Brock University

Department of Computer Science

Splitting Atoms in Relational Algebras

Prathap Siddavaatam and Michael Winter
Technical Report # CS-10-03
December 2010

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

A First-Order Calculus for Allegories

Bahar Aameri[1] and Michael Winter[2⋆]

[1] Department of Computer Science,
University of Toronto,
Toronto, Ontario, Canada, M5S 3G8
bahar@cs.toronto.edu
[2] Department of Computer Science,
Brock University,
St. Catharines, Ontario, Canada, L2S 3A1
mwinter@brocku.ca

Abstract. In this paper we a language and first-order calculus for formal reasoning about relations based on the theory of allegories. Since allegories are categories the language is typed in Church-style. We show soundness and completeness of the calculus and demonstrate its usability by presenting the RelAPS system; a proof assistant for relational categories based on the calculus presented here.

1 Introduction

Binary relations and categories of relations in particular have been widely used in mathematics and computer science for various purposes. For some very interesting applications we refer to [4, 5, 9, 15–17]. The most general and probably most influential theory is given by the notion of an allegory and its extensions due to P. Freyd and A. Scedrov [5, 9]. Further extensions of this theory including fuzzy relations have been proposed throughout the literature, e.g., [14–16, 19].

Many systems supporting theorem proving in general and for relations in particular have been developed during the past years. However, most systems show serious disadvantages if the theory of allegories is considered. In the following we want to explore some of those systems.

A typical example for a fully automated system is Prover9 [12]. Prover9 is an automated theorem prover for first order and equational logic. Therefore, the language of the system is not typed. The typing contained in the theory of allegories could be modeled by partial operations within an untyped language. However, this would produce additional proof obligations verifying that all entities are well defined. This is a serious disadvantage of the system in this context since the type information could be part of the language and checked in advance. Furthermore, the calculus used and the proofs generated are tailored for automatic proving. As a consequence they are usually very hard to read for a human being.

⋆ The author gratefully acknowledges support from the Natural Sciences and Engineering Research Council of Canada.

In another project, a semi-automated proof system has been developed for basic category-theoretic reasoning [11]. It is based on a first order sequent calculus that captures the basic properties of categories, functors and natural transformations as well as a small set of proof tactics that automate proof search in this calculus. Since it is a automated system, it has similar problems as Prover9. Typing is not part of its languages, hence, like Prover9, this would produce additional proof obligations. The system is also based on fixed theory and no additional operations or theories can be added to it. Therefore, it is not possible to work within the theory of allegories since the system only supports basic category theory.

RALF [3] was designed as a special purpose proof assistant for heterogeneous relation algebras with the goal of supporting proofs in a calculational style. RALF has a graphical user interface which represents theorems as trees, i.e., every term is displayed as a tree where the leaves are constants and variables and the nodes are the relational operations. This makes some terms hard to read. The system is based on a fixed axiomatization; the axioms of a heterogeneous relation algebra. It is not possible to work with weaker and/or stronger theories within the system. Furthermore, the system is no longer supported and there is no working version any longer available.

RALL [13] is another theorem proving system for heterogeneous relation algebra which has the ability of automatic proving for small theorems. It uses the Isabelle/HOL type system to support reasoning within abstract heterogeneous relation algebras with minimal effort. However, RALL limits itself to reasoning within representable relation algebras. The system works by translating relation-algebraic formulas into higher-order logic. As a consequence the system is incomplete. Moreover, this method cannot be applied to weaker structures like allegories. A further consequence of this method is that the proofs generated are proofs of the translated formulas within a fully automated system. One can easily imagine that they are extremely hard to read.

In this paper we present a language and first-order calculus for formal reasoning about relations. We use the theory of allegories as an underlying axiom system, i.e., a theory that is based on categories. Therefore, relations are typed and the regular operations on relations are partial, i.e., can only be applied if the relations involved have suitable type. The goal of designing the language and the calculus was to mimic human reasoning as closely as possible. The basic language was already presented and used in [18, 19] in the context of fuzzy relations.

The rules of natural deduction mimic human reasoning very closely. However, the explicit tree structure together with the practice of making assumptions and discarding them later by applying certain rules seems not to be very suitable for computer support. In particular, discarding assumptions is not a local operation; it affects whole subtrees. On the other hand, the sequent calculus keeps track of assumptions on the left-hand side of a sequent which makes every rule a local rule. The fact that the right-hand side of a sequent, i.e., the assertion or the goal of the proof, is also a list of formulas (including the empty list) does not really

model human reasoning. Therefore, we developed a calculus which is mixture of both calculi. It is based on a sequent, i.e., keeps rules local, but has exactly one formula on the right-hand side of that sequent, i.e., there is exactly one assertion or goal at any time of the proof.

The language and the calculus have been implemented in the RelAPS system. This system is an interactive proof assistant that was designed in order to work with allegories and any possible extension thereof avoiding the problems mentioned in the systems above.

The rest of the paper is organized as follows. In Section 2 we present the basics of the theory of allegories. The Sections 3 and 4 are dedicated to the formal language and the new calculus. Finally, we present a short introduction to the RelAPS system in Section 5.

2 Relational preliminaries

Throughout this paper, we use the following notation. To indicate that a morphism R of a category $\mathcal{R}$ has source A and target B we write $R : A \to B$. The collection of all morphisms $R : A \to B$ is denoted by $\mathcal{R}[A, B]$ and the composition of a morphism $R : A \to B$ followed by a morphism $S : B \to C$ by $R; S$. Last but not least, the identity morphism on A is denoted by $\mathbb{I}_A$.

We recall briefly some fundamentals on allegories [5] and relational constructions within them. For further details we refer to [5, 15, 19]. Furthermore, we assume that the reader is familiar with the basic notions from category theory. For unexplained material we refer to [2].

