A Relational Approach to Bidirectional Transformations

Nuno Macedo

Thesis Planning
MAP-i Doctoral Program
University of Minho

Supervisor: Alcino Cunha

August 31, 2011

Contents

Chapter 1

Introduction

Transforming data between different structures is a typical computing problem. Furthermore, in most cases we wish these transformations to be *bidirectional*, with the changes made in either structure reflected in the other. A naive approach to solve this problem involves the creation of two unidirectional transformations and then verifying that they are in some way each-other inverses (or *well-behaved*). Besides being an expensive and error-prone task, this verification would need to be performed every-time the transformation is modified. A better approach consists in designing a language where an expression represents both transformations, which are guaranteed to be *correct by construction*. Recently, bidirectional transformations under this perspective have become increasingly popular, with applications in synchronization of data in heterogeneous formats [FGM+07, BFP+08, BMS08], relational databases [BPV06], model transformation [Ste10, HHKN09], graph transformation [HHI+10, EEE+07] and programming languages [MHN+07, BCF+05, PC10, Voi09].

A particularly successful mechanism are the *lenses* [FGM+07], initially proposed by the Harmony group as a solution to the *view-update problem* of databases (how to propagate updates on a database view back to the source [BS81]). In this combinatorial approach, transformations comprise two operations: a forward transformation *get* that abstracts the source, and a backward transformation *put* that propagates changes on views back to the sources. Since views are assumed to contain less information than the sources, the backward transformation also takes the original source as input in order to recover lost information.

The *point-free* (PF) notation, a style where there are no variables (nor quantifiers), as opposed to the point-wise (PW) notation, provides a simpler framework where proofs can be carried out by simple equational steps. Although a PF version of the calculus of relations has long been axiomatized by Alfred Tarski in 1941 [TG87], it was only as expressive as a fragment of first-order logic with three variables. It was not until Freyd and Ščedrov defined the notion of an *allegory* [Fv90], the categorical counter-part of relation algebras (RAs), that the bases for the representation of programs in a relational setting were laid down. Eventually, a categorical algebra of programming was defined [BdM96, Bac04], evolving from the PF equational calculus originated by John Backus [Bac78], which had already been widely and successfully used in program construction and verification [CPP06]. This calculus has since been successfully applied by José N. Oliveira to a variety of areas, like data refinement [Oli08, OCV06],

extended static checking [Oli09] and operation refinement [OR06].

Recent work has been developed to specify lenses using functional PF notation [PC10, PC11], lifting standard PF combinators and recursion patterns to well-behaved lenses. This approach enables easy equational calculation with lenses and optimization of bidirectional transformations. However it has some limitations, namely it is not expressive enough to capture all sorts of bidirectional transformations. By generalizing to a relational setting we obtain a higher degree of freedom in the calculus, becoming easier to calculate inverses of transformations and enabling the representation of data invariants. This thesis intends precisely to explore this generalization, with the following particular goals and schedule in mind:

- To compare and classify the existing approaches to bidirectional transformations using RAs as the unifying formalism.
- To enlarge the applicability of the previously proposed PF lens [PC10], by proposing new combinators and using data invariants to enlarge the expressiveness of the language.
- To propose a methodology for the derivation of backward transformations by calculation, starting from the specification of the forward transformation and the desired well-behavedness criterion.
- To propose an effective technique for bidirectional transformation of non tree-like data structures (graphs and models in general). Some techniques have already been proposed, but they are either too ad-hoc, giving little or no guarantees to the user, or too complex and limited, making it difficult to express transformations or perform calculations / optimizations.
- To identify and give proper semantics to well-behaved subsets of existing high-level model transformation languages (namely QVT [Obj08]) using RAs.
- Implementation of the proposed framework.

The objective of the pre-thesis is to provide a deeper understanding of the subject of the thesis, and subsequently assess the viability of our relational approach and the proposed objectives. In particular, the plan of the pre-thesis consists of:

1. Survey the state-of-the-art of bidirectional transformation techniques.

   Bidirectional transformations is currently a hot topic in software engineering, with a wide and quickly increasing body of knowledge. Since they are the main subject of the thesis, the first step should be to study and write a comprehensive survey on the existing approaches and techniques.

2. Create a taxonomy to classify and compare the different approaches.

   The existing approaches to bidirectional transformations vary significantly in the underlying techniques, target artifacts, expressibility, semantics, and formal guarantees provided to the user. Since we are proposing RAs as a unifying formalism, it

is fundamental to first define a set of criteria to compare and classify them. As such, one of the objectives of the thesis planning is to define a formal taxonomy, under which the surveyed techniques should be subsequently classified.

3. Assess the viability of a relational approach to bidirectional transformations.

   In order to better understand exactly how the relational framework can be used to specify and reason about bidirectional transformations, a few classical examples should be restated following this approach. Moreover, in order to assess the viability of the goals, it was considered important to research how inverses could be derived by calculation, and how data invariants could be incorporated in the *lens* framework.

4. Detail the objectives of the thesis and the methodology to prosecute them.

   Finally, after surveying the current status of the subject area and assessing the viability of the relational approach, we will be able to better define the objectives of the thesis, and the methodology to achieve and validate them.

The structure of this document follows closely this plan. In Chapter 2 the basic concepts about RAs are presented. Chapter 3 presents most of the existing approaches on bidirectional transformations, starting by defining a taxonomy and classification system under the relational setting, and then moving on to analyze and classify the state-of-the-art in the area. Chapter 4 presents some applications of the relational approach, namely a redefinition of an existing framework and the derivation of backward transformations. In Chapter 5 a new framework for relational lenses, where the states are enhanced with invariants, is proposed. Lastly, in Chapter 6 the results already achieved and the contributions expected from the thesis are discussed.

Chapter 2

Relation Algebras

2.1 Introduction

Relations provide the most natural way to define a correspondence between two entities. Most relationships in real life situations are not necessarily total, deterministic or one-to-one. The same applies to program construction. Although functional calculus is usually used in program specification, the generalization from functions to relations provides many benefits. Most so called functions in functional programming are partial, so they are actually better represented by relations. Also, since relations are essentially non-deterministic they provide a natural way to represent non-deterministic problems. Finally, in program specification, the properties about the models are most of the times relations between states or objects.

Another advantage of the relational framework is its "calculational" power. Formula manipulation benefits from a higher level of freedom once the restriction to functions is lifted, since, for instance, all relations will have a well defined converse. If we then consider a PF characterization of relational logic, to those advantages we add those of the PF reasoning: simplicity and ease of manipulation.

In this report the classical definition of RAs will be presented, as well as its extension in the categorical framework, needed to overcome the expressiveness limitations of RAs. We begin in Section 2.2 by briefly presenting the origins of the calculus of relations and their evolution until the now commonly used RAs. In Section 2.3 we present the basic notions of category theory, needed to the definition of allegories in Section 2.4, a particular class of categories that extend RAs.

2.2 Relation Algebras

The concept of (binary) relations was first introduced by Augustus De Morgan [Mor60] in the 19th century, which he defined as:

> "Let $X..LY$ signify that X is some one of the objects of thought which stand to Y in the relation L, or is one of the Ls of Y."

Morgan emphasized the importance of relations, studied their properties and defined various operations on them. Charles Sanders Peirce [Pei33] casted that theory on the form of a calculus, based on the calculus of classes already developed by Boole. The last massive contribution to the calculus of relations was due to Ernst Schröder [Sch95], by systematically analyzing and summarizing the previous studies. From these works, the algebra of binary relations as it is known nowadays eventually emerged. Interestingly, first-order logic also arose from these works as a way to explain the calculus, eventually overcoming its popularity. For an in-depth presentation of the origin and evolution of the calculus of binary relations, see [Mad91, Mad06].

