reductions are applied to which sub-expression with which effect.

Several visual representations have been proposed and used for making $\lambda$-calculus more comprehensible, or easier to teach. *Alligator Eggs* [14] is such a visual and

[*] Email: Torsten.Strobl@unibw.de

[†] Email: Mark.Minas@unibw.de

playful language representing $\lambda$-calculus expressions. This language has alligators and their eggs as visual components. As an example, one type of reduction process can be represented by an alligator that eats other alligators and eggs. Afterwards, the eating alligator dies, but its eggs hatch into what the alligator ate. *Alligator Eggs* thus allows for obvious and intuitive animations of the formal concept of reductions. Hence, we chose *Alligator Eggs* for visually representing and animating $\lambda$-calculus.

Meta-tools greatly simplify the process of implementing editors for a specified visual language like *Alligator Eggs*. Several meta-tools also allow for implementing transformations of diagrams which can be used for realizing the reduction process. Examples for such meta-tools are DiaGen/DiaMeta [10], DEViL [4], GenGED [6], Tiger [1] and AToM$^3$ [9]. However, the support for realizing a visual editor that also allows for visualizing rather complex behavior by animations is limited. This paper describes an extension of the meta-model-based editor generator DiaMeta which allows for easy specification of visual languages with complex dynamic and animated behavior. The animated visual language *Alligator Eggs* is used as a running example although animating reductions of $\lambda$-calculus expressions is rather straight-forward.

The rest of the paper is structured as follows: The next section is a short introduction to the textual representation of $\lambda$-calculus, the visual language of *Alligator Eggs* and its dynamic behavior. Section 3 describes existing approaches in the context of specifications of animated visual languages and other visual representations of $\lambda$-calculus. Section 4 then briefly explains the existing meta-model-based editor generator and how it has been used to specify and generate the static aspects of *Alligator Eggs*. The extension of DiaMeta for animated visual languages is then described in Section 5 using the example of *Alligator Eggs*. The last section concludes the paper and reports about current work as well as plans for future work.

# 2 $\lambda$-Calculus

Textual $\lambda$-calculus expressions can be either a variable $x$, an application $FG$ of an expression $F$ to another expression $G$, or a $\lambda$-expression, i.e., an (anonymous) function $\lambda x.F$ where $F$ is again an expression. The latter is the key concept which means a function with a (bound) variable $x$ as parameter that may be used in $F$. The semantics of such expressions is defined by so-called reductions that reduce an expression to another, but semantically equivalent expression. $\alpha$-conversion changes bound variables, e.g., variable $x$ in $\lambda x.\lambda y.x$ to another, unused variable $z$, i.e., $\lambda x.\lambda y.x \to_\alpha \lambda z.\lambda y.z$. $\beta$-conversion applies a function to its argument by replacing each occurrence of the parameter by the argument, e.g., $(\lambda x.\lambda y.x)(\lambda z.z) \to_\beta \lambda y.\lambda z.z$. Of course, no variable in the argument may be the same as any variable in the function. Otherwise, bound variables must be changed by $\alpha$-conversion first.

As mentioned previously, $\lambda$-calculus can be used in order to evaluate boolean logic. The boolean values are represented by so-called Church booleans *true* $= \lambda x.\lambda y.x$ and *false* $= \lambda x.\lambda y.y$. It is easy to verify that the representations $not = \lambda p.\lambda x.\lambda y.pyx$, $and = \lambda p.\lambda q.pqp$, and $or = \lambda p.\lambda q.ppq$ actually are suitable definitions, i.e., reducing expressions that represent boolean terms is equivalent to evaluating the represented

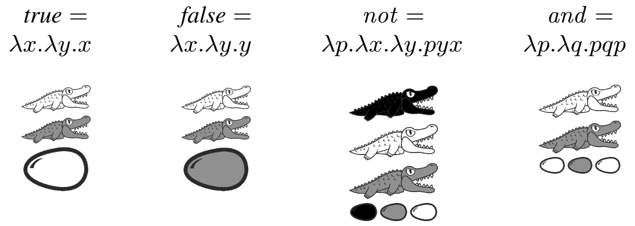


Figure 1: $\lambda$-expressions modeled by alligator families.

