A New Language for the Visualization of Logic and Reasoning

Gem Stapleton and Simon Thompson
University of Kent
Canterbury, UK
{g.e.stapleton,s.j.thompson}@kent.ac.uk

Andrew Fish, John Howse and John Taylor
Visual Modelling Group
University of Brighton
Brighton, UK
{andrew.fish,john.howse,john.taylor}@brighton.ac.uk

Abstract

Many visual languages based on Euler diagrams have emerged for expressing relationships between sets. The expressive power of these languages varies, but the majority can only express statements involving unary relations and, sometimes, equality. We present a new visual language called Visual First Order Logic (VFOL) that was developed from work on constraint diagrams which are designed for software specification. VFOL is likely to be useful for software specification, because it is similar to constraint diagrams, and may also fit into a Z-like framework. We show that for every First Order Predicate Logic (FOPL) formula there exists a semantically equivalent VFOL diagram. The translation we give from FOPL to VFOL is natural and, as such, VFOL could also be used to teach FOPL, for example.

1 Introduction

There is a growing interest in the use of languages based on Euler diagrams for expressing and reasoning about logical statements [1, 3, 4, 6, 7, 11, 12, 13]. The majority of these languages are monadic (meaning they can only express statements involving unary relations) and, hence, very limited in expressive power. *Constraint diagrams* [9] can make statements involving binary relations (as well as unary relations) and have been used to model object oriented software systems [8, 10]. They have been designed to complement the diagrammatic theme of the Unified Modeling Language (UML). In this paper we present a new diagrammatic language called Visual First Order Logic (VFOL) that grew out of work on constraint diagrams. VFOL is likely to be useful for software specification in the context of UML, because it is similar to constraint diagrams, and may also fit naturally into a Z-like framework. Another potential application domain is for teaching logic to computer scientists.

In figure 1 there are two constraint diagrams. The asterisks, labelled s, represent universal quantification and the nodes, labelled t, represent existential quantification. In order to disambiguate the diagrams, a *reading tree* [5] is used to indicate the order in which the quantifiers are to be interpreted. The reading trees both assert that s is read before t and d_1 expresses that A and B are disjoint, every element in A is related to precisely one element, under the relation r, which is in B. The diagrams d_1 and d_2 are examples of *unitary diagrams* which can be joined together using logical connectives such as 'and' and 'or'.

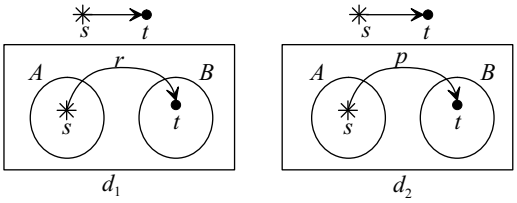

Figure 1. Two constraint diagrams.

In the constraint diagram language, it is difficult (if not impossible) to express statements such as

$$A \cap B = \emptyset \wedge \forall s \in A \, \exists t \in B \quad \{s\}.r = \{t\} \vee \{s\}.p = \{t\} \qquad (1)$$

where $\{s\}.r$ (which is called a *navigation expression*) denotes the relational image of r when the domain is restricted to $\{s\}$ (similarly for $\{s\}.p$). One reason that (1) is difficult to express is because of the disjunctive formula inside the scope of the universal quantifier. The two diagrams in figure 1 can be taken in disjunction, giving $d_1 \vee d_2$, to express

$$A \cap B = \emptyset \wedge (\forall s \in A \exists t \in B \, \{s\}.r = \{t\} \vee \forall s \in A \, \exists t \in B \{s\}.p = \{t\}),$$

but this is not semantically equivalent to (1).

Constraint diagrams are good at expressing conjunctive information inside unitary diagrams. All of the quantification occurs inside unitary diagrams, which means that First

Order Predicate Logic (FOPL) sentences involving universal quantification followed by disjunctive formulae (such as (1)) may not be realizable as constraint diagrams.