Definitions For a fixed non-empty set A, the universe of discourse, a binary relation R is a set of ordered pairs on A, or a subset of the cartesian product $A \times A$. Although the defined operations and distinguished relations vary, a standard definition of the algebra of binary relations follows. Note that it takes advantage of the fact that binary relations can be seen as sets.

The structure $\langle U, \cup, \cap, ^{-}, \bot, \top, \cdot, id, ^{\circ} \rangle$ is called a *algebra of binary relations* (ABR) (or proper relation algebra), where the universe U is a set of binary relations on a set A, and satisfies:

1. $\top$ is largest relation, i.e., $\top = \bigcup R \in U$, id is the identity relation on the set A and $\bot$ is the empty relation,
2. U is closed under set union $\cup$, intersection $\cap$ and complement $^{-}$ relative to $\top$,
3. $id, \bot, \top \in U$
4. U is closed under relational composition $(\cdot)$ and converse $(^{\circ})$, which are defined, for all $R, S \in U$, by:

$$R \cdot S = \{\langle a, b \rangle : \langle \exists c : a \, R \, c \wedge c \, S \, b \rangle\}$$

$$R^{\circ} = \{\langle a, b \rangle : b \, R \, a\}$$

In 1941, Alfred Tarski [Tar41] started developing a theory for a fragment of the calculus of relations consisting of boolean combinations of equations, which are formulas built up from disjunction, conjunction and negation of equations, proposing the first axiomatization for such fragment. However, Tarski eventually proved that every boolean combination of equations could be translated to an equivalent formula of the form $R = \top$, where R is an expression from the ABR. Following that theorem, Tarski began defining RAs using purely equational axioms, which resulted in the commonly used definition of RAs [TG87]. For the rest of the presentation, $R \subseteq S$ will stand as an abbreviation to the equation $R \cap S = R$.

A *relation algebra* (RA) (or abstract relation algebra) is an algebra

$$\langle A, +, ^{-}, ;, 1', ^{\smile} \rangle \tag{2.1}$$

where $+$ and $;$ are binary operations, $^-$ and $^\smile$ are unary, $1'$ is a distinguished element, and the following axioms hold:

$$x + y = y + x \tag{2.2}$$
$$x + (y + z) = (x + y) + z \tag{2.3}$$
$$\overline{\overline{x} + \overline{y}} + \overline{\overline{x} + y} = x \tag{2.4}$$
$$x ; (y ; z) = (x ; y) ; z \tag{2.5}$$
$$(x + y) ; z = x ; z + y ; z \tag{2.6}$$
$$x ; 1' = x \tag{2.7}$$
$$\breve{\breve{x}} = x \tag{2.8}$$
$$(x + y)^\smile = \breve{x} + \breve{y} \tag{2.9}$$
$$(x ; y)^\smile = \breve{y} ; \breve{x} \tag{2.10}$$
$$\breve{x} ; \overline{x ; y} + \overline{y} = \overline{y} \tag{2.11}$$

The first three axioms define the reduct $\langle A, +, ^- \rangle$ as a boolean algebra. The other operators and distinguished elements may then be defined as:

$$x \cdot y = \overline{\overline{x} + \overline{y}} \tag{2.12}$$
$$0 = \overline{1' + \overline{1'}} \tag{2.13}$$
$$1 = 1' + \overline{1'} \tag{2.14}$$

These axioms define an abstract algebraic structure, for which a concrete implementation is the ABR already presented, if to each operator and distinguished element in RA we match the concrete operator for relations in ABR. A RA which is isomorphic to an ABR is said to be *representable*.

Following an observation already stated by Peirce, Tarski [Tar41] also proved that the theory of RAs is undecidable, i.e., there is no algorithm for determining whether an equation in the calculus of relations can be derived from a given set of premises.

After presenting the first axiomatization, Tarski presented some questions about it [Tar41]. One that particularly interests us regards its expressability, i.e., if every formula of the calculus of relations is expressible in RAs. The answer to this question is due to Korselt [LÏ5] and had been known for several years. In fact, the RAs are equivalent to a three-variable restriction of first-order logic [TG87]. For instance, the following formula, from [TG87], is not expressible in RAs:

$$\langle \forall x, y, z : \langle \exists u : u \, \overline{1'} \, x \wedge u \, \overline{1'} \, y \wedge u \, \overline{1'} \, z \rangle \rangle \tag{2.15}$$

Tarski also questioned the *completeness* of its axioms, i.e., whether every sentence in the calculus of relations that is true in every ABR can be derived from his axioms, and their *representability*, that is, if every model of RAs is isomorphic to an ABR. Both questions were answered negatively by R. Lyndon [Lyn50] by presenting a non-representable RA.

2.3 Category Theory

Various frameworks have been proposed to extend RAs and overcome their expressiveness limitations. The choice of the categorical framework is due to

the fact that it also allows the specification of abstract datatypes. In particular, this presentation will follow the approach of Bird and Moor [BdM96].

Category theory [Fv90] provides a way to abstractly reason about mathematical structures and the relationships between them. The central idea is to generalize any mathematical constructions as objects and arrows between them. Categories and its central concepts were first introduced by Eilenberg and Lane in 1942-45 [EM45], but it was the definition of allegories by Freyd and Ščedrov [Fv90] that allowed the RAs to be defined in a categorical framework.

In this Section we begin by presenting the basic concepts of category theory, as well as concrete results for the category of sets and functions, and for the category of relations when applicable. Although the functional framework is not the focus of this report, many constructions of allegories are defined as an extension of their functional counterpart, and thus the need to present them as a starting point.

Definition A *category* **C** is an algebraic structure consisting of a collection of *objects* (A,B,C,...) and a collection of *arrows* (or *morphisms*) (f,g,h,...), as well as four operations. The first two operations are total and assign a *source* and a *target* objects to an arrow f, which is denoted by $f : B \to A$ (pronounced 'f is of type A from B'). Every object is also connected to itself by an *identity arrow*, i.e., for every object A there is an arrow $id_A : A \to A$.

Finally, there is a partial operation called *composition* which takes two arrows f and g and creates a new one, $f \cdot g$ (pronounced 'f after g'). It is defined if and only if $f : B \to A$ and $g : C \to B$, in which case $f \cdot g$ has type $C \to A$. Composition is associative and has identity arrows as units, i.e., for all $f : B \to A$, $g : C \to B$ and $h : D \to C$:

$$f \cdot (g \cdot h) = (f \cdot g) \cdot h \tag{2.16}$$

$$id_A \cdot f = f = f \cdot id_B \tag{2.17}$$

Whenever the type of the identity arrow is irrelevant or easily deduced, it is simply denoted by id.

Concrete categories A concrete example of a category is **Fun**, the category of sets and total functions. In **Fun**, objects are sets and arrows are typed total functions $f : B \to A$, where A is the range and B is the domain of f. The identity arrows are identity functions and the composition of arrows is the typical composition of functions.

A generalization of **Fun** is the category **Rel**, the category of sets and relations. Intuitively, an arrow $R : B \to A$ can be seen as a subset of the cartesian product $A \times B$. The identity arrow $id_A : A \to A$ is defined by:

$$id_A = \{(a, a) \mid a \in A\} \tag{2.18}$$

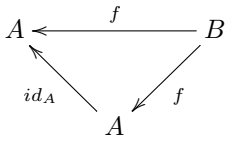
and the composition of $R : B \to A$ with $S : C \to B$ is the arrow $T : C \to A$ where

$$a\, T\, c \equiv \langle \exists b : a\, R\, b \wedge b\, S\, c \rangle \tag{2.19}$$

Since total functions are particular cases of relations, **Fun** is a *subcategory* of **Rel**.