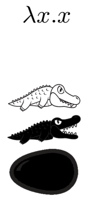


Figure 2: Old alligator

boolean terms[1]. An example evaluation is:

$$\begin{aligned}
\textit{and false true} \quad &= \quad (\lambda p.\lambda q.pqp)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)\\
&\to_\beta \quad (\lambda q.(\lambda x.\lambda y.y)q(\lambda x.\lambda y.y))(\lambda x.\lambda y.x)\\
&\to_\alpha \quad (\lambda q.((\lambda x.\lambda y.y)q(\lambda x.\lambda y.y)))(\lambda u.\lambda v.u)\\
&\to_\beta \quad (\lambda x.\lambda y.y)(\lambda u.\lambda v.u)(\lambda x.\lambda y.y)\\
&\to_\beta \quad (\lambda y.y)(\lambda x.\lambda y.y)\\
&\to_\beta \quad \lambda x.\lambda y.y\\
&= \quad \textit{false}
\end{aligned}$$

The visual $\lambda$-calculus language *Alligator Eggs* has hungry alligators, old alligators, and eggs as visual components. Let $E$ be an arbitrary $\lambda$-calculus expression and $\langle E\rangle$ the representation of $E$ in *Alligator Eggs*. $\langle E\rangle$ is a collection of visual components that are arranged in a tabular shape. A variable $x$ is represented by an egg $\langle x\rangle$ where the egg color corresponds to the variable's identifier $x$, i.e., different variables have eggs of different color. The application $FG$ of two expressions $F$ and $G$ with their visual representations $\langle F\rangle$ resp. $\langle G\rangle$ is drawn as $\langle F\rangle$ and $\langle G\rangle$ side by side with $\langle F\rangle$ left of $\langle G\rangle$, but aligned at their top. A $\lambda$-expression $\lambda x.F$ is represented by an hungry alligator (with open mouth) and the visual representation $\langle F\rangle$ of $F$. The color of the hungry alligator corresponds to the identifier $x$, i.e., every egg in $\langle F\rangle$ that represents $x$ has the same color as the hungry alligator. The hungry alligator is drawn on top of $\langle F\rangle$; its width is the same as the width[2] of $\langle F\rangle$. Altogether, they model a so-called "family". Figure 1 shows examples of expressions for boolean logic and how they are represented in *Alligator Eggs*. Colors have been replaced by shades of gray and hatching. Expressions in parentheses, $(F)$, are represented by an old alligator (a white alligator with closed mouth) and $\langle F\rangle$. Again, the alligator is drawn on top of $\langle F\rangle$ with the same width as $\langle F\rangle$. It is said that the old alligator "protects" $\langle F\rangle$. An example is shown in Figure 2.

$\beta$-conversion is translated into the *eating rule* in *Alligator Eggs*: The hungry alligator on top of $\langle\lambda x.F\rangle$ in an application $\langle(\lambda x.F)G\rangle$ "eats" the family $\langle G\rangle$, i.e., $\langle G\rangle$ gets deleted from the diagram. Then the hungry alligator dies, leaving $\langle F\rangle$. However, each

[1] Please note that application in $\lambda$-calculus is left-associative, i.e., $EFG = (EF)G$.

[2] This is an extension of the original description [14] of *Alligator Eggs* in order to make the language unambiguous.

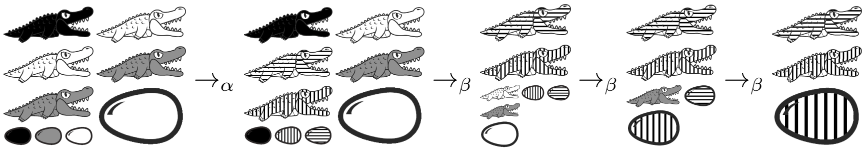


Figure 3: Evaluation of *not true* $= (\lambda p.\lambda x.\lambda y.pyx)(\lambda x.\lambda y.x)$ in *Alligator Eggs*.

egg in $\langle F \rangle$ with the same color as the died alligator gets replaced by $\langle G \rangle$. $\alpha$-conversion is also necessary in *Alligator Eggs*. This is realized by the so-called *color rule* that appropriately changes the color of each hungry alligator and its eggs in $\langle \lambda x.F \rangle$ if this color also occurs in $\langle G \rangle$.[3] Finally, the *old age rule* defines the semantics of old alligators: An old alligator (representing parenthesis in *Alligator Eggs*) dies as soon as there is only a single component directly below the old alligator (e.g. like in Figure 2). A longer example, the evaluation of *not true* in *Alligator Eggs*, is shown in Figure 3.

# 3 Related Work

There are several tools supporting the generation of editors from visual language specifications and meta-models [6, 9, 1, 4]. However, only few of them allow animation specifications or the creation of animated editors in general. Most tools or common approaches supporting animation specifications are very limited, e.g. there is no possibility for interaction during animation, the specification of concurrent animation steps is complicated or impossible, or flexibility is missing. Also older versions of DiaMeta rudimentarily support animations [11], but practically this only means that diagram (state) changes, especially position changes, can be interpolated.

Some of the listed limitations are attributed to the utilization of transformation rules in simulation and animation, because these transformations must basically be considered as atomic operations. Therefore, a lot of efforts are put into the investigation of transformations with specified timed behavior. Transformation rules could contain a (conditional) duration and further mechanisms like interruptibility. In articles like [7] and [8] these topics are described in more detail. In [12] a graphical notion is shown, and in [13] also an event-based approach is presented.