Definition 1. *An allegory $\mathcal{R}$ is a category satisfying the following:*

1. *For all objects A and B the collection $\mathcal{R}[A, B]$ is a meet semi-lattice, whose elements are called relations. Meet and the induced ordering are denoted by $\sqcap$ and $\sqsubseteq$, respectively.*
2. *There is a monotone operation $^\smile$ (called converse) such that for all relations $Q : A \to B$ and $R : B \to C$ the following holds: $(Q; R)^\smile = R^\smile; Q^\smile$ and $(Q^\smile)^\smile = Q$.*
3. *For all relations $Q : A \to B$, $R, S : B \to C$ we have $Q; (R \sqcap S) \sqsubseteq Q; R \sqcap Q; S$.*
4. *For all relations $Q : A \to B, R : B \to C$ and $S : A \to C$ the modular law $Q; R \sqcap S \sqsubseteq Q; (R \sqcap Q^\smile; S)$ holds.*

In this paper we only need some basic properties of relations in some examples. We have listed those properties in the following lemma:

Lemma 1. *Let $Q; Q : A \to B$ and $R, R' : B \to C$ be relations. Then we have the following:*

1. *Composition is monotonic, i.e., $Q \sqsubseteq Q'$ and $R \sqsubseteq R'$ implies $Q; R \sqsubseteq Q'; R'$.*
2. *$Q \sqsubseteq Q; Q^\smile; Q$.*
3. *$\mathbb{I}_A{}^\smile = \mathbb{I}_A$.*

A proof of the previous lemma can be found in any of [5, 9, 15–17, 19].

Besides the basic theory of allegories the following two extension are of particular interest.

Definition 2. *An allegory $\mathcal{R}$ is called distributive if:*

1. *For every pair of objects A and B the class $\mathcal{R}[A, B]$ is a distributive lattice with smallest element $\perp\!\!\!\perp_{AB}$. Union is denoted by $\sqcup$.*
2. *For all relations $Q : A \to B$, $R, S : B \to C$ we have $Q; \perp\!\!\!\perp_{BC} = \perp\!\!\!\perp_{AC}$ and $Q; (R \sqcup S) = Q; R \sqcup Q; S)$.*

A distributive allegory is called a division allegory if for all relations $R : B \to C$ and $S : A \to C$ there is a relation $S/R : A \to B$ (called the left residual of S and R) so that for all $Q : A \to B$ we have $Q; R \sqsubseteq S$ iff $Q \sqsubseteq S/R$.

3 A first-order language for allegories

The language of allegories is two-sorted. One kind of terms will denote objects, and the other kind denotes relations. In addition, the language is typed, i.e., every relational term has a source and a target. We have chosen a Church-style typing system, i.e., every relational variable, constant symbol, and function symbol has a fixed (and known) type.

Because of its differences to a regular language for first-order logic we describe the syntax of our language in the following section in detail. In particular the dependency of relational variables on object terms, and, hence, the notion of free and bounded variables, is unusual and needs a proper definition.

3.1 Syntax

In order to provide a proper language for allegories, we require a countable set of object variables V_{obj} and a countable set of object constant symbols C_{obj}. The two sets V_{obj} and C_{obj} as well as similar sets introduced later are supposed to be disjoint, i.e., $V_{obj} \cap C_{obj} = \emptyset$.

Definition 3. *The set of object terms consists of object variables and object constant symbols.*

We also require the following components:

- V_{rel} is a countable set of relational variables. Each variable r has a type $t_1 \to t_2$ where t_1 and t_2 are object terms. To indicate that the variable r has type $t_1 \to t_2$ we write $r : t_1 \to t_2$,
- C_{rel} is a countable set of relational constant symbols. Each constant symbol c has a type $t_1 \to t_2$ where t_1 and t_2 are object terms. To indicate that the constant symbol c has type $t_1 \to t_2$ we write $c : t_1 \to t_2$,

– F is a countable set of typed function symbols. Each function symbol f has a type $\{(t_1 \to s_1), ..., (t_n \to s_n)\} \to (t \to s)$ where $t_1, s_1, ..., t_n, s_n, t, s$ are object terms. To indicate that the variable f has type $\{(t_1 \to s_1), ..., (t_n \to s_n)\} \to (t \to s)$ we write $f : \{(t_1 \to s_1), ..., (t_n \to s_n)\} \to (t \to s)$.

Based on the components above we define relational terms as follows:

Definition 4. *The set of relational terms of type $s_1 \to s_2$, where s_1 and s_2 are object terms is defined recursively as follows:*

1. *If $r : s_1 \to s_2$ is a relational variable, then r is a relational term of type $s_1 \to s_2$.*
2. *If $c : s_1 \to s_2$ is a relational constant symbol, then c is a relational term of type $s_1 \to s_2$.*
3. *If s is an object term, then $\mathbb{I}_s$ is a relational term of type $s \to s$.*
4. *If t is a relational term of type $s_1 \to s_2$, then $t^\smile$ is a relational term of type $s_2 \to s_1$.*
5. *If t_1 and t_2 are relational terms of type $s_1 \to s_2$, then $t_1 \sqcap t_2$ is a relational term of type $s_1 \to s_2$.*
6. *If t_1 and t_2 are relational terms of type $s_1 \to s_2$ resp. $s_2 \to s_3$, then $t_1; t_2$ is a relational term of type $s_1 \to s_3$.*
7. *If t_1,...,t_n are relational terms of type $s_1 \to s'_1$,..., $s_n \to s'_n$ and f is a n-ary function symbol with type $f : \{(s_1 \to s'_1), ..., (s_n \to s'_n)\} \to (s \to s')$, then $f(t_1, ..., t_n)$ is a relational term of type $s \to s'$.*

In order to define formulas we need an additional component, a countable set P of typed predicate symbols. Each predicate symbol p has a type $\{(t_1 \to s_1), ..., (t_n \to s_n)\}$ where $t_1, s_1, ..., t_n, s_n$, are object terms. To indicate that the predicate symbol p has type $\{(t_1 \to s_1), ..., (t_n \to s_n)\}$ we write $p : \{(t_1 \to s_1), ..., (t_n \to s_n)\}$. Finally we can define the set of formulas.