Note that order of the input and output of the relations are interchanged in relation to the RAs. This actually makes sense if we see relations as an extension of functions, which also take the input on the right-hand side.

Diagrams Since everything is represented by arrows, *diagrams* provide an intuitive way to represent information. For example, the equation $id_A \cdot f = f$ could be represented as:



When different paths between a pair of objects represent the same arrow, like in this case, the diagram is said to *commute*.

Duality For any category $\mathbf{C}$, the *opposite* category $\mathbf{C}^{op}$ is defined by the same objects and arrows of $\mathbf{C}$, but with their source and target are interchanged. Also, composition is defined by swapping the arguments. Since reversing arrows twice does nothing, $(\mathbf{C}^{op})^{op} = \mathbf{C}$. Categories benefit from a special case of symmetry called *duality*, which means that any statement valid in a category is valid in its opposite.

Another interesting result is that $\mathbf{Rel} = \mathbf{Rel}^{op}$, because every relation, by definition, has a converse, and thus every arrow in **Rel** also has a converse. This also means that in **Rel** every construction is equal to its dual.

Isomorphisms An *isomorphism* is an arrow $i : B \to A$ such that there is an arrow $j : A \to B$ for which $j \cdot i = id_B$ and $i \cdot j = id_A$. The arrow j is unique, being called the *inverse* of i, and denoted by i^{-1}. In this case, we say that A and B are isomorphic, denoted by $A \cong B$. In **Fun** and **Rel**, these arrows represent bijections.

Terminal and Initial objects A *terminal* object 1 of a category $\mathbf{C}$ is an object such that, for any object A of $\mathbf{C}$, there is exactly one arrow $A \to 1$. All terminal objects are isomorphic, so we will refer only to *the* terminal object, and denote the unique arrow $A \to 1$ by $!_A$.

The uniqueness of arrows is usually defined by *universal properties*. In the case of $!_A$, it is defined by

$$h = !_A \equiv h : A \to 1 \tag{2.20}$$

Terminal objects represent a datatype with only one element. In **Fun** it is a singleton set, which means all singleton sets are isomorphic, and the arrow $!_A$ maps every element of A to the element of the singleton set. In **Rel** it is the empty set, and the arrow is the empty relation $\emptyset$.

An *initial* object 0 of $\mathbf{C}$ is an object for each there is exactly one arrow $0 \to A$, for all objects A of $\mathbf{C}$. An initial object of $\mathbf{C}$ is a terminal object in $\mathbf{C}^{op}$. All initial objects are also isomorphic, so we refer only to *the* initial object. The unique arrow $0 \to A$ is represented by $¡_A$.

In **Fun** the initial object is the empty set, and the arrow $¡_A$ is the empty function. Note that the names 1 and 0 represent the cardinality of its sets in **Fun**. In **Rel** the initial object is also the empty set.

Functors A *functor* $\mathsf{F} : \mathbf{D} \to \mathbf{C}$ is a homomorphism between categories $\mathbf{C}$ and $\mathbf{D}$. They consist of two mappings, usually defined by the same letter F: one that associates every object $A \in \mathbf{C}$ to an object $\mathsf{F}A \in \mathbf{D}$, and one from every morphism $f : B \to A \in \mathbf{C}$ to morphisms $\mathsf{F}f : \mathsf{F}B \to \mathsf{F}A$, satisfying the following properties:

$$\mathsf{F}id_A = id_{\mathsf{F}A} \tag{2.21}$$

$$\mathsf{F}(g \cdot f) = \mathsf{F}g \cdot \mathsf{F}f \tag{2.22}$$

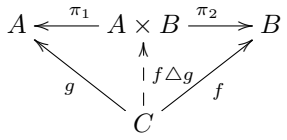
The most simple example, for every category $\mathbf{C}$ there is an *identity functor* $\mathsf{id}_{\mathbf{C}} : \mathbf{C} \to \mathbf{C}$. Functors may also be composed as $(\mathsf{G} \cdot \mathsf{F})\mathbf{C} = \mathsf{G}(\mathsf{F}\mathbf{C})$, and as such, they represent morphisms on the category of categories.

Another useful functor is the *constant functor* $\mathsf{K}_C : \mathbf{D} \to \mathbf{C}$, that matches every object of $\mathbf{D}$ to the same object $C \in \mathbf{C}$.

Product The *product* of two objects A and B is an object $A \times B$ and two arrows: $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$, such that, for each arrow $f : C \to A$ and $g : C \to B$ there is a unique arrow $f \triangle g : C \to A \times B$, called '*split* of f and g' with:

$$h = f \triangle g \equiv \pi_1 \cdot h = f \text{ and } \pi_2 \cdot h = g \tag{2.23}$$

for all $h : C \to A \times B$. This defines the universal property of products: $f \triangle g$ is the unique arrow that satisfies the property on the right-hand side of equation (2.23). This property can be represented by the following diagram:



where a dashed arrow represents its uniqueness.

Considering concrete categories, the product in **Fun** is given by the cartesian product and π_1 and π_2 are the projection functions. The arrow $f \triangle g$ would then be defined by

$$(f \triangle g)\, a = (f\, a, g\, a)$$

This definition however does not hold for **Rel**, since the pairing would have undefined values when the functions are partial. We will postpone the definition of relational products to the Section 2.4 where allegories are defined.

A useful operator derived from splits is the *product of morphisms* $\times$, that is defined by:

$$f \times g = (\pi_1 \cdot f) \triangle (\pi_2 \cdot g) \tag{2.24}$$

In **Fun** and **Rel** this operator represents the parallel application of functions. A property directly defined by the split universal law is the following absorption law:

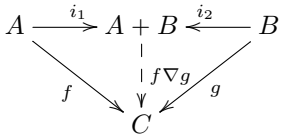
$$(f \times g) \cdot (h \triangle i) = (f \cdot h) \triangle (g \cdot i) \tag{2.25}$$

In categories where every pair of objects has a product (like in **Fun** and **Rel**), the $\times$ operator defines a bifunctor $\mathbf{C} \times \mathbf{C} \to \mathbf{C}$.

Coproduct The *coproduct* is the dual operation of the product, also consisting of an object and two arrows, but with the source and target interchanged. The object is denoted by $A + B$ and the arrows $i_1 : A \to A + B$ and $i_2 : B \to A + B$. Given $f : A \to C$ and $g : B \to C$, the unique arrow $f \nabla g$, called '*either* f or g' is defined by the universal property:

$$h = f \nabla g \equiv h \cdot i_1 = f \text{ and } h \cdot i_2 = g \tag{2.26}$$

for all $h : A + B \to C$. Once again, this information can be depicted in a diagram:



In the category **Fun**, coproducts are disjoint unions. In functional programming this would be represented by:

$$A + B ::= i_1\, A \,|\, i_2\, B$$

and the either operator defined by:

$$(f \nabla g)\,(i_1\, a) = f\, a \text{ and } (f \nabla g)\,(i_2\, a) = g\, a$$

Coproducts are represented in **Rel** the same way they are in **Fun**. Since $\mathbf{Rel} = \mathbf{Rel}^{op}$, and coproducts are dual to products, this means products may also be represented by a disjoint union, i.e., $A \times B \cong A + B$.