An exemplary generator system, which - similar to DiaMeta - also applies graph models and transformations, is GenGED [6]. The system not only allows the implementation of visual language editors, but also to write (rule-based) simulation specifications. The visualization of the simulation - in this case called animation - can be specified separately, so this visualization can have a completely different layout compared to the visual language itself. In this way, the animation can be presented in another domain-specific layout, for which the term *animation view* has been introduced. However, GenGED editors cannot show such views. Instead, an automatically running "movie" has to be exported.

Next to *Alligator Eggs*, there are also other visual representations for $\lambda$-calculus. Vex uses circles for representing $\lambda$-expressions and variables [3]. Parameters are rep-

[3] This *color rule* is a bit more specific that in [14].

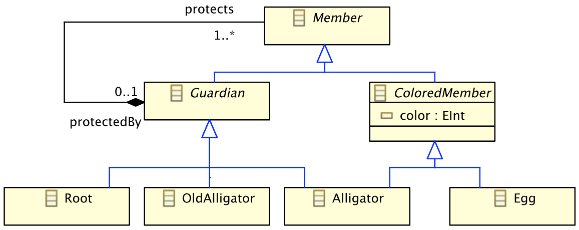


Figure 4: Architecture of a diagram editor based on DIAMETA.

resented by internally tangential circles, application by externally tangential circles. The binding of variables is explicitly represented by connecting lines.

# 4 DIAMETA

DIAMETA is a framework together with a specification tool for generating diagram editors from a specification [10]. The abstract syntax of a diagram language has to be specified as a meta model based on EMF [5]. Figure 4 shows the meta model for *Alligator Eggs* which comprises the composite pattern: each expression as a diagram represents an (expression) tree. *Guardian* comprises a composite node and represents an object with a (horizontal) sequence of sub-diagrams below. Concrete sub-classes are *Root* that represents a complete diagram, *Alligator* for hungry alligators, and *OldAlligator* for old alligators. Each instance of these classes contains a sequence of protected objects. Instances of *Egg*, which represents alligator eggs, are the leaves of this composite pattern. The *protects* association is ordered from left to right (not shown in Figure 4). Instances of this meta model, hence, uniquely represent *Alligator Eggs* diagrams.

Each editor generated by DIAMETA is a free-hand editor, i.e., the user may arrange visual components freely on the screen. The specification must contain descriptions of all required visual components. For *Alligator Eggs*, these are hungry and old alligators as well as eggs. When the editor user arranges such diagram components on the screen, the editor has to check whether the arrangement is a correct diagram, and, if so, what its syntactic structure is. Each editor generated by DIAMETA uses a generic architecture for solving this problem, see Figure 5: The editor consists of a drawing tool which is used by the editor user for arranging the diagram components on the screen. The arrangement is then internally represented by a so-called *graph model* as a homogeneous representation which can be used for all diagram languages. Figure 6(b) shows the *graph model* for the simple *Alligator Eggs* diagram in Figure 6(a). Each diagram component is represented by a *component node*, here *alligator*, *egg*, and *root*. The latter represents an invisible component representing the whole canvas. The specification describes how each diagram component is represented: Each component has a certain number of attachment areas. The canvas and eggs have a single attachment area, alligators have two, the alligator shape itself, and the area from the alligator to the

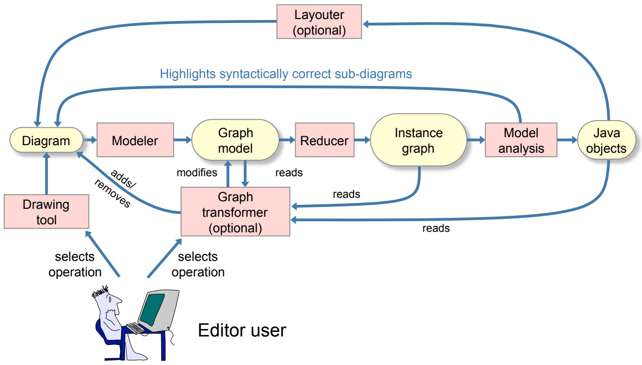


Figure 5: Architecture of a diagram editor based on DiaMeta.

bottom of the canvas. Each of these areas is represented by an attachment node that is connected with its component node by an edge with labels *canvas*, *shape*, or *below*. A *relation edge* connects two attachment areas that are related in a specific way. E.g., a *protects* edge connects the corresponding nodes if an alligator or egg lies underneath another alligator.