Definition 5. *The set of formulas is defined recursively as follows:*

1. *$\bot$ is a formula.*
2. *If t_1 and t_2 are relational terms of type $s_1 \to s_2$, then $t_1 = t_2$ is a formula.*
3. *If $t_1, \ldots, t_n$ are relational terms of type $s_1 \to s'_1$,..., $s_n \to s'_n$ and p is a n-ary predicate symbol with type $\{(s_1 \to s'_1), ..., (s_n \to s'_n)\}$, then $p(t_1, \ldots, t_n)$ is a formula.*
4. *If φ_1 and φ_2 are formulas, then $\varphi_1 \wedge \varphi_2$ is a formula.*
5. *If φ_1 and φ_2 are formulas, then $\varphi_1 \vee \varphi_2$ is a formula.*
6. *If φ_1 and φ_2 are formulas, then $\varphi_1 \to \varphi_2$ is a formula.*
7. *If φ is a formula, then $\neg\varphi$ is a formula.*
8. *If φ is a formula and $r : s_1 \to s_2$ is a relation variable, then $(\forall r : s_1 \to s_2)\varphi$ is a formula.*
9. *If φ is a formula and a is an object variable, then $(\forall a)\varphi$ is a formula.*
10. *If φ is a formula and $r : s_1 \to s_2$ is a relation variable, then $(\exists r : s_1 \to s_2)\varphi$ is a formula.*
11. *If φ is a formula and a is an object variable, then $(\exists a)\varphi$ is a formula.*

In the next step, we want to introduce the concept of free variables in a formula. Obviously, there are two kinds of variables; object variables and relational variables. The set of object variables in an object term is defined as usual. Since relational variables and constants are typed with object terms, the notion of an object variable in a relational term is not standard.

Definition 6. *The set of object variables $OV(t)$ and the set of relational variables $RV(t)$ of a relational term t is defined recursively as follows:*

1. $OV(r) = OV(s_1) \cup OV(s_2)$ *for a relational variable* $r : s_1 \to s_2$,
2. $OV(c) = OV(s_1) \cup OV(s_2)$ *for a relational constant symbol* $c : s_1 \to s_2$ *in* C_{rel},
3. $OV(t^\smile) = OV(t)$,
4. $OV(t_1; t_2) = OV(t_1 \sqcap t_2) = OV(t_1) \cup OV(t_2)$,
5. $OV(f(t_1, ..., t_n)) = OV(t_1) \cup ... \cup OV(t_n)$ *for every function symbol* $f : \{(s_1 \to s'_1), ..., (s_n \to s'_n)\} \to (s \to s')$.
6. $RV(r) = \{r\}$ *for every relational variable* r,
7. $RV(c) = \emptyset$ *for every relational constant symbol* c *in* C_{rel},
8. $RV(t^\smile) = RV(t)$,
9. $RV(t_1; t_2) = RV(t_1 \sqcap t_2) = RV(t_1) \cup RV(t_2)$,
10. $RV(f(t_1, ..., t_n)) = RV(t_1) \cup ... \cup RV(t_n)$ *for every function symbol* f.

Now we can define free object and relational variables in a formula.

Definition 7. *The set of free object variables $OFV(\varphi)$ and the set of free relational variables $RFV(\varphi)$ of a formula φ is defined as follows:*

1. $OFV(\bot) = \emptyset$,
2. $OFV(t_1 = t_2) = OV(t_1) \cup OV(t_2)$,
3. $OFV(p(t_1, ..., t_n)) = OV(t_1) \cup ... \cup OV(t_n)$,
4. $OFV(\varphi_1 \otimes \varphi_2) = OFV(\varphi_1) \cup OFV(\varphi_2)$ *where* $\otimes \in \{\wedge, \vee, \to\}$,
5. $OFV(\neg\varphi) = OFV(\varphi)$,
6. $OFV((Qa)\varphi) = OFV(\varphi) \setminus \{a\}$ *where* $Q \in \{\forall, \exists\}$,
7. $OFV((Qr : s_1 \to s_2)\varphi) = OFV(\varphi) \cup OV(r)$ *where* $Q \in \{\forall, \exists\}$,
8. $RFV(\bot) = \emptyset$,
9. $RFV(t_1 = t_2) = RV(t_1) \cup RV(t_2)$,
10. $RFV(p(t_1, ..., t_n)) = RV(t_1) \cup ... \cup RV(t_n)$,
11. $RFV(\varphi_1 \otimes \varphi_2) = RFV(\varphi_1) \cup RFV(\varphi_2)$ *where* $\otimes \in \{\wedge, \vee, \to\}$,
12. $RFV(\neg\varphi) = RFV(\varphi)$,
13. $RFV((Qa)\varphi) = RFV(\varphi)$ *where* $Q \in \{\forall, \exists\}$,
14. $RFV((Qr : s_1 \to s_2)\varphi) = RFV(\varphi) \setminus \{r\}$ *where* $Q \in \{\forall, \exists\}$,

3.2 Semantics

As for the syntax the dependency of relational variables on object terms requires a careful definition of the semantics of the language. In particular, the notion of an environment, i.e., a function mapping variables to values, is not standard

since the collection of possible values for a relational variable may depend on the values of certain object variables. For example, assume a is an object variable, $r : a \to a$ is a relational variable, and that the environment maps a to the object A and r to a relation $R : A \to A$. An update of a to the object B now requires that r is mapped to a relation with source and target B since $r : a \to a$.

As a first step to a proper definition of the semantics need a universe where all syntactic entities can be interpreted by suitable values.

Definition 8. *A pre-model $\mathcal{P}$ consists of the following data:*

1. *$|\mathcal{P}|$ a non-empty allegory,*
2. *For each constant symbol $c \in C_{obj}$ a constant $c^{\mathcal{P}} \in \mathrm{Obj}_{|\mathcal{P}|}$.*

In order to define the semantics of terms and formulas we have to replace the free variables of the formula by actual values. Those values are stored in so called environments.

Definition 9. *An object environment σ_o over a pre-model $\mathcal{P}$ is a function from the set of object variables to the objects of $|\mathcal{P}|$.*

We are now ready to define the value of an object term in a pre-model.

Definition 10. *The value $\mathcal{V}_{\mathcal{P}}$ of object terms under the environment σ_o is defined by:*

- *$\mathcal{V}_{\mathcal{P}}(a)(\sigma_o) = \sigma_o(a)$ for every object variable a,*
- *$\mathcal{V}_{\mathcal{P}}(c)(\sigma_o) = c^{\mathcal{P}}$ for every constant symbol $c \in C_{obj}$.*

In the next definition we define an environment for both relational and object variables.