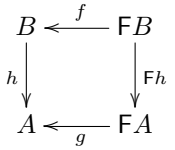
Like with the product, we define the *coproduct of morphisms* $+$, that defines a bifunctor:

$$f + g = (f \cdot i_1) \nabla (g \cdot i_2) \tag{2.27}$$

Polynomial functors Functors built up from constants, products and coproducts are said to be *polynomial*, which already allow the definition of some interesting datatypes. However, they still lack the expressive power to define recursively define types, which will be presented next, or to define morphisms over the function space, which is provided by *exponentiation*, that goes beyond the scope of this presentation.

Catamorphisms Given a functor F, an arrow of the type $\mathsf{F}A \to A$ is said to be an F-*algebra*. A F-*homomorphism* between two F-algebras f and g is given by an arrow h such that:

$$h \cdot f = g \cdot \mathsf{F}h \tag{2.28}$$



F-algebras and F-homomorphisms form a particular category. For some functors F (including the polynomial), this category has an initial algebra, meaning that for every F-algebra f exists a unique homomorphism $h : T \to A$. This homomorphisms are called *catamorphisms*, denoted by $(\!|f|\!) : T \to A$, and defined by the following universal law:

$$h = (\!|f|\!) \equiv h \cdot \mu\mathsf{F} = f \cdot \mathsf{F}h \tag{2.29}$$

$$\begin{array}{ccc} T & \xleftarrow{\mu\mathsf{F}} & \mathsf{F}T \\ {\scriptstyle h}\,\vdots & & \downarrow{\scriptstyle \mathsf{F}h} \\ A & \xleftarrow[f]{} & \mathsf{F}A \end{array}$$

One of the most useful laws of the catamorphisms is the fusion law:

$$h \cdot (\!|f|\!) = (\!|g|\!) \Leftarrow h \cdot f = g \cdot \mathsf{F}h \tag{2.30}$$

2.4 Allegories and Relations

Allegories abstract RAs in the same way categories abstract function algebras, and in particular, **Rel** is an allegory. Since **Fun** is a subcategory of **Rel**, some relational operators are defined as an extension of its functional counterpart.

An allegory **A** is a category extended with three operators inspired by RAs: inclusion, meet and converse.

Inclusion Two arrows of the same type can be compared by the partial order $\subseteq$, with respect to which composition is monotonic. In **Rel**, inclusion represents the set-theoretical inclusion, i.e.:

$$R \subseteq S \equiv \langle \forall a, b : a\, R\, b \Rightarrow a\, S\, b \rangle \tag{2.31}$$

Expressions of the shape $R \subseteq S$, called *inequations*, are the basic components of this framework. As in equations, diagrams are also used to represent inequations, in which case they are said to *semi-commute*, which is denoted by a $\subseteq$ symbol. For instance, the expression $S_2 \cdot S_1 \subseteq R_2 \cdot R_1$ is depicted by:

$$\begin{array}{ccc} A & \xleftarrow{S_1} & B \\ {\scriptstyle S_2}\downarrow & \subseteq & \downarrow{\scriptstyle R_1} \\ C & \xleftarrow[R_2]{} & D \end{array}$$

Meet and converse In an allegory, for all arrows $R, S : B \to A$ there is an arrow $R \cap S : B \to A$, called *meet* of R and S, whose universal property is, for all $X : B \to A$:

$$X \subseteq (R \cap S) \equiv (X \subseteq R) \text{ and } (X \subseteq S) \tag{2.32}$$

From this property we can derive that meet is commutative, associative and idempotent:

$$R \cap S = S \cap R \tag{2.33}$$

	Reflexive	Coreflexive
$\ker R$	entire	injective
$\mathrm{img}\, R$	surjective	simple

Table 2.1: Relation properties.

$$R \cap (S \cap T) = (R \cap S) \cap T \tag{2.34}$$

$$R \cap R = R \tag{2.35}$$

Finally, for every arrow $R : B \to A$ there is an arrow $R^\circ : A \to B$ called the *converse* of R, which is an involution, order-preserving and contravariant:

$$(R^\circ)^\circ = R \tag{2.36}$$

$$R \subseteq S \equiv R^\circ \subseteq S^\circ \tag{2.37}$$

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{2.38}$$

Reflexive and Coreflexive Arrows Since we are using arrows to represent relations, they inherit some properties from RAs. An arrow $R : A \to A$ is said to be *reflexive* if $id_A \subseteq R$. On the other hand, an arrow $R : A \to A$ is said to be *coreflexive* if $R \subseteq id_A$, i.e., a fragment of the identity arrow.

Consider the definition of *kernel*, $\ker R \triangleq R^\circ \cdot R$, and *image*, $\mathrm{img}\, R \triangleq R \cdot R^\circ$, of a relation. If the kernel of an arrow is reflexive ($id_A \subseteq \ker R$), then the arrow is *total* (or *entire*). Also, if the image of an arrow is coreflexive ($\ker R \subseteq id_A$), the arrow is *simple* (or *functional*). Coreflexive kernels and reflexive images induce injectivity and surjectivity in a relation, respectively. These properties are resumed in Table 2.1. Combinations of these properties define different classes of relations, as presented in Figure 2.1. For instance, arrows that are both entire and simple are functions. Let's quickly convert the definition os an injective relation back to PW to check its meaning:

$$\begin{aligned}
&\ker R \subseteq id \\
\Leftrightarrow&\{\ker \text{ - DEF}\} \\
&R \cdot R^\circ \subseteq id \\
\Leftrightarrow&\{\subseteq \text{ - PW}\} \\
&\langle \forall x, y : x\, R \cdot R^\circ\, y \Rightarrow x\, id\, y \rangle \\
\Leftrightarrow&\{\cdot \text{ - PW}, {}^\circ \text{ - PW}, id \text{ - PW}\} \\
&\langle \forall x, y : \langle \exists k : x\, R\, k \wedge y\, R\, k \rangle \Rightarrow x = y \rangle
\end{aligned}$$

Since coreflexives are contained in the identity relation, they act as a filter, since whatever value is passed as input comes out as the output, if it belongs to the coreflexive. As such, coreflexives can be used to encode predicates on the relational setting. For a perdicate p, Φ_p will denote the coreflexive containing the values that satisfy p:

$$x\, \Phi_p\, y \Leftrightarrow p\, x \wedge x = y$$

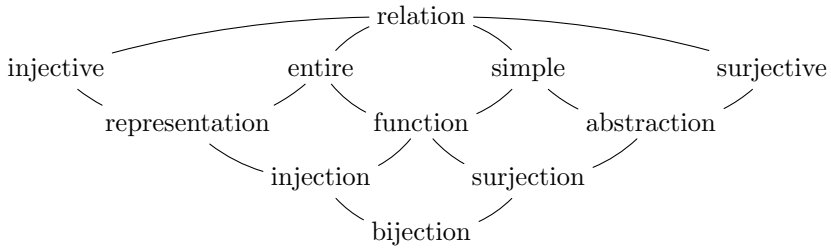


Figure 2.1: Relation taxonomy.