The graph model may grow quite large. E.g., a stack of $n$ alligators requires $O(n^2)$ *protects* edges. The reducer (see Figure 5) transforms this graph into the *instance graph* by applying reducer rules in the specification. They are omitted here since they are not crucial for the setting of this paper. The obtained instance graph represents an instance of the specified meta model if the diagram is syntactically correct. This is checked by the model analysis.[4] Model analysis provides feedback to the user about diagram parts that are not syntactically correct by highlighting those diagram components. Model analysis also instantiates the EMF model that implements the meta model. This data structure can then be used by an automatic layout facility for beautifying or layouting the diagram.

Free-hand editing is complemented by (optional) graph transformation rules that utilize a graph transformation facility. Graph transformations may be specified for "implementing" complex diagram modifications that are triggered by the editor user. However, such transformations that operate on the graph model, but that may use information from the instance graph, too, are also a helpful mechanism for animation support.

# 5 Animated Alligator Eggs

This section describes how the animated aspects of *Alligator Eggs* can be specified. First the desired animations are described, and subsequently the different animation states are identified. Based on such states the used animation approach is explained,

[4] Model analysis is actually more sophisticated. Not all classes must have been determined by the reducer rules. Model analysis uses constraint solving techniques for identifying undetermined classes [10].

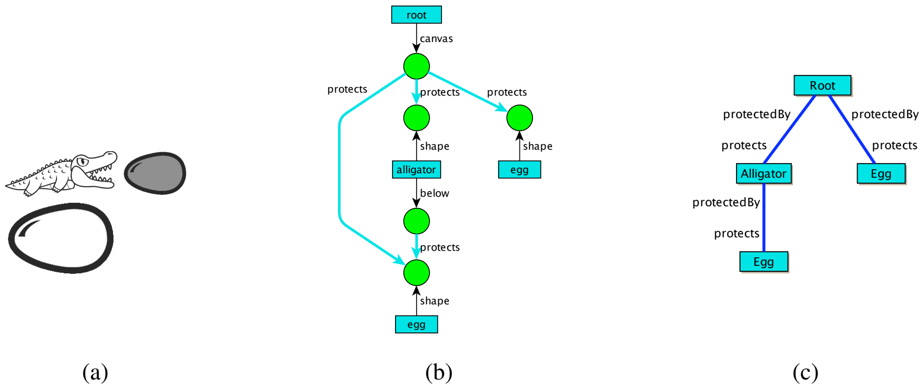


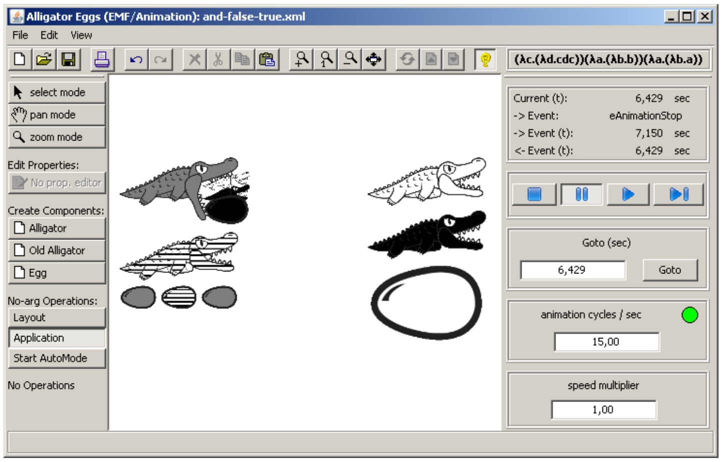
Figure 6: (a) Visual representation of $(\lambda x.x)y$, (b) graph model, and (c) instance graph.

including an event-based strategy. Finally, the concept is compared with common approaches of other frameworks, which are similar to DiaMeta.

## 5.1 Animation Description

Section 4 has described in short how the static part of *Alligator Eggs* can be specified. As a result, a fully functional editor for static *Alligator Eggs* diagrams can already be generated. It has also been mentioned that DiaMeta supports the specification of graph transformations. The step semantics of *Alligator Eggs* (see Section 2), can be implemented by utilizing these transformations. Indeed, only one, but rather complex graph transformation is required for this. The operation has to analyze the graph, decide on the next applicable rule and finally arrange modifications in order to get the results after the step. By triggering such a transformation the editor's user can watch the conversions step-by-step. However, users cannot follow the reduction process in more detail, which could be crucial for a better understanding of *Alligator Eggs* and the $\lambda$-calculus. Therefore, it is desirable to generate an editor which also shows internal processes by animating them. Each individual part of a rule application shall be visualized in a movie-like fashion. Again, the description in [14] was used as orientation.