Definition 11. *An environment $\sigma = (\sigma_o, \sigma_r)$ over a pre-model $\mathcal{P}$ is a pair of functions so that σ_o is an object environment over $\mathcal{P}$ and σ_r maps each relational variable $r : s_1 \to s_2$ to a relation $\sigma_r(r) : \mathcal{V}_{\mathcal{P}}(s_1)(\sigma_o) \to \mathcal{V}_{\mathcal{P}}(s_2)(\sigma_o)$.*

In the following σ_o and σ_r will always refer to the object and relational part of an environment σ, respectively. Similarly, we will write $\sigma(a)$ instead of $\sigma_o(a)$ for object variables a, and $\sigma(r : s_1 \to s_2)$ instead of $\sigma_r(r : s_1 \to s_2)$ for relational variables $r : s_1 \to s_2$.

Storing a new value for a variable in an environment is called update. Such an update of an environment yields again an environment. Recall that updating an object variable may change the collection of relations that a relational variable might be mapped to.

Definition 12. *The update $\sigma[A/a]$ of σ at the object variable a with the object A is defined by:*

$$\sigma[A/a](b) = \begin{cases} \sigma(b) & \text{iff } a \neq b, \\ A & \text{iff } a = b, \end{cases}$$

$$\sigma[A/a](r : s_1 \to s_2) = \begin{cases} \sigma(r : s_1 \to s_2) & \text{iff } s_1 \neq a \text{ and } s_2 \neq a, \\ R : \sigma[A/a](s_1) \to \sigma[A/a](s_2) & \text{iff } s_1 = a \text{ or } s_2 = a \end{cases}$$

for an arbitrary relation $R : \sigma[A/a](s_1) \to \sigma[A/a](s_2)$.
The update $\sigma[R/r : s_1 \to s_2]$ *at the relation variable* $r : s_1 \to s_2$ *with the relation* $R : \sigma(s_1) \to \sigma(s_2)$ *is defined by:*

$$\sigma[R/r : s_1 \to s_2](a) = \sigma(a),$$

$$\sigma[R/r : s_1 \to s_2](q : s_1 \to s_2) = \begin{cases} \sigma(q : s_1 \to s_2) & \text{iff } r : s_1' \to s_2' \neq s_1 \to s_2, \\ R & \text{iff } r : s_1' \to s_2' = q : s_1 \to s_2. \end{cases}$$

To ascribe meaning to all formulas, we need, besides a non empty allegory, an appropriate interpretation of each of the constant, function and predicate symbols.

Definition 13. *A relational model* $\mathcal{M}$ *is a pre-model with the following data:*

1. *For each* $c : s_1 \to s_2$ *in* C_{rel} *and environment* σ *a constant* $c_\sigma^{\mathcal{M}} : \sigma(s_1) \to \sigma(s_2)$ *so that* $\sigma(s_1) = \sigma'(s_1)$ *and* $\sigma(s_2) = \sigma'(s_2)$ *implies* $c_\sigma^{\mathcal{M}} = c_{\sigma'}^{\mathcal{M}}$,
2. *For each function symbol* $f : \{(t_1 \to s_1), ..., (t_n \to s_n)\} \to (t \to s)$ *in* F *and environment* σ, *a n-ary function* $f_\sigma^{\mathcal{M}}$ *which is mapping* $|\mathcal{M}|[\sigma(t_1), \sigma(s_1)] \times ... \times |\mathcal{M}|[\sigma(t_n), \sigma(s_n)]$ *to* $|\mathcal{M}|[\sigma(t), \sigma(s)]$ *so that* $\sigma(t_1) = \sigma'(t_1), ..., \sigma(t_n) = \sigma'(t_n), \sigma(t) = \sigma'(t)$ *and* $\sigma(s_1) = \sigma'(s_1), ..., \sigma(s_n) = \sigma'(s_n), \sigma(s) = \sigma'(s)$ *implies* $f_\sigma^{\mathcal{M}} = f_{\sigma'}^{\mathcal{M}}$,
3. *For each predicate symbol* p *in* P *with type* $(t_1 \to s_1), ..., (t_n \to s_n)$ *and environment* σ, *a subset* $p_\sigma^{\mathcal{M}} \subseteq \{|\mathcal{M}|[\sigma(t_1), \sigma(s_1)] \times ... \times |\mathcal{M}|[\sigma(t_n), \sigma(s_n)]\}$ *so that* $\sigma(t_1) = \sigma'(t_1), ..., \sigma(t_n) = \sigma'(t_n), \sigma(t) = \sigma'(t)$ *and* $\sigma(s_1) = \sigma'(s_1), ..., \sigma(s_n) = \sigma'(s_n), \sigma(s) = \sigma'(s)$ *implies* $p_\sigma^{\mathcal{M}} = p_{\sigma'}^{\mathcal{M}}$.

In the previous definition, an object environment would be sufficient because it is just needed to get the value of an object term. Note that the value of an object term in a model $\mathcal{M}$ is the same as that in the pre-model $\mathcal{P}$ it contains, i.e., $\mathcal{V}_{\mathcal{M}}(s)(\sigma) = \mathcal{V}_{\mathcal{P}}(s)(\sigma_o)$.

Now we are ready to define the value of relational terms and the validity of formulas. Both definitions are done inductively on the structure of the language.

Definition 14. *Let* $\mathcal{M}$ *be a relational model and* σ *be an environment. The value* $\mathcal{V}_{\mathcal{M}}$ *of terms under the environment* σ *is defined by:*

1. $\mathcal{V}_{\mathcal{M}}(r : s_1 \to s_2)(\sigma) = \sigma(r : s_1 \to s_2)$ *for every relational variable* $r : s_1 \to s_2$,
2. $\mathcal{V}_{\mathcal{M}}(c : s_1 \to s_2)(\sigma) = c_\sigma^{\mathcal{M}}$ *for every constant* $c : s_1 \to s_2$ *in* C_{rel},
3. $\mathcal{V}_{\mathcal{M}}(f(t_1, ..., t_n))(\sigma) = f_\sigma^{\mathcal{M}}(\mathcal{V}_{\mathcal{M}}(t_1)(\sigma), ..., \mathcal{V}_{\mathcal{M}}(t_n)(\sigma))$
4. $\mathcal{V}_{\mathcal{M}}(\mathbb{I}_a)(\sigma) = \mathbb{I}_{\sigma(a)}$,
5. $\mathcal{V}_{\mathcal{M}}(t^\smile)(\sigma) = (\mathcal{V}_{\mathcal{M}}(t)(\sigma))^\smile$,
6. $\mathcal{V}_{\mathcal{M}}(t_1 \sqcap t_2)(\sigma) = \mathcal{V}_{\mathcal{M}}(t_1)(\sigma) \sqcap \mathcal{V}_{\mathcal{M}}(t_2)(\sigma)$,
7. $\mathcal{V}_{\mathcal{M}}(t_1; t_2)(\sigma) = \mathcal{V}_{\mathcal{M}}(t_1)(\sigma); \mathcal{V}_{\mathcal{M}}(t_2)(\sigma)$.