VFOL retains many features of constraint diagrams that are useful for modelling software systems. In particular, navigation expressions can still be made in VFOL. In order

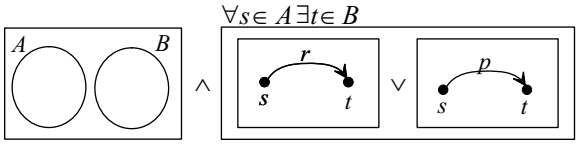

Figure 2. A VFOL diagram.

to represent relations that have arity three or greater we use multi-sourced arrows and quantification is an explicit operation which does not appear symbolically within a (unitary) diagram. Performing quantification outside diagrams also removes the need for reading trees to accompany the diagrams: the order of quantification is automatically explicit. An example of a VFOL diagram is shown in figure 2, which is semantically equivalent to statement (1) above. Unlike constraint diagrams, in VFOL distinct nodes do not necessarily denote distinct elements. This is similar to the interpretation of constant sequences in Euler/Venn diagrams: distinct constant sequences do not necessarily denote distinct individuals [13].

The syntax of VFOL and FOPL are given in section 2. The semantics of VFOL and FOPL are specified in section 3 and in section 4 we map FOPL formulae to semantically equivalent VFOL diagrams.

2. Syntax

2.1. An Alphabet

In this section, we introduce an alphabet that will be common to VFOL and FOPL. Firstly, we have a countably infinite set of **variables**, $\mathcal{V} = \{x_1, x_2, ...\}$. We define a set of **function symbols**, $\mathcal{F} = \{f_1, f_2, ...\}$, and a set of **relation symbols**, $\mathcal{R} = \{r_1, r_2, ...\}$. These two sets may be finite. A function $\alpha\colon \mathcal{F} \cup \mathcal{R} \to \mathbb{N}$ returns the arity of each symbol. Relation symbols have arity at least one. Every variable is a **term**. If f_i is a function symbol and $t_1, ..., t_{\alpha(f_i)}$ are terms then $f_i(t_1, ..., t_{\alpha(f_i)})$ is a **term**. The set of terms is denoted $\mathcal{T}$.

In this paper, we will use symbols of the form f_i and r_i in our examples. We expect users of the notation will prefer to choose sensible names for their functions and relations.

2.2. Syntax of VFOL

Relation symbols with arity one, $\mathcal{R}_1 = \{r_i \in \mathcal{R} : \alpha(r_i) = 1\}$, will be used to label contours, and we call them **given contour labels**. Function symbols with arity 0, $\mathcal{F}_0 = \{f_i \in \mathcal{F} : \alpha(f_i) = 0\}$, are constants. The remaining relation and function symbols will be used to label arrows. A special symbol, $\mathcal{U}$, represents the universal set.

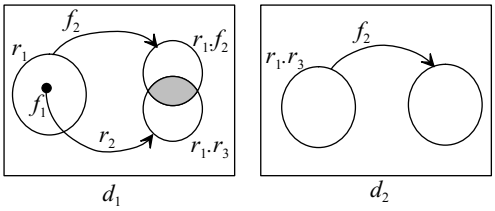

Figure 3. Two VFOL diagrams.

Example 2.1 The diagram d_1 in figure 3 contains one given contour, labelled r_1. The other two contours are *derived contours* which represent the image of a relation or function under certain restrictions. The function symbol f_1 has arity 0 and is, therefore, a constant. Locating f_1 inside r_1 expresses that $f_1 \in r_1$. The arrow sourced on r_1, labelled with the unary function symbol f_2, targets a derived contour. This arrow expresses that $r_1.f_2$ (the image of f_2 when the domain is restricted to r_1) is disjoint from r_1. This derived contour is labelled with the navigation expression $r_1.f_2$. The arrow labelled with the binary relation symbol r_2 expresses that $f_1.r_2$ equals $r_1.r_3$. By the use of shading we have expressed that $r_1.r_3$ is disjoint from $r_1.f_2$.