Predicate logic can also be expressed in terms of coreflexives due to the following equivalences:

$$\Phi_{p \wedge q} = \Phi_p \cdot \Phi_q = \Phi_p \cap \Phi_q \tag{2.39}$$
$$\Phi_{p \vee q} = \Phi_p \cup \Phi_q \tag{2.40}$$
$$\Phi_{\neg p} = id - \Phi_p \tag{2.41}$$
$$\Phi_{false} = \bot \tag{2.42}$$
$$\Phi_{true} = id \tag{2.43}$$

For simplification purposes, $\Phi_\neg$ will denote the coreflexive $id - \Phi$

Range and Domain Every arrow $R : B \to A$ has associated two coreflexives, the *domain* $\delta(R) : B \to B$ and the *range* $\rho(R) : A \to A$. Note that since $\delta(R) = \rho(R^\circ)$, the properties are easily interchangeable. The range may be defined by the universal law:

$$\delta(R) \subseteq X \equiv R \subseteq X \cdot R, \text{for all } X \subseteq id_A \tag{2.44}$$

However, it is usually directly defined by:

$$\delta(R) = (R \cdot R^\circ) \cap id \tag{2.45}$$

The domain definition and rules can be obtain by duality:

$$\rho(R) = (R^\circ \cdot R) \cap id \tag{2.46}$$

Tabular allegories Allegories do not fully abstract RAs. However, one needs only to assume the existence of *tabulations* and a *unit* in the allegory to achieve set-theoretic relations. In particular, **Rel** is tabular and unitary.

An allegory is said to be *tabular* if for every arrow $R : B \to A$, there exist two functions f and g such that:

$$R = f \cdot g^\circ \tag{2.47}$$

$$(f^\circ \cdot f) \cap (g^\circ \cdot g) = id \tag{2.48}$$

Since relations can be seen as a subset C of a cartesian product $A \times B$, we have that projections $\pi_1 : C \to A$ and $\pi_2 : C \to B$ are tabulations of R.

A *unit* U is an object for which every arrow $U \to U$ is coreflexive. Moreover, for every object A of the allegory, there is an arrow $p_A : A \to U$. An allegory with a unit is said to be unitary. As a consequence, $p_A^\circ \cdot p_B$ is the largest arrow of type $B \to A$, which will be denoted by $\top_{B \to A} : B \to A$, called *top*. Whenever the type can be deduced, we omit it and simply denote the relation as $\top$. In **Rel** the unit is the singleton set, and $\top : B \to A$ is the cartesian product $A \times B$.

Join We can extend the allegories with the *join* operator, denoted by $R \cup S$ for all $R, S : B \to A$, which represents the reunion of two relations. It is defined by the following universal law:

$$R \cup S \subseteq C \equiv (R \subseteq C) \text{ and } (S \subseteq C) \tag{2.49}$$

for all $X : B \to A$.

Like meet, join is associative, commutative and idempotency. However, unlike with meet, composition distributes over join:

$$R \cdot (S \cup T) = (R \cdot S) \cup (R \cdot T) \tag{2.50}$$

$$(S \cup T) \cdot R = (S \cdot R) \cup (T \cdot R) \tag{2.51}$$

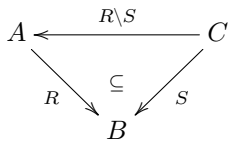
Allegories where join operator is defined for all morphisms, like **Rel**, are called *locally complete*, and give rise to other useful operators, which will be presented next.

Division The *left-division* and *right-division* operators are defined respectively by the following universal properties:

$$X \subseteq R \backslash S \equiv R \cdot X \subseteq S \tag{2.52}$$

$$X \subseteq R / S \equiv X \cdot R \subseteq S \tag{2.53}$$

Drawing the diagram of this definition, which semi-commutes, we get:



Since these are not common operators, we will resort to the PW definitions, which are more intuitive:

$$a \, R \backslash S \, b \equiv \langle \forall c : c \, R \, a \Rightarrow c \, S \, b \rangle$$

$$a \, R / S \, b \equiv \langle \forall c : a \, R \, c \Rightarrow b \, S \, c \rangle$$

Division allows us to represent universal quantifiers in PF notation, much as the composition defines existential quantifiers.

Tabular allegories also allow us to define the *implication* and *difference* operators:

$$X \subseteq R \Rightarrow S \equiv R \cup X \subseteq S \tag{2.54}$$

$$X \subseteq S \cup R \equiv X - S \subseteq R \tag{2.55}$$

Note the similarity with the definition of the division operators. These kind of rules are called *Galois connections* [Ore44]. Although it is not the approach we follow, there is a different trend on the calculus of relations highly based on GC [Bac04]. In category theory they are called *adjoints*.

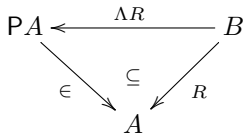
Power allegories Typically, relations are seen as a subset of the cartesian product of two sets, as defined in set theory. However, relations can also be seen as set-valued functions, providing a different approach to reason about relational formulas. Such functions can be achieved by defining the notion *powerset*, giving rise to *power allegories*. This correspondence between functions and relations will also prove useful in the extension of functional structures to the relational settings.

To model power-set functions we need three operators:

- for each object A, a *power-set object* $\mathsf{P}A$;
- a function Λ called *power transpose* that converts a relation $R : B \to A$ into the function $\Lambda R : B \to \mathsf{P}A$;
- the membership arrow $\in\colon \mathsf{P}A \to A$.

which are defined by the following universal property, depicted in the diagram:

$$f = \Lambda R \equiv \in \cdot f = R \tag{2.56}$$



As such, these operators allow the definition of relations as functions, following the PW intuition:

$$(\Lambda R)\, b = \{a \mid a\, R\, b\}$$

Relators Let **A** and **B** be tabular allegories. Then, a *relator* is a monotonic functor $\mathsf{F} : B \to A$, i.e.

$$R \subseteq S \Rightarrow \mathsf{F}R \subseteq \mathsf{F}S \tag{2.57}$$

Equivalently, a functor F is a relator if it preserves converse, that is, $(\mathsf{F}R)^\circ = \mathsf{F}(R^\circ)$.

Relational product As has been seen when presenting the categorical datatypes, the product as defined in **Fun** is not valid in allegories due to the possible partiality of the relations, so they must be defined in a different way. Without going into much detail, in a unitary tabular allegory, the product may be defined as:

$$R \triangle S = (\pi_1^\circ \cdot R) \cap (\pi_2^\circ \cdot S) \tag{2.58}$$

The $\times$ operator is defined the usual way by:

$$R \times S = (R \cdot \pi_1) \triangle (S \cdot \pi_2) \tag{2.59}$$

The cancellation laws of categorical products do not hold for the allegorical product (unless we are within the subcategory of functions). Instead, the following properties hold:

$$\pi_1 \cdot (R \triangle S) = R \cdot \delta(S) \text{ and } \pi_2 \cdot (R \triangle S) = S \cdot \delta(R) \tag{2.60}$$

$$(R \triangle S)^\circ \cdot (X \triangle Y) = (R^\circ \cdot X) \cap (S^\circ \cdot Y) \tag{2.61}$$

Moreover, the $\times$ operator is a relator.

The PF reasoning about the relational product is that $(a, b)\, R \triangle S\, c$ is defined when $a\, R\, c$ and $b\, S\, c$. If c is not in the domain of R or S, their product is not defined.

Relational coproduct Unlike the product, the coproduct in a power allegory may be defined as an extension of the coproduct in **Fun**. As such, considering a power allegory where the coproduct is defined in its subcategory of functions, the coproduct may be defined as $R \nabla S = \in \cdot (\Lambda R \nabla \Lambda S)$. However, by performing simple calculations, this definition gives rise to the most direct and usual definition of the relational coproduct:

$$R \nabla S = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \tag{2.62}$$

The $+$ operator, a relator, is defined the same way as in the categorical setting:

$$R + S = (i_1 \cdot R) \nabla (i_2 \cdot S) \tag{2.63}$$

Relational catamorphisms Catamorphisms in allegories are defined as an extension of categorical catamorphisms, in a way similar to the coproducts. Given a functor F with an initial algebra $\mu\mathsf{F}$ in the subcategory of functions, it can be shown that $\mu\mathsf{F}$ is also initial in the whole allegory, and thus:

$$(\!|R|\!) = \in \cdot (\!|\Lambda(R \cdot \mathsf{F} \in)|\!) \tag{2.64}$$

Chapter 3

Bidirectional Transformations

The foundations of the bidirectional transformations as they are understood today arise from the research in the database community on the *view-update* problem, in particular the "correct update translation" from Dayal and Bernstein [DB82], the "constant complement" from Bancilhon and Spyratos [BS81] and the "dynamic views" from Gottlob, Paolini and Zicari [GPZ88].