The *eating rule* is split into subparts. First, the alligator eats the family in front of him. Therefore, the family moves towards the alligator's mouth. Meanwhile, the alligator is snapping, and it is also meaningful to decrease the victim's overall size. Afterwards, the alligator dies, so the shape actually rotates until the alligator is lying on its back, and it disappears by shrinking. Intermediately, the "reborn family" will hatch out of the according eggs. This means, that the egg cracks and the families appear. This way, the user can track the way of the eaten family, and also the alligator, which causes the action, can be identified. In order to have enough space for the new structure the whole diagram is also re-layouted. The *old age rule* is applied similarly. The old alligator dies and the diagram is layouted (both overlapping in time). Finally, during the *color rule* affected components are recolored, which is transacted by cross-



Alligator Eggs (EMF/Animation): and-false-true.xml

(λc.(λd.cdc))(λa.(λb.b))(λa.(λb.a))

Current (t): 6,429 sec
-> Event: eAnimationStop
-> Event (t): 7,150 sec
<- Event (t): 6,429 sec

Goto (sec): 6,429

animation cycles / sec: 15,00

speed multiplier: 1,00

Figure 7: Screenshot of the *Alligator Eggs* editor while alligator is eating

fading their colors. In Figure 7 an exemplary scene while processing the eating rule is illustrated.[5]

## 5.2 Animation States

The previous description already indicates that required animations can be separated into multiple, also concurrent, phases. The idea now is to encode informations about currently running phases (along with possible parameters) within the diagram's state or even the state of individual components or component groups. Specific state transitions then imply the transition from one phase to the next. Figure 8 shows a possible translation of the textual animation description (see Section 5.1) into a state machine diagram, even with more details like the duration of the individual phases. In particular, it depicts the states of one resp. two families during the execution of a rule, and how individual members are animated in the meantime. In the lower left corner the diagram's start state *Static* can be found, the editor must not show animation for involved diagram components here. If a rule shall be applied, the state switches to state *Animated*, if possible. Before this transition, the system actually has to determine the applicable rule as shown (also with according priorities).

In the following, the most complex substate of *Animated* will be exemplified: *Eating Rule*. In this state the associated rule with the same name is processed. Directly after the rule is applied, involved components pass through the *Eating Phase*, also a substate. 3 sec. afterwards, the state automatically passes over into the *Rebirth Phase*. Concurrently, the dying alligator and the animated re-layout process are shown during this phase. All substates again can pass over after 3 sec., however, the re-layout process is delayed by 1.5 sec., so actually 4.5 sec. are required.

[5] An animated example can be found at: http://www.youtube.com/user/diametaanimated

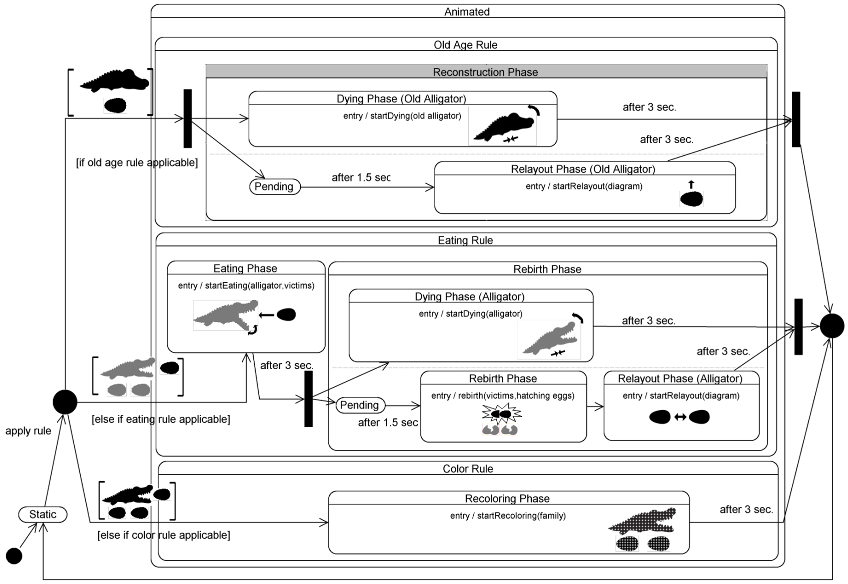

Animated
Old Age Rule
Reconstruction Phase
Dying Phase (Old Alligator)
entry / startDying(old alligator)
after 3 sec.
after 3 sec.
[if old age rule applicable]
Pending
after 1.5 sec
Relayout Phase (Old Alligator)
entry / startRelayout(diagram)
Eating Rule
Eating Phase
entry / startEating(alligator,victims)
Rebirth Phase
Dying Phase (Alligator)
entry / startDying(alligator)
after 3 sec.
after 3 sec.
after 3 sec.
Rebirth Phase
entry / rebirth(victims,hatching eggs)
Relayout Phase (Alligator)
entry / startRelayout(diagram)
Pending
after 1.5 sec
[else if eating rule applicable]
apply rule
Color Rule
Recoloring Phase
entry / startRecoloring(family)
after 3 sec.
Static
[else if color rule applicable]

Figure 8: State diagram - rule application

The example has shown the definition of *animation states* for a group of components, but such states can also be defined for individual components. For instance, each component can be in state *Static* or *Animated*. The *Animated* state itself can be separated into multiple substates: *Rotating*, *Shrinking*, *Moving*, *Snapping* (only alligators), *Hatching* (only eggs), etc. Again, concurrent states are possible, e.g., if an alligator is rotating and shrinking. Figure 9 shows a timing diagram, which outlines the states of individual diagram components while processing the *eating rule*.