Derived contour labels allow us to talk about the image of a relation or function without using arrows. These labels provide an efficiency and flexibility of notation that was not present in constraint diagrams (where derived contours are never labelled). To make statements about the image in constraint diagrams, one had to first construct the image using sequences of arrows. The benefits of our new approach become apparent when constructing complex navigation expressions.

In the example above, in d_1 the derived contour label $r_1.f_2$ is redundant, since the arrow targeting $r_1.f_2$ is sourced on r_1 and labelled f_2. We will not force users of the notation to label derived contours, unless a label is essential for the interpretation of the diagram. For example, d_2 in figure 3 contains a derived contour with label $r_1.r_3$. Without this label, we could not interpret d_2 in a first order manner. The other derived contour in d_2 has not been labelled, since it represents the set $(r_1.r_3).f_2$. For space reasons, we omit the conditions under which a derived contour label is required, but they are similar to the readability criteria given for constraint diagrams in [5].

Further examples of derived contour labels are $\{x\}.f_2$ and $(\{x\} \times r_1).f_3$ where f_3 is a binary function symbol. From these simple derived contour labels we can construct more complex expressions, such as $((\{x\} \times r_1).f_3).f_2$. In order to formally define derived contour labels, we start with the set $\mathcal{DC}_0 = \{r : r \in \mathcal{R}_1 \cup \mathcal{T}\} \cup \{\mathcal{U}\}$. The elements of $\mathcal{DC}_0$ are not derived contour labels but are essential for our inductive definition below.

1. If f is a function symbol in $\mathcal{F} - \mathcal{F}_0$ and $D_1, ..., D_{\alpha(f)}$ are in $\mathcal{DC}_i$ then $((D_1 \times ... \times D_{\alpha(f)}).f)$ is in $\mathcal{DC}_{i+1}$.
2. If r is a relation symbol in $\mathcal{R} - \mathcal{R}_1$ and $D_1, ..., D_{\alpha(r)-1}$ are in $\mathcal{DC}_i$ then $((D_1 \times ... \times D_{\alpha(r)-1}).r)$ is in $\mathcal{DC}_{i+1}$.
3. Every element of $\mathcal{DC}_i$ is in $\mathcal{DC}_{i+1}$.

The set of **derived contour labels** is

$$\mathcal{DC} = \bigcup_{n \in \mathbb{N}} \mathcal{DC}_n - \mathcal{DC}_0.$$

We define $\mathcal{CL} = \mathcal{R}_1 \cup \mathcal{DC}$ to be the set of **contour labels**. The definition of $\mathcal{DC}$ could be simplified if we went against convention and defined the arity of each function symbol to be the number of inputs plus 1: we could treat relation symbols and function symbols in the same way. This is also the case for several other definitions given later in the paper.

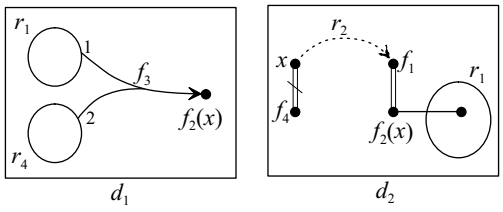

Figure 4. Multi-sourced arrows and equality.

Example 2.2 The diagram d_1 in figure 4 contains a function label f_3 that has arity 2. The arrow has two sources and the order in which these are read is indicated by labelling the arrow. The diagram expresses that $r_1 \cap r_4 = \emptyset$ and $(r_1 \times r_4).f_3 = f_2(x)$, where x is a variable which is free in d_1. Here, in our informal explanation, we have identified $f_2(x)$ with $\{f_2(x)\}$.

The arrows in a diagram can be sourced and targeted on terms, contours and the rectangle which encloses the diagram. In our formal syntax, this rectangle is denoted by $\mathcal{U}$ and represents the universal set. We define the set of sources and targets of the arrows to be $\mathcal{AST} = \mathcal{T} \cup \mathcal{CL} \cup \{\mathcal{U}\}$. Arrows are defined as follows.