The next step is to define the validity of formulas.

Definition 15. *Let* $\mathcal{M}$ *be a relational model, and* σ *be an environment. The validity of a formula in* $\mathcal{M}$ *under* σ *is defined inductively as follows:*

1. $\mathcal{M} \models_\sigma t_1 = t_2$ *iff* $\mathcal{V}_\mathcal{M}(t_1)(\sigma) = \mathcal{V}_\mathcal{M}(t_2)(\sigma)$,
2. $\mathcal{M} \models_\sigma p(t_1, \ldots, t_n)$ *iff* $(\mathcal{V}_\mathcal{M}(t_1)(\sigma), \ldots, \mathcal{V}_\mathcal{M}(t_n)(\sigma)) \in p^\mathcal{M}_\sigma$,
3. $\mathcal{M} \models_\sigma \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{M} \models_\sigma \varphi_1$ *and* $\mathcal{M} \models_\sigma \varphi_2$,
4. $\mathcal{M} \models_\sigma \varphi_1 \vee \varphi_2$ *iff* $\mathcal{M} \models_\sigma \varphi_1$ *or* $\mathcal{M} \models_\sigma \varphi_2$,
5. $\mathcal{M} \models_\sigma \varphi_1 \to \varphi_2$ *iff* $\mathcal{M} \models_\sigma \neg\varphi_1$ *or* $\mathcal{M} \models_\sigma \varphi_2$,
6. $\mathcal{M} \models_\sigma \neg\varphi$ *iff* $\mathcal{M} \not\models_\sigma \varphi$
7. $\mathcal{M} \models_\sigma (\forall r : s_1 \to s_2)\varphi$ *iff* $\mathcal{M} \models_{\sigma[R/r:s_1\to s_2]} \varphi$ *for all relations* $R : \sigma(s_1) \to \sigma(s_2)$,
8. $\mathcal{M} \models_\sigma (\forall a)\varphi$ *iff* $\mathcal{M} \models_{\sigma[A/a]} \varphi$ *for all objects* A,
9. $\mathcal{M} \models_\sigma (\exists r : s_1 \to s_2)\varphi$ *iff* $\mathcal{M} \models_{\sigma[R/r:s_1\to s_2]} \varphi$ *for some relation* $R : \sigma(s_1) \to \sigma(s_2)$,
10. $\mathcal{M} \models_\sigma (\exists a)\varphi$ *iff* $\mathcal{M} \models_{\sigma[A/a]} \varphi$ *for some object* A.

As usual we write $\mathcal{M} \models \varphi$ *if* $\mathcal{M} \models_\sigma \varphi$ *holds for all environment* σ *and* $\models \varphi$ *if* $\mathcal{M} \models \varphi$ *holds for all relational models* $\mathcal{M}$.

For our language versions of a coincidence and a substitution lemma hold.

Lemma 2. *Let* a *be an object variable,* $r : s_1 \to s_2$ *be a relational variable,* s, s' *be object terms,* t, t' *be relational terms,* φ *be a formula, and* $\mathcal{M}$ *be a relational model. Furthermore, suppose* σ_1 *and* σ_2 *are environments over* $\mathcal{M}$ *so that* $\sigma_1(a) = \sigma_2(a)$ *for all free object variables in* s, t *or* φ *and* $\sigma_1(r : s_1 \to s_2) = \sigma_2(r : s_1 \to s_2)$ *for all free relational variables* r *in* t *or* φ, *respectively. Then we have the following:*

1. $\mathcal{V}_\mathcal{M}(s)(\sigma_1) = \mathcal{V}_\mathcal{M}(s)(\sigma_2)$
2. $\mathcal{V}_\mathcal{M}(t)(\sigma_1) = \mathcal{V}_\mathcal{M}(t)(\sigma_2)$.
3. $\mathcal{M} \models_{\sigma_1} \psi$ *iff* $\mathcal{M} \models_{\sigma_2} \psi$.
4. $\mathcal{V}_\mathcal{M}(s'[s/a])(\sigma) = \mathcal{V}_\mathcal{M}(s')(\sigma[\mathcal{V}_\mathcal{M}(s)(\sigma)/a])$.
5. $\mathcal{V}_\mathcal{M}(t'[t/r])(\sigma) = \mathcal{V}_\mathcal{M}(t')(\sigma[\mathcal{V}_\mathcal{M}(t)(\sigma)/r])$.
6. $\mathcal{V}_\mathcal{M}(t'[s/a])(\sigma) = \mathcal{V}_\mathcal{M}(t')(\sigma[\mathcal{V}_\mathcal{M}(s)(\sigma)/a])$.
7. $\mathcal{M} \models_\sigma \varphi[t/r]$ *iff* $\mathcal{M} \models_{\sigma[\mathcal{V}_\mathcal{M}(t)(\sigma)/r]} \varphi$.
8. $\mathcal{M} \models_\sigma \varphi[s/a]$ *iff* $\mathcal{M} \models_{\sigma[\mathcal{V}_\mathcal{M}(s)(\sigma)/a]} \varphi$.

For a proof of the lemma above we refer to [1].

4 A first-order calculus

In this section we introduce our first-order logic calculus for relational categories. The calculus is formulated in a sequent style [6] but with exactly one formula on the right-hand side. It has three different types of rules; axioms, which represent the basic tautologies of logic and the axioms of the theory of allegories, structural rules, which operate on the sequent of formula in a judgment, and logical rules, which are concerned with the logical operations.