## 5.3 Animation Concept

The way a visual language like *Alligator Eggs* is presented is specified by its *concrete syntax* (in contrast to the abstract syntax, e.g., given by the meta-model). In DiaMeta this syntax can be specified for each visual component. Until now, the possibilities have been designed for specifying non-animated (static) components. Considering this specification and the internal *graph model* together with attributes of component nodes, e.g., the $x$ and $y$-position of the component, the editor is able to draw the static visual representation. However, this basic approach is not necessarily limited to draw static diagrams, if time is considered for the mapping of the graph to its representation. Hereby, the time $t$ could be a given parameter for the whole drawing process. Other *animation parameters* can be stored within component nodes in exactly the same way as parameters for the static representation. Possible animation parameters would be a given start time, a start position or a constant velocity. With these informations a

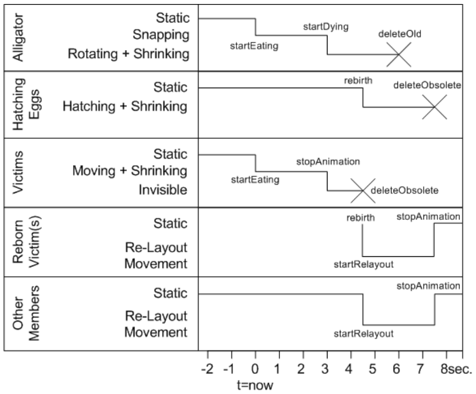


Figure 9: Timing diagrams - *eating rule*

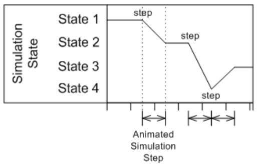


Figure 10: State transitions consuming time

component can be visualized at different $x$-positions depending on the drawing time $t$. In an animation sequence with continuously increasing $t$ and a periodically redrawn diagram, this results in a smoothly animated motion. However, animation parameters are not limited to simple values in the affected component node. Also the relations to other components and their attributes can be taken into account, e.g. an attached arrow component which specifies the movement direction.

In order to implement *Alligator Eggs*, we chose this approach and extend the DiaMeta framework by adding an animation package. This package provides the required functionality, e.g. the global time parameter. The specification for visualizing the animated component itself is currently a programmed block of code in order to allow flexible mathematical calculations depending on arbitrary information contained in the graph model.

Now, also the animation states described in Section 5.2 can be reconsidered. The state of an individual component can be stored within the component's attributes, and this state is the basis for deciding how the component is animated. In addition, animation parameters (also attributes) can be required for the animated visualization. For example, if an alligator is in state *Rotating* (see Figure 9), the following attributes are used: $animationStart$, $animationStop$, $angleStart$, $angleStop$. Together with the global time and a specified algorithm, e.g. linear interpolation between start/stop time and start/stop angle, the animation can be calculated. Another example is an alligator's *Eating* state in which the alligator's mouth is snapping. A simple flag $snapping$, along with $animationStart$ and $animationStop$, is sufficient in this case. The alligator's component is drawn via two images: the main body and the lower jaw. If the flag is not set, the lower jaw is drawn statically onto the rest of the body. Otherwise, it is drawn slightly rotated around the jaw's joint. Thereby, the angle is interpolated (cosine-based) between fixed values with periodic repetitions.

## 5.4 Events

Of course, animation parameters can be changed by the user like other attributes. Components can be dragged via mouse, or property editors can be used to change internal values. However, changing animation parameters this way does not make much sense for many use cases. A useful mechanism in order to set animation parameters, or animation states to be more general, are graph transformations, which are already available in DiaMeta. With them important values like start/stop times can be calculated automatically, and complex diagrams can be analyzed and modified in an established way. As a consequence, they can be used to change animation states. Some of these graph transformations required in *Alligator Eggs* are already shown in Figure 8. For example, if the *eating rule* shall be applied and the *Eating Phase* is entered, the *startEating* graph transformation is applied on entry, and this transformation actually sets the new animation state. Also arguments can be passed to the transformation, in this case that is the alligator and its victims. A more complex transformation is *startRelayout*. It is able to layout the whole diagram. Indeed, the functionality can be shared with the static layout specification, which is already specified (see Section 2). The difference is that the results of the algorithm are either used for the definite positions or the arrival points of linear movements.