1. If f is a function symbol in $\mathcal{F} - \mathcal{F}_0$, $s \in \mathcal{AST}^{\alpha(f)}$ and $t \in \mathcal{AST}$ then (f, s, t) is an **arrow**.
2. If r is a relation symbol in $\mathcal{R} - \mathcal{R}_1$, $s \in \mathcal{AST}^{\alpha(r)-1}$ and $t \in \mathcal{AST}$ then (r, s, t) is an **arrow**.

The set of all arrows is denoted $\mathcal{AR}$. The **label** of arrow (l, s, t) is l, the **source** is s and the **target** is t; the **components** of s are the elements of the set $Com(s) = \{a_i : s = (a_1, ..., a_n) \wedge 1 \leq i \leq n\}$.

We assume that the sets $\mathcal{T}$, $\mathcal{F}$, $\mathcal{R}$, $\mathcal{DC}$, $\mathcal{AR}$, and $\{\mathcal{U}\}$ are pairwise disjoint.

Example 2.3 The diagram d_2, in figure 4, expresses that $f_1 = f_2(x)$, by the use of a pair of parallel straight line segments, like an equals sign. We say that f_1 and $f_2(x)$ are *identified*. Similarly, $x \neq f_4$ and we say that x and f_4 are *separated*. The term $f_2(x)$ has a location that consists of two *zones*. In a drawn diagram, a zone can be described by a two-way partition of the contour label set. In our formalization, a zone is an ordered pair of disjoint sets of contour labels, (a, b), where a contains the zone and b excludes the zone. The diagram d_2 expresses that $f_2(x)$ is in r_1 or $\mathcal{U} - r_1$. Since $f_2(x) = f_1$, we can deduce from d_2 that $f_2(x) \in \mathcal{U} - r_1$. Finally, d_2 contains a *dashed arrow*. Dashed arrows, which are not part of the constraint diagram language, allow us to represent partial information. In the particular case here, the arrow expresses that $\{x\}.r_2$ includes f_1. In other words, x is related to (at least) f_1 under r_2.

We are now in a position to define unitary diagrams.

Definition 2.1 *A **unitary diagram** is a tuple*

$$d = \langle CL, T, SA, DA, Z, SZ, \lambda, \iota, \sigma \rangle$$

whose component parts are as follows.

1. $CL \subseteq \mathcal{CL}$ *is a finite set of contour labels.*
2. $T \subseteq \mathcal{T}$ *is a finite set of terms.*
3. $SA \subseteq \mathcal{AR}$ *is a finite set of **solid arrows** and* $DA \subseteq \mathcal{AR}$ *is a finite set of **dashed arrows** such that each arrow* $(l, s, t) \in SA \cup DA$ *satisfies* $Com(s) \cup \{t\} \subseteq CL \cup T \cup \{\mathcal{U}\}$.
4. $Z \subseteq \{(a, b) : a \cup b = CL \wedge a \cap b = \emptyset\}$ *is a set of **zones**.*
5. $SZ \subseteq Z$ *is a set of **shaded zones**.*
6. *A function* $\lambda \colon T \to \mathbb{P}Z - \{\emptyset\}$ *returns the **location** of each term.*
7. *A relation* $\iota \subseteq T \times T$*. We say that terms* t_1 *and* t_2 *are **identified** in* d *if* $(t_1, t_2) \in \iota$ *or* $(t_2, t_1) \in \iota$.
8. *A relation* $\sigma \subseteq T \times T$*. We say that terms* t_1 *and* t_2 *are **separated** in* d *if* $(t_1, t_2) \in \sigma$ *or* $(t_2, t_1) \in \sigma$.

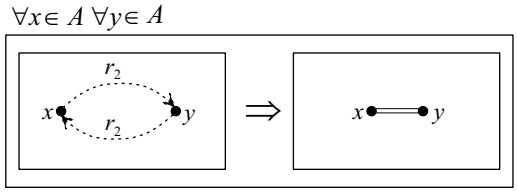

Figure 5. A compound diagram.

Unitary diagrams form the basic building blocks of *compound diagrams*.