Definition 16. *The axioms in Figures 1 and the rules of Figure 2 and 3 constitute the formal calculus of allegories. If* Γ *is a sequence of formulas and* φ *is a formula, then we write* $\Gamma \vdash \varphi$ *to indicate that there is a derivation in the calculus ending in that sequence.*

$$\varphi \vdash \varphi \quad \text{(Axtiom)}$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)\mathbb{I}_a; r = r$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b) r; \mathbb{I}_b = r$$

$$\vdash (\forall a_1)(\forall a_2)(\forall a_3)(\forall a_4)(\forall r : a_1 \to a_2)(\forall q : a_2 \to a_3)(\forall u : a_3 \to a_4)(r; q); u = r; (q; u)$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b) r \sqcap r = r$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)(\forall q : a \to b)(\forall u : a \to b)(r \sqcap q) \sqcap u = r \sqcap (q \sqcap u)$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)(r^\smile)^\smile = r$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)(\forall q : a \to b)(r \sqcap q)^\smile = r^\smile \sqcap q^\smile$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)(\forall q : b \to c)(r; q)^\smile = q^\smile; r^\smile$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)(\forall q : b \to c)(\forall u : b \to c) r; (q \sqcap u) = r; (q \sqcap u) \sqcap r; q \sqcap r; u$$

$$\vdash (\forall a)(\forall b)(\forall r : a \to b)(\forall q : b \to c)(\forall u : a \to c) r; q \sqcap u = r; (q \sqcap r^\smile; u) \sqcap r; q \sqcap u$$

Fig. 1. Axioms

$$\text{Weakening rule} \qquad \frac{\Gamma \vdash \psi}{\Gamma, \varphi \vdash \psi}\ \text{Weak}$$

$$\text{Contraction rule} \qquad \frac{\Gamma, \varphi, \varphi \vdash \psi}{\Gamma, \varphi \vdash \psi}\ \text{Cont.}$$

$$\text{Permutation rule} \qquad \frac{\Gamma, \varphi_2, \varphi_1 \vdash \psi}{\Gamma, \varphi_1, \varphi_2 \vdash \psi}\ \text{Perm.}$$

$$\text{Cut rule} \qquad \frac{\Gamma \vdash \varphi \quad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}\ \text{Cut}$$

Fig. 2. Structural rules

left logical rules	right logical rules
$\dfrac{\Gamma \vdash t_1 = t_2 \quad \Gamma \vdash \psi[t_1/r]}{\Gamma \vdash \psi[t_2/r]}$ =L	$\dfrac{}{\vdash t = t}$ =R
$\dfrac{\Gamma, \varphi_1, \varphi_2 \vdash \psi}{\Gamma, \varphi_1 \wedge \varphi_2 \vdash \psi}$ $\wedge$L	$\dfrac{\Gamma \vdash \varphi_1 \quad \Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \wedge \varphi_2}$ $\wedge$R
$\dfrac{\Gamma, \varphi_1 \vdash \psi \quad \Gamma, \varphi_2 \vdash \psi}{\Gamma, \varphi_1 \vee \varphi_2 \vdash \psi}$ $\vee$L	$\dfrac{\Gamma \vdash \varphi_1}{\Gamma \vdash \varphi_1 \vee \varphi_2}$ $\vee$R $\quad$ $\dfrac{\Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \vee \varphi_2}$ $\vee$R
$\dfrac{\Gamma \vdash \varphi_1 \quad \Gamma, \varphi_2 \vdash \psi}{\Gamma, \varphi_1 \to \varphi_2 \vdash \psi}$ $\to$L	$\dfrac{\Gamma, \varphi_1 \vdash \varphi_2}{\Gamma \vdash \varphi_1 \to \varphi_2}$ $\to$R
$\dfrac{\Gamma \vdash \varphi}{\Gamma, \neg\varphi \vdash \psi}$ $\neg$L	$\dfrac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}$ $\neg$R
$\dfrac{\Gamma, \varphi[t/r] \vdash \psi}{\Gamma, (\forall r : s_1 \to s_2)\varphi \vdash \psi}$ $\forall$L (rel)	$\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash (\forall r : s_1 \to s_2)\varphi}$ $\forall$R (rel) If r does not occur free in any formula of Γ.
$\dfrac{\Gamma, \varphi[s/a] \vdash \psi}{\Gamma, (\forall a)\varphi \vdash \psi}$ $\forall$L (obj)	$\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash (\forall a)\varphi}$ $\forall$R (obj) If a does not occur free in any formula of Γ.
$\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma, (\exists r : s_1 \to s_2)\varphi \vdash \psi}$ $\exists$L (rel) If r does not occur free in any formula of Γ and in ψ.	$\dfrac{\Gamma \vdash \varphi[t/r]}{\Gamma \vdash (\exists r : s_1 \to s_2)\varphi}$ $\exists$R (rel)
$\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma, (\exists a)\varphi \vdash \psi}$ $\exists$L (obj) If a does not occur free in any formula of Γ and in ψ.	$\dfrac{\Gamma \vdash \varphi[s/a]}{\Gamma \vdash (\exists a)\varphi}$ $\exists$R (obj)

$$\frac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}\ \text{PBC}$$

Fig. 3. Logical rules

Note that for the =L rule in the form presented here (and implemented in the system) is not a really left rule since the equation appears on the right-hand side of $\vdash$. The rule can alternatively be formulated as

$$\frac{\Gamma, t_1 = t_2 \vdash \psi[t_1/r]}{\Gamma \vdash \psi[t_2/r]} \; =\mathrm{L}'$$

having the equation on the left-hand side. We have chosen the version presented in Figure 3 because that version seems more convenient to use. In particular this rule models the way equational reasoning is implemented using the 'Working Area' in RelAPS (see Section 5).

Theorem 1 (Soundness and Completeness). *The calculus is sound and complete, i.e.,* $\vdash \varphi$ *iff* $\models \varphi$ *for all formulas* φ*.*

The previous theorem was shown in [1]. The soundness proof uses a straightforward induction on the structure of the derivation. The completeness result was obtained following Henkin's method, i.e., it was actually shown that every consistent theory in our language has a model in the sense of Definition 13. Significant modifications to the original proof had to be made in order to cope with the categorical structure of the models.