The remaining question concerns the point in time these graph transformations, which actually control the animation flow, are executed. Therefore, it must be possible to specify *events*. Whenever an event is triggered, the according transformation is performed. DiaMeta already provides the specification of an obvious event type: *user events*. It is possible to add editor buttons, which initiate graph transformation, if clicked. However, also other unanticipated events can be considered as user events: moving components via mouse, key strokes, adding or removing components. The state transition from *Static* to *Animated* is an example of user events, because it is actually triggered, if the user clicks on the button "Application" (see Figure 7).

The second type of events are *timed events*. In Figure 8 several transitions are fired after specific time events, e.g. "after 3 sec". Therefore, the DiaMeta animation package also includes an event queue, wherein timed events must be registered. Thereby, the event times are calculated based on the graph model. For example, three seconds must be added to the point in time when the *Eating Phase* was entered in order to obtain the time of the next transition. Also more complex calculations are possible, e.g. computing the time when two components collide on their trajectory, etc. Individual component attributes as well as the whole graph structure can be taken as basis of the calculation. Again, graph matching mechanisms can help finding and calculating the times of these events. The time along with affected components, the event context, is then stored in the event queue. The queue, hence, must be revised after each change of the graph model. If components are added, removed or changed, it is also possible that new events appear and already registered events must be modified or updated. As soon as the global time $t$ reaches the time of an enqueued event, the graph transformation that has been specified for this event type gets triggered along with the actual event context (affected components). As described, this can lead to a change of the animation state. If more than one event share the same time, the DiaMeta approach is to priorize some events by specification.

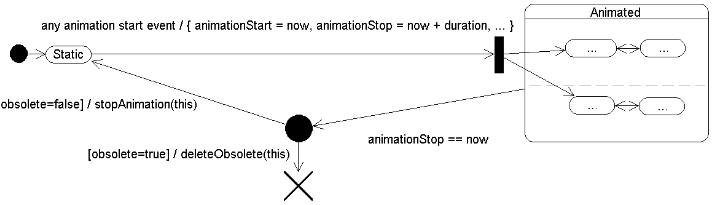

any animation start event / { animationStart = now, animationStop = now + duration, ... }
Animated
Static
...
...
...
...
[obsolete=false] / stopAnimation(this)
[obsolete=true] / deleteObsolete(this)
animationStop == now

Figure 11: State diagram - animation states of individual components

There is also a third event type: *immediate events*. They occur time-independently as soon as a specific state is reached, e.g. after a graph transformations. An example is the event which fires the transition from the *Rebirth Phase* to the *Relayout Phase (Alligator)*. Assuming the former state, the connected graph transformation (*rebirth*) recreates an instance of the eaten family for each hatching egg. Directly afterwards, the state changes to *Relayout (Alligator)* and the graph transformation for re-layouting, *startRelayout*, is performed.

Otherwise, the work for creating the *Alligator Eggs* event specification for starting animations based on the state machine diagram is straight-forward. However, there is still the need to stop animations after specific phases end. Of course, these animation stops could be initiated by individual events, which are triggered if a state exits. Figure 11 shows an elegant alternative, if "animation stop" always means to reset all animation parameters, no matter which component type. The component's state shall switch to *Static* automatically, if the global time $t$ reaches a component's animation parameter $animationStop$. Regularly, this will cause the graph transformation *stopAnimation*. However, as a nice additional feature, if the flag $obsolete$ is set for the component, it is marked for deletion and can even be deleted automatically by the graph transformation *deleteObsolete* (cp. Figure 9).

## 5.5 Comparison With Other Approaches

As already mentioned in Section 3, animation within diagram editors, especially in the field of generator frameworks, is often used as a supplement to simulation (e.g. GenGED [6], DEViL [4]). Thereby, animation is used for user-friendly visualization of individual simulation steps. Without additional techniques each step is an atomic operation. However, its duration - which is actually 0 - must be stretched in time in order to animate changes. Figure 10 shows the chronological sequence of state transitions, which are applied this way. In the following, this animation concept is called *transition animation*. On the contrary, the *state animation* concept outlined in this paper sticks with instantaneous transitions (cp. Figure 9).

The *Alligator Eggs* editor can be generated via both approaches. Each rule application can be considered as simulation step. With transition animations a way to specify the whole animation for this single step is required. In case of the rather complex eating rule, specification possibilities must be very flexible, e.g. an arbitrary amount of reborn families must be animated. A specification following the concept of state animations applies to shorter, usually less complex animation sequences, which can be described via graph transformations and events.