Example 2.4 The compound diagram in figure 5 expresses that the relation r_2 is anti-symmetric when restricted to A.

To allow us to quantify over sets outside unitary diagrams, we define **set expressions**. Any contour label (given or derived) is a set expression and $\mathcal{U}$ is a set expression. If A and B are set expressions then $(A \circ B)$ is a set expression where $\circ \in \{\cup, \cap, -\}$.

Definition 2.2 *A **diagram** is defined as follows. A unitary diagram is a diagram. If d_1 and d_2 are diagrams then $\neg d_1$ and $(d_1 \circ d_2)$ where $\circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$ are diagrams. Additionally, if x_i is a variable and A is a set expression then $(\forall x_i \in A)\ d_1$ and $(\exists x_i \in A)\ d_1$ are diagrams.*

2.3. Syntax of FOPL

We briefly summarize the syntax of FOPL. The variables and terms are elements of $\mathcal{V}$ and $\mathcal{T}$ respectively. **Atomic formulae** are of two kinds. If s and t are terms then $(s = t)$ is an atomic formula. If r is a relation symbol and $t_1, ..., t_{\alpha(r)}$ are terms then $r(t_1, ..., t_{\alpha(r)})$ is an atomic formula. **Formulae** are of four kinds. Atomic formulae are formulae. If p and q are formulae then $\neg p$ and $(p \circ q)$ are a formulae where $\circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$. Additionally, if x_i is a variable then $\exists x_i\ p$ and $\forall x_i\ p$ are formulae.

3. Semantics

So far, we have given the syntax of VFOL and FOPL. We shall assume the standard semantic interpretation of FOPL formulae (see, for example, [2]). In VFOL, we briefly note that contour labels represent sets, terms represent elements (although in our formalization they represent singleton sets) and arrow labels represent relations or functions. An arrow, together with its source and target, represents a property of the relation or function represented by its label. We note that dashed arrows are syntactic sugar. This section formalizes the semantics.

Definition 3.1 *A **structure**, S, is a non-empty set U, called the **domain** of S, together with a single element subset of U for each $f \in \mathcal{F}_0$, a function $S(f)\colon U^{\alpha(f)} \to U$ for each $f \in \mathcal{F} - \mathcal{F}_0$, and a relation $S(r) \subseteq U^{\alpha(r)}$ for each $r \in \mathcal{R}$.*

Arrows give information about the image of a relation (or function) when the domain is restricted to the (set represented by) the arrows source.

Definition 3.2 *Let U denote the universal set and let f be a function. The **image** of f is the set*

$$im(f) = \{a_{\alpha(f)+1} : \exists a_1, ..., a_{\alpha(f)}\ (a_1, ..., a_{\alpha(f)+1}) \in f\}.$$

*Let A be a subset of $U^{\alpha(f)}$. We define $A.f$ to be the image of f with the domain restricted to A: $A.f = im(f|_A)$. Let r be a relation. We define the **image** of r to be*

$$im(r) = \{a_{\alpha(r)} : \exists a_1, ..., a_{\alpha(r)-1}\ (a_1, ..., a_{\alpha(r)}) \in r\}.$$

Let A be a subset of $U^{\alpha(r)-1}$. We define the image of r with the domain restricted to A to be

$$A.r = im(r \cap (A \times U)).$$

We wish to identify when a structure satisfies a VFOL diagram. In order to do so, we will interpret the component parts of the diagram, illustrated in the following example.