Recently, the interest on the view-update problem has risen again, but on the context of structured data [Mee98, FGM+07]. From these works, a large body of work has been emerging, with applications on programming languages, model-driven software development, data transformation or graphical user interfaces.

In this Chapter, we will start by defining a unifying framework where most of the existing approaches can be classified and compared. RAs were chosen as the underlying framework precisely due to their flexibility. In the following Section, some of the related work on bidirectional transformations will be presented and compared under the defined taxonomy.

3.1 RAs as a Unifying Framework

Representing all known approaches on bidirectional transformations under the same formal framework poses a challenge due to the variety of mathematical formalisms over which they are defined. For instance, while some assume the framework to be fully functional, others assume transformations to be partial and non-deterministic. Others still assume the calculus to be functional but possibly partial, which gives rise to the need to consider the definability of the properties.

Moreover, the notation and conventions vary greatly from community to community, as do the notions of formality. Some may rely heavily on mathematical foundations, while other provide little formal reasoning, but both must be comparable nonetheless.

Few efforts have been developed in order to unify all bidirectional approaches under the same framework. Diskin [Dis08] defined a set of laws in three generic frameworks, analyzing their different meaning in each context. However, his goal was not to compare concrete, existing approaches. The report from the

GRACE international meeting [CFH+09] also presented an attempt to clarify some concepts and relate the bidirectional approaches from several different areas, but not with a formal specification in mind.

RAs, as a very general framework, seems to be a natural choice to unify the approaches. Since relations are not bound by any restriction of totality or determinism, they provide a way to represent any approach based on a formal framework, and compare previously incomparable frameworks.

During the presentation, we will assume the transformations to be arbitrary relations, and thus denoted by *Get*, *Put* and *Create*. This means that the analysis of the behavior of the operation and its classification (from Figure 2.1) will be preformed separately, resulting in more general rules. Once they are analyzed together, they will degenerate in the more concrete laws defined in previous work.

3.2 Taxonomy

As the classification is defined, the symbol which will denote each class is also presented between parenthesis.

3.2.1 Frameworks

The framework in which the transformations are defined directly influences the expressibility of the transformation system and its underlying properties. Generally, bidirectional frameworks fit in one of three classes, depending on the shape of the transformations:

Mappings ($\rightleftharpoons$) transformations consist of a forward transformation $Get : S \to V$ and a backward transformation $Create : V \to S$;

Lenses ($\trianglerighteq$) transformations consist of a forward transformation $Get : S \to V$ and a backward transformation $Put : S \times V \to S$, where the S in the domain is the original model;

Maintainers ($\lesseqgtr$) transformations consist of a forward transformation $\overrightarrow{R} : S \times V \to V$ and a backward transformation $\overleftarrow{R} : S \times V \to S$, which preserve a consistency relation $R : S \to V$.

Since mappings preserve no information about the original model, forward transformation usually need to be injective in order to preserve some kind invertibility. They can be used to define refinement systems by abstraction [Oli08], or in the programming language community, reversible languages. At the extreme case, they define bijective transformations, where the information in both sides is exactly the same.

In lenses, even though the views usually contain less information than the sources, since the original model is preserved forward transformations are able to disregard information that can be retrieved when applying the backward transformation. Nevertheless, although some restrictions on the transformations are lifted, in order to be provide interesting properties transformations should still be within a restricted scope. For instance, to obtain total transformations, the forward transformation must be surjective.

Constraint maintainers, by being symmetric provide an even higher degree of freedom, since they allow the definition of transformations where both sides may contain information that does not exist in the other. Moreover, restrictions on the transformations are now only imposed by the consistency relation. This generalization, however, comes with a cost, since compositional reasoning about the correctness of transformations is usually no longer possible. Updating a value to a consistent state is usually call synchronization.

3.2.2 Representation of Updates

Update representation concerns the information available to the transformations in order to calculate the updated model.

State-based (S) transformations take as input only the states of the model, ignoring how the new state was attained;

Operation-based (O) transformations take as input the new state of the model, along with some information about how the new state was attained. Operation-based approaches can be further classified as:

History-based (or directed) (O_H) updates are defined by the sequence of editing operations performed on the model;

Canonical (or symmetric) (O_C) updates specify only the modified elements of the model, by establishing the correspondence between the elements of the original and modified models.

As the name implies, history-based updates can be converted into canonical updates, by removing possible redundancies in the editing sequence.

State-based approaches can lead to incorrect transformations. For instance, the removal and insertion of an object with the same name may be identified as a modification in a state-based transformation. This problem of matching the elements of different models is known as *alignment*, and is still an active research topic. However, not only operation-based approaches are usually more complex to implement, but sometimes there is still the need to restrict the range of possible operations in order to avoid an excessive complexity of the system.

3.2.3 Bidirectionalization

Bidirectionalization is the process through which, for an unidirectional transformation, a second transformation is calculated which guarantees the bidirectional properties. Bidirectionalization may be analyzed in two axis. First, the framework in which the transformations are defined:

Ad-hoc ($\times$) the approach does not define a framework;

Combinatorial (C) transformations are defined in a domain-specific language, where bidirectionalization is ensured by construction;

Syntactic (S) forward transformations are defined in a possibly general language, while the backward transformation is calculated based on the syntactic analysis of former;

Semantic (s) forward transformations are defined in a possibly general language, while the backward transformation is calculated at run-time for each specific execution.

The second classification regards how the bidirectional properties of each transformation are guaranteed:

Manual (M) the backward transformation was manually defined and checked for correctness at the development of the framework;

Calculated (C) the backward transformation is automatically calculated and correct by construction;

Syntactic (S) the backward transformation is calculated and then checked syntactically for correctness at compilation time;

Semantic (s) bidirectionalization is checked at run-time through the analysis of each execution.

The primitives of combinatorial frameworks are usually defined manually, but since they form a domain-specific language the transformations will be correct by construction. Syntactic frameworks may also rely in syntactic analysis to filter out some transformations for which the desired properties do not hold. All frameworks may have the bidirectionality checked at semantic level, since the primitives themselves may perform some checks at run-time.

Another characteristic that may affect the well-behavedness of the system is allowing the user to define bidirectional transformations (U), which may or not guarantee the desired properties.

3.2.4 Transformation Properties

The behavior of the transformations is defined by a set of laws, that define what is considered a correct bidirectional transformation.

Round-trip Round-trip laws concern the information preserved by the transformations:

Invertibility These laws guarantee that applying a transformation and then instantly applying the inverse transformations, i.e., no update is performed, yields no changes:

$$
\begin{array}{cl}
\rightleftharpoons & (\longrightarrow)\ Get \cdot Create \subseteq id \\
 & (\longleftarrow)\ Create \cdot Get \subseteq id \\
\trianglerighteq & (\longrightarrow)\ Get \cdot Put \subseteq \pi_1 \\
 & (\longleftarrow)\ Put \cdot (Get \triangle id) \subseteq id \\
\trianglelefteq & (\longrightarrow)\ \overleftarrow{R} \cdot (\pi_1 \triangle \overrightarrow{R}) \subseteq \pi_1 \\
 & (\longleftarrow)\ \overrightarrow{R} \cdot (\overleftarrow{R} \triangle \pi_2) \subseteq \pi_2
\end{array}
$$

In mappings, the $\longrightarrow$ rule defines *left-invertibility* and the $\longleftarrow$ defines *right-invertibility*. If both rules hold, the transformations are invertible. In the lens framework these laws are known as *stability* and *acceptability* respectively.