5 The system RelAPS

The purpose of the RelAPS system is to provide an environment in which a user may construct a proof of certain theorems as close to a hand-written proof as possible. This provides the benefits of having a system ensuring that an individual proof-step is executed properly, while it remains the responsibility of the user to complete the proof. It should be mentioned that it is not the aim of the system to provide automated deduction.

In order to achieve the goal mentioned above the following design decisions have been made:

1. Allegories are only the beginning of a whole hierarchy of relational categories. For an overview we refer to [10]. Therefore, the system was designed to handle extensions of the theory of allegories such as distributive allegories, division allegories and so on.
2. A lot of proofs in the theory of relations are either based on equational and/or inclusion based reasoning or on a chain of equivalences. For example, in order to proof that a partial identity is idempotent, i.e., $Q \sqsubseteq \mathbb{I}$ implies $Q = Q;Q$, one could argue as follows:

$$\begin{aligned} Q &\sqsubseteq Q;Q^{\smile};Q && \text{Lemma 1(2)} \\ &\sqsubseteq Q;\mathbb{I}^{\smile};Q && \text{assumption} \\ &= Q;\mathbb{I};Q && \text{Lemma 1(3)} \\ &= Q;Q, && \text{identity axiom} \\ \text{and } Q;Q &\sqsubseteq Q;\mathbb{I} && \text{assumption} \\ &= Q. && \text{identity axiom} \end{aligned}$$

Another example is the following proof of $(Q/R)/S = Q/(S;R)$ for suitable relations in a division allegory:

$$\begin{aligned} X \sqsubseteq (Q/R)/S &\iff X;S \sqsubseteq Q/R && \text{residual axiom} \\ &\iff X;S;R \sqsubseteq Q && \text{residual axiom} \\ &\iff X \sqsubseteq Q/(S;R). && \text{residual axiom} \end{aligned}$$

In both examples we use a chain of inclusion or equivalences, and in each step we apply an axiom, an assumption or a lemma. The RelAPS system has a special 'working area' to perform proofs in that style.

3. The previous examples also show that humans usually use certain properties of operations and predicates without mentioning. Composition is associative which is used in the second proof. Both, composition and converse, are monotonic which is used in the first proof. Similar examples can be constructed in which the symmetry or an anti-monotonic behavior is used implicitly. The RelAPS system is capable of handling this kind of reasoning.
4. As already mentioned in the introduction we have chosen a logic calculus that mimics human reasoning closely. The proofs is constructed bottom-up and it is complete when all its subtrees have been ended by the application of axioms.

5.1 A short tour through the system

Upon starting RelAPS a dialog requires the user to specify which theory should be loaded. When the program is started for the first time, the only option presented is the (default) theory of allegories. Later on, user defined theories together with their set of operations and constants will also be available. After selecting an appropriate theory the user may access the RelAPS interface, which is shown in Figure 4.

The 'Assertions' (or goal) window displays the assertion of the current proof corresponding to the right-hand side of $\vdash$ in a derivation. The text area of the 'Assertions' window simply displays the current state of the assertion being worked with. The user may only work with one assertion at a time.

The 'Assumptions' window displays the assumptions that are associated with the current proof. This corresponds with the sequence Γ on the left-hand side of $\vdash$. The text area of the 'Assumptions' window allows to select any assumption in order to apply logical rules. Therefore, the order of the formulas is not important, i.e., the Permutation rule is implemented implicitly. The 'Weakening' button corresponds to the Weakening rule and removes a selected assumption from the current proof. The 'Duplicate' button implements the Contraction rule by duplicating a selected assumption.

The buttons on the right side of 'Working Area' are used to apply the logical rules, introduced in Section 4. All derivation buttons are disabled by default except PBC and Cut since the corresponding rules can always be applied. The right-hand rule buttons are enabled based on current formula in the 'Assumption' window. An appropriate left-hand rule button will be enabled when the user

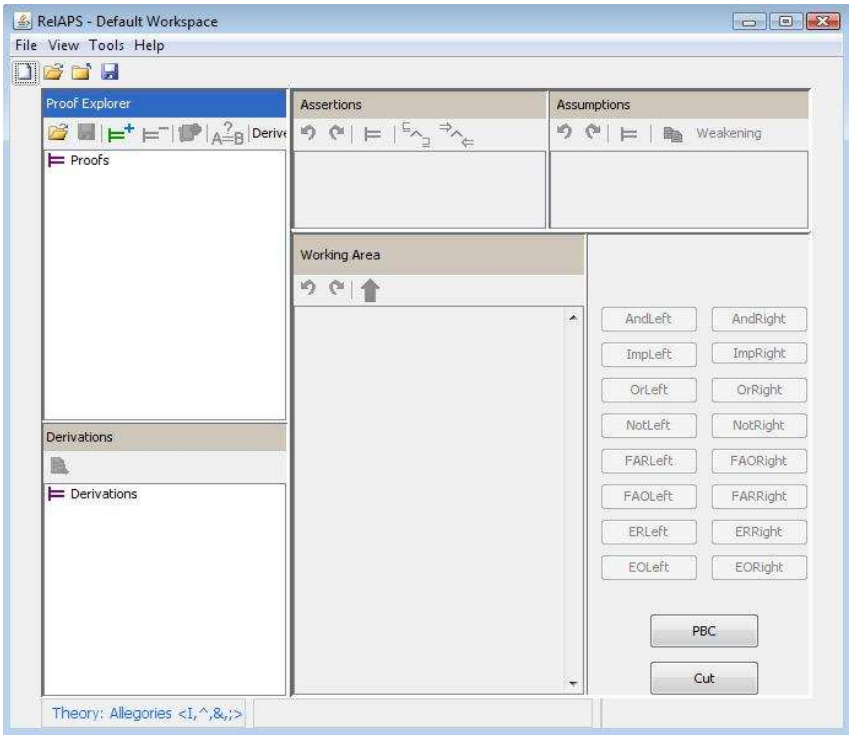

Fig. 4. The RelAPS Interface

selects an assumption in the 'Assumptions' window. There are no button for the (=L) and (=R) rules. These rules can be applied by using the 'Working Area' window described below.