Example 3.1 Let S be a structure and consider the diagrams in figure 4. Some components of d_1 are interpreted in S: $S(r_1)$, $S(r_4)$, $S(f_3)$ and $S(f_2)$. We interpret $\mathcal{U}$ as the universal set: $S(\mathcal{U}) = U$. The term $f_2(x)$ is located outside both r_1 and r_4 and we associate with d_1 a *terms condition*:

$$\{x\}.S(f_2) \subseteq S(\mathcal{U}) - (S(r_1) \cup S(r_4)).$$

The solid arrow expresses

$$(S(r_1) \times S(r_4)).S(f_3) = \{x\}.S(f_2)$$

and this is called the *solid arrows condition*. The placement of r_1 and r_4 expresses that $S(r_1) \cap S(r_4) = \emptyset$. To capture this, we define the *plane tiling condition*, which asserts that the union of the sets represented by the zones is the universal set. For d_1, the plane tiling condition is:

$$(S(\mathcal{U}) - (S(r_1) \cup S(r_4))) \cup (S(r_1) \cap (S(\mathcal{U}) - S(r_4))) \cup (S(r_4) \cap (S(\mathcal{U}) - S(r_1))) = S(\mathcal{U}).$$

In d_2, the dashed arrow expresses $\{x\}.S(r_2) \supseteq S(f_1)$. The terms f_1 and $f_2(x)$ are identified, which asserts $S(f_1) = \{x\}.S(f_2)$. Separated terms denote distinct elements (strictly, singleton sets), so $\{x\} \neq S(f_4)$. Additionally, d_2 has the terms condition

$$\{x\} \subseteq S(\mathcal{U}) - S(r_1) \wedge S(f_4) \subseteq S(\mathcal{U}) - S(r_1) \wedge S(f_1) \subseteq S(\mathcal{U}) - S(r_1) \wedge \{x\}.S(f_2) \subseteq S(\mathcal{U}).$$

To facilitate the construction of a set of conditions that will allow us to determine whether a structure is a model for a diagram , we overload S. The result will include symbolic statements since, in general, the overloading contains uninterpreted variables.

Definition 3.3 *Let S be a structure with domain U. We overload S and define the following.*

1. *Universe: $S(\mathcal{U}) = U$.*
2. *Set expressions: If $A \circ B$ is a set expression where $\circ \in \{\cup, \cap, -\}$ then $S(A \circ B) = (S(A) \circ S(B))$.*
3. *Variables: for each $x_i \in \mathcal{V}$, $S(x_i) = \{x_i\}$.*
4. *Terms: for each $t \in \mathcal{T}$, if t is a constant or variable then $S(t)$ is already defined. Otherwise t is of the form $f_i(t_1, ..., t_{\alpha(f_i)})$ for some $f_i \in \mathcal{F} - \mathcal{F}_0$ and terms $t_1, ..., t_{\alpha(f_i)}$ and we define*

$$S(t) = ((S(t_1) \times ... \times S(t_{\alpha(f_i)})).S(f_i)).$$

5. *Derived contour labels: let D be a derived contour label. Then D is of the form $((D_1 \times ... \times D_n).g)$ and we define $S(D)$ recursively:*

$$S(D) = ((S(D_1) \times ... \times S(D_n)).S(g)).$$

6. *Zones: for each zone (a, b) we define*

$$S(a, b) = \bigcap_{l \in a} S(l) \cap (S(\mathcal{U}) - \bigcup_{l \in b} S(l)).$$

   We define the union (intersection) over the empty set to be the empty set (the domain).

7. *Sets of zones: for each set of zones, $\mathcal{Z}$, $S(\mathcal{Z}) = \bigcup_{(a,b) \in \mathcal{Z}} S(a, b)$.*

Definition 3.4 *Let d be a diagram. The **semantics predicate**, denoted $P_d(S)$, for d is defined as follows. If d is a unitary diagram then $P_d(S)$ is the conjunction of the following conditions.*