Convergence These weaker laws do not guarantee that the inverse transformation yields the same result, but that it somehow contains the same information, since applying the transformation again yields the same result:

$$\begin{array}{ll} \rightleftharpoons & (\dashrightarrow)\ Get \cdot Create \cdot Get \subseteq Get \\ & (\dashleftarrow)\ Create \cdot Get \cdot Create \subseteq Create \\ \trianglerighteq & (\dashrightarrow)\ Get \cdot Put \cdot (Get \triangle id) \subseteq Get \\ & (\dashleftarrow)\ Put \cdot (Get \triangle id) \cdot Put \subseteq Put \\ \trianglelefteq & (\dashrightarrow)\ \overleftarrow{R} \cdot (\pi_1 \triangle \overrightarrow{R}) \cdot (\overleftarrow{R} \triangle \pi_2) \subseteq \overleftarrow{R} \\ & (\dashleftarrow)\ \overrightarrow{R} \cdot (\overleftarrow{R} \triangle \pi_2) \cdot (\pi_1 \triangle \overrightarrow{R}) \subseteq \overrightarrow{R} \end{array}$$

The presented definitions are what we consider the standard ones. However, several variants are presented in the various techniques. For instance, exact invertibility is sometimes an excessive restriction, since in some cases the identity can be relaxed in order to ignore details. In this situation, the laws are defined modulo some equivalence relation, that defines which values are considered "identical" and the round-trip laws will be denoted by $\overset{\sim}{\longleftrightarrow}$. Some laws also need to define additional restrictions on the transformations, in which case we will denote the laws by $\overrightarrow{\longleftarrow}$. Typically this happens in operation-based approaches, where the updates must also be constrained by the laws.

Definitions may also vary slightly in order to impose different partiality properties. In our presentation it is assumed that the operations are partial, leaving the reasoning about totality to a different axis.

Evolution Evolution regards the behavior of the system upon multiple transformations.

History ignorance ($\circlearrowleft$) A system is history ignorant if for a sequence of updates, applying only the last one yields the same result:

$$Put(v_k, \ldots Put(v_2, Put(v_1, s)) \ldots) = Put(v_k, s)$$

History ignorance is usually guaranteed by the following law:

$$\begin{array}{ll} \rightleftharpoons & \text{trivial} \\ \trianglerighteq & Put \cdot (id \times Put) \subseteq Put \cdot (id \times \pi_2) \\ \trianglelefteq & \overrightarrow{R} \cdot (id \times \overrightarrow{R}) \subseteq \overrightarrow{R} \cdot (id \times \pi_2) \quad \wedge \quad \overleftarrow{R} \cdot (\overleftarrow{R} \times id) \subseteq \overleftarrow{R} \cdot (\pi_1 \times id) \end{array}$$

Undoability ($\hookleftarrow$) A particular case of history ignorance (assuming stability also holds), is undoability. A system is undoable if, after updating the source with a new view, providing the original view again yields the original source:

$$\begin{array}{ll} \rightleftharpoons & \longleftarrow \\ \trianglerighteq & Put \cdot (Get \cdot \pi_2 \triangle Put) \subseteq \pi_2 \\ \trianglelefteq & \overrightarrow{R} \cdot (\overleftarrow{R} \triangle \overrightarrow{R}) \subseteq \pi_2 \quad \wedge \quad \overleftarrow{R} \cdot (\overleftarrow{R} \triangle \overrightarrow{R}) \subseteq \pi_1 \end{array}$$

Consistency (R) Consistency, as the following property hippocratiness, is a law that may only be defined for systems where there is a consistency

relation. In consistent systems, synchronizing two inconsistent values restores the consistency.

$$\leqslant \quad \pi_1 \subseteq R \cdot \overrightarrow{R} \quad \wedge \quad \pi_2 \subseteq R^{\circ} \cdot \overleftarrow{R}$$

Hippocratiness ($\downdownarrows$) A system is hippocratic if, for two models already consistent, performing a synchronization does not change either model.

$$\leqslant \quad R \subseteq \pi_2^{\circ} \cdot \overrightarrow{R} \quad \wedge \quad R \subseteq \pi_1^{\circ} \cdot \overleftarrow{R}$$

In some systems the laws that are expected to hold are simply specified, following generally some informal argument. In such cases we denote the laws as *moral* (M).

3.2.5 Totality

Totality regards the values for which the transformations are defined:

Partial ($-$) both forward and backward operations are partial;

Safe ($\ominus$) both *Put* and *Get* operations are defined at least for the range of each other:

$$\begin{array}{ll} \rightleftharpoons & \rho Create \subseteq \delta Get \\ & \rho Get \subseteq \delta Create \\ \trianglerighteqslant & \rho Put \subseteq \delta Get \\ & \rho(Get \times id) \subseteq \delta Put \\ \leqslant & \rho\overrightarrow{R} \subseteq \delta\overleftarrow{R} \\ & \rho\overleftarrow{R} \subseteq \delta\overrightarrow{R} \end{array}$$

Correct ($\oplus$) the *Get* operation is total, while the *Put*, for each view, is at least defined by all values in the range of *Get*:

$$\begin{array}{ll} \rightleftharpoons & id \subseteq \delta Get \\ & \rho Get \subseteq \delta Create \\ \trianglerighteqslant & id \subseteq \delta Get \\ & \rho(Get \times id) \subseteq \delta Put \\ \leqslant & \delta\overrightarrow{R} \subseteq id \times \rho R \\ & \delta\overleftarrow{R} \subseteq \delta R \times id \end{array}$$

Total ($+$) both *Put* and *Get* operations are total.

The level of totality of a framework depends on the goal of the system. For instance, while correct systems suffice to guarantee correct bidirectional properties, only total systems allow the user to update the view to an arbitrary value.

3.2.6 Application Domain

Application domain of the framework regards the underlying data structure manipulated by the transformations. While some are clearly more expressive than others, there is always a trade-off between the expressiveness and the efficiency and complexity of the transformations on those structures.

Strings (S) Strings are one of the simplest data structures, defining ordered unstructured data. Nonetheless, since there exists a lot of data in string format, and string representation and manipulation are extremely simple and efficient, they still provide an useful framework.

Trees (T) Trees provide a way to define hierarchical structured data, comprising most of the possible types in common programming languages. Also, unlike strings, trees provide a syntactic representation of the data.

Relational data (R) Relations provide a way to represent relational databases. Although relational data can be encoded in trees, some characteristics of relational data, like functional dependencies, can not be naturally captured.

Graphs (G) Graph structures generalize trees by allowing cycles and multiple parents, providing means to represent more complex data structures. In fact, most models from software engineering, an area where bidirectional transformations are essential, are only representable by graphs. However, graph manipulation and transformation is much more complex.

3.2.7 Classes of Transformations

The combination of the transformation and totality properties gives rise to different kinds of systems, with particular properties. In this Section we will briefly analyze those classes.