When the user selects a term of either an equation or inclusion in the 'Assertions' or 'Assumption' window, the corresponding 'Derive' button ($\models$) will become enabled. This allows the user to move the selection to the 'Working Area' in order to use equation or inclusion based reasoning. A proof in the 'Working Area' is based on the equational rules (=L) and (=R). If a subterm of the current term is selected, a menu immediately pops up allowing the user to apply an axiom, an assumption, or a previously proved theorem to the current selection. By pressing the Apply button ($\Uparrow$) the original term in the 'Assertions' or 'Assumption' window will be replaced by latest term in the 'Working Area'. Instead of selecting a term, the user can also select an equation or inclusion and move the whole formula to the 'Working Area' in order to start a chain of equivalences. Notice that in the 'Working Area' certain properties of operations such as monotonicity, associativity, and symmetry are applied automatically. The system keeps track of those properties for each of the operations defined.

Each time a rule is applied the Axiom rule is automatically checked by the system. In that step the system actually checks whether the assertion is among

the formulas in the assumption window, i.e., the Weakening rule is implicitly used in this process.

There are two additional buttons in the 'Assertion' window allowing the user to split an equation into two inclusions, and to split an equivalence into two implications.

In the 'Proof Explorer' window the tree of the current proof is displayed. The user can switch between the different subtrees by clicking the appropriate assertion in that tree view. Subtrees that have already successfully be shown are tagged by 'thumb up', and all other subtrees are tagged by 'thumb down'. This window also contains buttons to save and load proofs and to enter a new theorem.

A user may define a new theory. In order to do so the user has to define new operations and axioms, which can be done by selecting the appropriate function of the 'Tools' menu. During this process the user may also specify certain properties of operations such as monotonicity, associativity, and symmetry. Once specified the system requires the property of the operation either as an axiom or that it is shown as a theorem. Afterwards the property can be used in the 'Working Area' automatically as described above. A new theory can be saved and used later. In particular, it will be available as a selection in the staring dialog.

6 Conclusion and future work

There is plenty of further work that could improve the RelAPS system. In this section we want to sketch four of those projects.

The main focus in the future should be on automating some proof steps, particularly very basic steps. Certain sub-theories of allegories, such as the equational theory, are known to be decidable. Once it has been implemented, the system could suggest to the user that the current proof obligation is in a certain sub-theory that can be decided. If the user chooses to let the system finish the proof, the corresponding algorithm is used to find that proof.

More flexible user-defined operations is another way to extend the system. Currently it is not possible to define functions that take objects as parameters and return objects. The user only can define functions that return relations. In addition, function symbols could take a mixture of relational and object terms as parameters which would be useful for relational constructions such as splittings. Such an extension would also require a modification of the language studied in this paper.

Even though user defined predicate symbols are already part of the language, they have not been implemented in the RelAPS system, i.e., the only defined predicate symbols are '$<$', '$>$', and '$=$'. Here the same flexibility as outlined in the previous paragraph for function symbols might be useful.

Producing LaTeX output of the proofs is another possible project. A researcher could use the system to prove theorems which are then automatically

checked for correctness and use the LaTeX output for publications. For any textual representation of a RelAPS proof it would be very helpful if the user could specify the level of detail in which the proof should be presented. Depending on that level each step in the textual representation could summarize several steps in the actual derivation.

References

1. Aameri B.: Extending RelAPS to First-Order Logic. MSc. Thesis, Brock University (2010).
2. Asperti A., Longo G.: Categories, Types, and Structures. Foundation of Computing Series MIT Press (1991).
3. Berghammer R., Hattensperger C., Schmidt G.: RALF:A Relation Algebraic Formula manipulation system and proof checker. *In:* Nivat M., Rattray C., Rus T., Scollo G. (Eds.), Proc. 3rd Conference on Algebraic Methodology and Software Technology, Workshops in Computing, Springer Verlag, 407-408 (1994).
4. Bird R., de Moor O.: Algebra of Programming. Prentice-Hall (1997).
5. Freyd P., Scedrov A.: Categories, Allegories. North-Holland (1990).
6. Gentzen G.: Untersuchungen über das logische Schließen I. Mathematische Zeitschrift 39 (2), 176-210 (1934).
7. Glanfield J.: RelAPS: A Proof System for Relational Categories. Presented at RelMICS/AKA: Relations and Kleene Algebra in Computer Science. Manchester, UK. August (2006).
8. Glanfield J.: Towards Automated Derivation in the Theory of Allegories. MSc. Thesis, Brock University (2008).
9. Johnstone P.T.: Sketches of an Elephant, A Topos Theory Compendium, Vol. 1. Oxford Logic Guides **42** (2002).
10. Kahl W.: Refactoring Heterogeneous Relation Algebras around Ordered Categories and Converse. J. Rel. Meth. Comp. Sci. (JoRMiCS) **1** (2004), 277–313.
11. Kozen D., Kreitz C., Richter E.: Automating Proofs in Category Theory. IJCAR 2006, Springer Verlag, 392-407 (2006).
12. McCune W.: Prover9: Automated Theorem Prover for First Order and Equational Logic. http://www.cs.unm.edu/∼mccune/prover9.
13. Oheim D.V., Gritzner T.F.: RALL: Machine Supported Proofs for Relation Algebra. *In:* W. McCune, ed., Conference on Automated Deduction, CADE 14, LNCS 1249, Springer, 380-394 (1997).
14. Olivier J.P., Sarrato D.: Catégories de Dedekind. Morphismes dans les categories de Schröder. C. R. Acad. Sci. Paris **290** (1980), 939–941.
15. Schmidt G., Ströhlein T.: Relationen und Graphen. Springer (1989); English version: Relations and Graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993).
16. Schmidt G., Hattensperger C., Winter M.: Heterogeneous Relation Algebras. *In:* Brink C., Kahl W., Schmidt G. (eds.), Relational Methods in Computer Science, Advances in Computing Science, Springer Vienna (1997).
17. Schmidt G.: Relational Mathematics. Cambridge University Press (2010).
18. Winter M.: A new algebraic approach to L-fuzzy relations convenient to study crispness. Information Science **139** (2001), 233–252.
19. Winter M.: Goguen categories. A categorical approach to L-fuzzy relations. Trend in Logic Vol. 25, Springer (2007).