1. ***Terms Condition** Terms denote elements (strictly, singleton sets) in the sets represented by their locations:*

$$\bigwedge_{t \in T} S(t) \subseteq S(\lambda(t)).$$

2. ***Solid Arrows Condition** Each solid arrow expresses that, when the domain is restricted to the source, the image of the label equals the target:*

$$\bigwedge_{(l,s,t) \in SA} S(s).S(l) = S(t).$$

3. ***Dashed Arrows Condition** Each dashed arrow expresses that, when the domain is restricted to the source, the image of the label is a superset of the target:*

$$\bigwedge_{(l,s,t) \in DA} S(s).S(l) \supseteq S(t).$$

4. ***Plane Tiling Condition** The union of the sets represented by the zones is the universal set:*

$$\bigcup_{z \in Z} S(z) = S(\mathcal{U}).$$

5. ***Shading Condition** The sets represented by shaded zones contain only elements represented by terms in the diagram:*

$$\bigwedge_{z \in SZ} S(z) \subseteq \bigcup_{t \in T} S(t).$$

6. ***Equality Condition** Terms that are identified represent the same elements:*

$$\bigwedge_{(t_i,t_j) \in \iota} S(t_i) = S(t_j).$$

7. ***Distinctness Condition** Terms that are separated represent distinct elements:*

$$\bigwedge_{(t_i,t_j) \in \sigma} S(t_i) \neq S(t_j).$$

*If $d = \neg d_1$ for some d_1 then $P_d(S) = \neg P_{d_1}(S)$. If $d = (d_1 \circ d_2)$ for some d_1, d_2 and $\circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$ then $P_d(S) = (P_{d_1}(S) \circ P_{d_2}(S))$. If $d = (Qx_i \in A)\, d_1$ for some $Q \in \{\forall, \exists\}$, variable x_i and set expression A then $P_d(S) = (Qx_i \in S(A))\, P_{d_1}(S)$. If $P_d(S)$ is true then S is a **model** for d.*

4. Mapping from FOPL to VFOL

A FOPL formula and a VFOL diagram are **semantically equivalent** when they have the same models. In order to show that FOPL is at most as expressive as VFOL we will map formulae to semantically equivalent diagrams. For example, the FOPL formula, p,

$$\exists x \exists y\, \neg(x = y) \Rightarrow (r_1(x) \wedge r_4(x, y, z))$$

is semantically equivalent to the diagram in figure 6. There is an obvious mapping from the atomic parts of p to the unitary parts of the diagram.

Definition 4.1 *Define a function, $\mathcal{E}$, which maps formula p to diagram d as follows.*

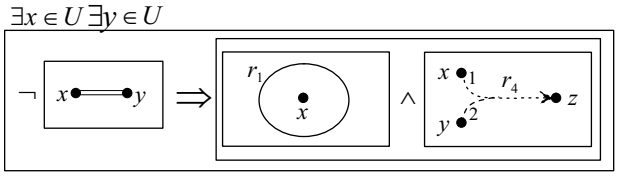

Figure 6. Mapping FOPL to VFOL.

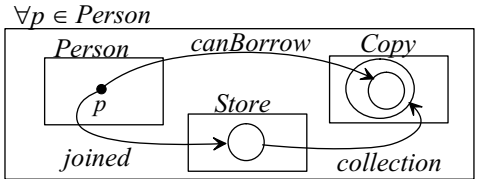

Figure 7. Specifying Software Systems.

1. p is atomic. *Then p can be one of three types and $\mathcal{E}(p) = d$ is a unitary diagram. In each case we only specify the non-empty components of d.*

   (a) p is of the form $(s = t)$ for terms s and t. *The terms are* $T = \{s, t\}$. *The zones are* $Z = \{(\emptyset, \emptyset)\}$. *The locations of the terms are* $\lambda(s) = Z$ *and* $\lambda(t) = Z$. *The terms are identified,* $\tau = \{(s, t)\}$.

   (b) p is of the form $r(t)$ for some $r \in \mathcal{R}_1$ and term t. *The labels are* $CL = \{r\}$. *The terms are* $T = \{t\}$. *The zones are* $Z = \{(\{r\}, \emptyset), (\emptyset, \{r\})\}$. *The location of the term is* $\lambda(t) = \{(\{r\}, \emptyset)\}$.