Furthermore, transition animation sequences cannot overlap in time, because the underlying simulation steps themselves cannot overlap. For example, the *Alligator Eggs* rule applications - considered as simulation steps - must be executed consecutively. However, it would be possible to execute several independent rule applications simultaneously and time-shifted.[6] Components can be animated independently, and by using the event approach they could also interact with each other, e.g. collision events during a movement. Even unpredictable interactivity while running the animation is possible (*user events*), and user activities can be considered in the animated visual language's design. In case of the *Alligator Eggs* editor, it is possible to delete hatching eggs or victims during the *Eating Phase*, for example.

Finally, the state animation concept also covers the animation of diagram languages, whose underlying models cannot be simulated. Animations could always be used as eye-catcher to highlight particular diagram elements, for example.

# 6 Conclusions

This paper has shown that also animated editors for visual logic languages like *Alligator Eggs* can be successfully specified and generated using DIAMETA. The resulting editor can be used in order to model visual $\lambda$-expressions. Animations enable the user to track changes during the reduction of expressions.

The DIAMETA framework has been extended in order to apply the presented animation strategy. This strategy differs from other approaches widely spread in the area of meta-tools and editor generator frameworks. Even some limitations of other approaches are resolved. However, the specification complexity of animations and events must still be improved. Currently, the concrete visualization and calculations must be written by hand. Patterns for different animation types and processes including their parameters would be desirable, even if loosing flexibility. With such patterns, it is also more likely, that higher-level cartoon-like animations (e.g. by using predefined effects), which are especially suitable for an editor like *Alligator Eggs*, are used instead of simple interpolated transformations. The approach should also be investigated with regard to different types of animated visual languages. Perhaps certain specification procedures, e.g. using state machine diagrams, can be identified and described in order to simplify overly complex specifications. Finally, the applied strategy also seems to be suitable for specifying highly interactive animated languages.

# References

[1] E. Biermann, C. Ermel, J. Hurrelmann, and K. Ehrig. Flexible visualization of automatic simulation based on structured graph transformation. In *VLHCC '08: Proc. IEEE Symp. on Visual Languages and Human-Centric Computing*, pages 21–28, Washington, DC, USA, September 2008. IEEE Computer Society.

[6] This feature has been realized with the concept presented in this article

[2] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:354–363, 1936.

[3] W. Citrin, R. Hall, and B. Zorn. Programming with visual expressions. In *VL '95: Proc. 1995 IEEE Symp. on Visual Languages*, pages 294–301, Darmstadt, Germany, Sep 1995. IEEE Computer Society Press.

[4] B. Cramer and U. Kastens. Animation automatically generated from simulation specifications. In *VLHCC '09: Proc. IEEE Symp. on Visual Languages and Human-Centric Computing*, Corvallis, Oregon, USA, September 2009. IEEE Computer Society.

[5] EMF – Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/, 2009.

[6] C. Ermel. *Simulation and Animation of Visual Languages based on Typed Algebraic Graph Transformation*. PhD thesis, Tech. Univ. Berlin, Fak. IV, Books on Demand, Norderstedt, 2006.

[7] S. Gyapay, R. Heckel, D. Varro, and D. Varr. Graph Transformation with Time: Causality and Logical Clocks. In *ICGT '02: Proc. 1st Int. Conf. on Graph Transformation*, pages 120–134. Springer-Verlag, 2002.

[8] H.-J. Kreowski and S. Kuske. Graph transformation units with interleaving semantics. *Formal Asp. Comput.*, 11(6):690–723, 1999.

[9] J. d. Lara and H. Vangheluwe. AToM3: A Tool for Multi-formalism and Meta-modelling. In *FASE '02: Proc. 5th Int. Conf. on Fundamental Approaches to Software Engineering*, pages 174–188, London, UK, 2002. Springer-Verlag.

[10] M. Minas. Generating meta-model-based freehand editors. In *GraBatTs '06: Proc. 3rd Int. Workshop on Graph Based Tools, Satellite event of 3rd Int. Conf. on Graph Transformation*, Natal, Brazil, September 2006.

[11] M. Minas and J. Gottschall. Specifying animated diagram languages. In *TVL '97: Proc. Workshop on Theory of Visual Languages*, Capri, Italy, 1997.

[12] J. E. Rivera, C. Vicente-Chicote, and A. Vallecillo. Extending visual modeling languages with timed behavioral specifications. In *IDEAS 2009: Proc. 12th Iberoamerican Conf. on Requirements Engineering and Software Environments*, pages 87–100, Colombia, April 2009.

[13] E. Syriani and H. Vangheluwe. Programmed Graph Rewriting with Time for Simulation-Based Design. In *ICMT '08: Proc. 1st Int. Conf. on Theory and Practice of Model Transformations*, pages 91–106, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] B. Victor. Alligator eggs! A puzzle game. http://worrydream.com/AlligatorEggs/, May 2007.