   (c) p is of the form $r(t_1, ..., t_{\alpha(r)})$ for some $r \in \mathcal{R} - \mathcal{R}_1$ and terms $t_1, ..., t_{\alpha(r)}$. *The terms are* $T = \{t_1, ..., t_{\alpha(r)}\}$. *The zones are* $Z = \{(\emptyset, \emptyset)\}$. *The dashed arrows are* $DA = \{(r, (t_1, ..., t_{\alpha(r)-1}), t_{\alpha(r)})\}$. *The locations of the terms are, for each* t_i $(1 \leq i \leq \alpha(r))$ $\lambda(t_i) = \{(\emptyset, \emptyset)\}$.

2. p is of the form $\neg q$ where q is a formula. $\mathcal{E}(p) = \neg\mathcal{E}(q)$.

3. p is of the form $(q \circ s)$ where q and s are formulae and $\circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$. $\mathcal{E}(p) = (\mathcal{E}(q) \circ \mathcal{E}(s))$.

4. p is of the form $Qx_i\ q$ where $Q \in \{\forall, \exists\}$, x_i is a variable and q is a formula. $\mathcal{E}(p) = (Qx_i \in \mathcal{U})\ \mathcal{E}(q)$.

Theorem 4.1 *Any formula f is semantically equivalent to $\mathcal{E}(f)$.*

5. Conclusion

In this paper we have introduced VFOL which is capable of expressing any FOPL formula. We anticipate that VFOL will be useful for software specification and may also be useful for teaching logic and reasoning. An invariant we may wish to write when modelling a video rental store can be seen in figure 7. The diagram asserts that people can only borrow copies from stores they have joined.

We are developing a set of sound and complete reasoning rules for VFOL. Some of these rules will transform a diagram into a semantically equivalent diagram whose unitary parts correspond to atomic formulae. We plan to write diagrammatic versions of FOPL rules to give a complete set for VFOL and also extend to a second order language.

Acknowledgment This work is supported by the UK EPSRC grant numbers GR/R63516 and GR/R63509 for the Reasoning with Diagrams project. Thanks to Chris John for comments on earlier drafts of this paper.

References

[1] J. Barwise and E. Hammer. Diagrams and the concept of logical system. In G. Allwein and J. Barwise, editors, *Logical Reasoning with Diagrams*. OUP, 1996.

[2] S. N. Burris. *Logic for Mathematics and Computer Science*. Prentice Hall, 1998.

[3] L. Choudhury and M. K. Chakraborty. On extending Venn diagrams by augmenting names of individuals. In *Proceedings of Diagrams 2004*, pages 142–146, Cambridge, UK, March 2004. Springer-Verlag.

[4] A. Fish and J. Flower. Investigating reasoning with constraint diagrams. In *Visual Languages and Formal Methods*, volume 127 of *ENTCS*, pages 63–69, Rome, Italy, 2004. Elsevier.

[5] A. Fish, J. Flower, and J. Howse. The semantics of augmented constraint diagrams. *Journal of Visual Languages and Computing*, to appear, 2005.

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

[7] J. Howse, F. Molina, J. Taylor, S. Kent, and J. Gil. Spider diagrams: A diagrammatic reasoning system. *Journal of Visual Languages and Computing*, 12(3):299–324, June 2001.

[8] J. Howse and S. Schuman. Precise visual modelling. *Journal of Software and Systems Modelling, to appear*, 2005.

[9] S. Kent. Constraint diagrams: Visualizing invariants in object oriented modelling. In *Proceedings of OOPSLA97*, pages 327–341. ACM Press, October 1997.

[10] S.-K. Kim and D. Carrington. Visualization of formal specifications. In *6th Aisa Pacific Software Engineering Conference*, pages 102–109. IEEE Computer Society Press, 1999.

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

[12] G. Stapleton, J. Howse, and J. Taylor. A constraint diagram reasoning system. In *Proceedings of International Conference on Visual Languages and Computing*, pages 263–270. Knowledge Systems Insitute, 2003.

[13] N. Swoboda and G. Allwein. Using DAG transformations to verify Euler/Venn homogeneous and Euler/Venn FOL heterogeneous rules of inference. *Journal of Software and System Modelling*, 3(2):136–149, May 2004.