Since mappings do not preserve the original model, systems that guarantee round-tripping rules will be much more restrictive. Following the work from [Oli08] on abstraction transformations, systems with left-invertibility provide the "no confusion" principle, meaning that abstracting after representing always yields the same value. If the operations are correct concerning totality, the transformations define a refinement system, with the abstraction/representation pair $Get/Create$, where $Create \cdot Put = id$. In this case there is also the "no loss" principle, meaning the operations never fail. This directly implies that $Create$ is total and injective, and that Get is surjective, which are very strong restrictions. Considering the case where full invertibility is attained, the transformations are bijections, in which case the views must contain exactly the same information as the sources.

The original work on lenses [FGM+07] defines as well behaved lenses the ones guaranteeing invertibility. In the lenses framework, the original source object is stored in order to calculate the new one, improving the expressibility and allowing Get to no longer be injective, since information may be retrieved form the original source. Assuming the operations to be total, the pairs $Get/\pi_1^\circ \cdot Put$ and $Put/(Get \triangle id)$ also form refinement patterns. As such, we know that Get is surjective and total and Put left-total, left-injective and surjective. In fact, it had already been shown that any for Get operation that is total and surjective

there is a Put operation that forms a well-behaved lens, and for any left-total, left-injective and surjective Put, there is also a corresponding Get that forms a lens [Fos09].

The convergence laws were first proposed in approaches where that duplication was allowed, since such combinator breaks the invertibility laws. However, these weaker laws allow some non-desirable behavior. For instance, constant Put operations like $Put(v, s) = k$ are valid lenses.

It is important to note however, that even the invertibility laws leave room for a lot of undesirable lenses. For instance, the law $Put \cdot (Get \triangle id) \subseteq id$ only guarantees that, if the view is not changed, the source is not as well. But in the case when the view is changed, there is no restriction on the value that the source will take. In fact, the authors of the lens framework recognized this issue as well, stating that these laws are more of a guideline to correctness [Fos09].

A classic approach to the view-update problem is the constant complement technique from the database community [BS81]. In this case all information discarded by Get is stored in a complement and is then retrieved by Put. In the lens framework this can be achieved by enforcing the history-ignorance property, since it forces the second Put to have all information available to the first, in order to produce the same result. Lenses with this property are called *very well-behaved*, which is sometimes too strong, since, for instance, many transformation involving list manipulation would break this law.

In the case of maintainers, properties are typically much looser. In fact, the original definition [Mee98] only required the transformations to be consistent and hippocratic. It is possible however to define constraint maintainers with round-trip properties, as was presented.

3.3 State-of-the-art

A survey of the existing bidirectional approaches, which present some kind of formal guarantee, will now be presented. In the end, the classification of each technique, following the defined taxonomy, is presented in Table 3.1.

Mappings Wadler [Wad87] proposes the concept of *views*, a transformation that extends abstract data types with pattern matching and inductive reasoning. Views comprise two operations, *in* and *out*, that form isomorphisms between subsets of the two types. Recently, Wang et al. [WGMH11] proposed a new approach based on weaker version of the PF invertible language from [MHT04]. The system only guarantees right-invertibility, assuming that the source values may change at round-trip. However, while the transformations from [Wad87] are defined by the users, with no formal guarantee of correctness, in [WGMH11] the invertibility is enforced.

XSugar [BMS08] and biXid [KH06] are bidirectional languages for the transformation between XML and strings and different XML formats, respectively. The models are related by intertwined pairs of grammars, from which the forward and backward transformations are syntactically inferred. However, while in the former the grammars are checked for ambiguities, guaranteeing that the two transformations are bijections, up to an equivalence relation disregarding irrelevant details, in the latter no round-trip condition is guaranteed, since ambiguities play an essential role in their approach.

Kennedy [Ken04] proposes a combinatorial approach for the definition of picklers, where both the serializer and deserializer are defined by a single construction. However, by taking into account structure sharing, the round-trip properties do not completely hold, since serializing data with sharing may generate different strings.

Mu et al. [MHT04] proposed a PF injective language for structured editors, where the backward transformations are automatically calculated as the converse of a function. However, since duplication plays an essential role in the approach, due to data dependency in the views, only weaker versions of the round-trip properties hold. The framework uses tags to identify modified values in the view in order to preserve the injectivity.

Berdaguer et al. [BCPV07] propose a two-level data transformation system, where transformations applied to the data format are propagated to instances of that type. In particular, the 2LT system implements the transformation of XML schemas and their corresponding documents, and SQL language and the corresponding SQL databases. The transformations are built from PF combinators that define correct refinement steps.

Lenses The lens framework was first proposed by the Harmony group [FGM+07]. Arising from the view-update problem from databases, they define a domain specific programming language for bidirectional transformations on trees. The bidirectionalization technique is combinatorial, since transformations are built from bidirectional combinators. Lenses are required to obey the stability and acceptability laws, and optionally be history-ignorant. Although in the original definition lenses are partial, totality is considered an essential feature and the defined combinators are actually total. In [BPV06] Bohannon et al. propose a solution to the classical view-update problem of relational databases based on the lenses framework, with a particular focus on functional dependencies.

Later, the same team defined a bidirectional language for strings [BFP+08], with the goal of dealing with ordered data. In the *dictionary lenses*, models are considered to be equivalent up to reordering of "chunks", and thus the laws must be relaxed. In order to keep track of the "chunks" as the model evolves, they are annotated with keys. In [FPP08] these lenses for strings are relaxed by allowing correctness modulo a definable equivalence relation on the types. This is attained by the creation of *canonizers*, which can be composed with the lenses, mapping elements with their canonical representative. The same team later enhanced the lenses framework once again in order to deal with the problem of alignment [BCF+10]. Although evolving from the dictionary lenses from [BFP+08], they explicitly separate the alignment and operation decisions into two phases, providing a more generic framework and allowing the definition of different alignment strategies. The laws, however, remain state-based, not taking into consideration the behavior of the updates. In [FPZ09] the authors propose a different framework were lenses are defined with security in mind, by ensuring confidentiality and integrity.

Pacheco and Cunha [PC10] lifted most functional PF combinators to bidirectional lenses, including recursion patterns, resulting in a framework similar to that defined in [FGM+07] for trees, but with optimization of complex transformations. In [PC11] an equational calculus was built upon this PF framework, and thus several properties about the combinators could be analyzed and

proved, including properties for folds and unfolds, which allow a higher level of optimization.

The string lenses defined in [BFP+08] have also been implemented commercially in the Augeas system [Lut08], developed by Red Hat Linux. The lenses are used to implement the transformations between Linux configuration files and a more structured representation.

At the same time, another team has also been working on bidirectional systems in a framework similar to the lenses, albeit with weaker laws. Hu et al. [HMT08] proposed a bidirectional language for interactive structured editors. However, since data dependency is considered a key feature, duplication is allowed, meaning invertibility laws no longer hold. As such, the round-trip laws of the framework were weakened. An operation-based approach, it is assumed that a forward transformation is performed after every update, in order to identify which duplicated value was modified. Matsuda et al. [MHN+07] propose a bidirectionalization technique inspired by the constant complement. By constructing an injective function from the forward transformation, the backward transformation can be automatically calculated. A *view update checker* is also automatically generated, which filters out, in run-time, updated views that can not be correctly propagated to the source.

In later work by the same team, Liu et al. [LHT07] propose a bidirectionalization of the XML query language XQuery [BCF+05]. By allowing variable binding, and consequently duplication, the bidirectional properties must be relaxed, as in the previous work. Hidaka et al. [HHI+10] proposed an operation-based approach to bidirectionalization of graph transformation by providing a bidirectional semantic for UnCal, a graph algebra for the graph query language UnQl [BFS00]. Due to the possible partiality of the operations, only a weaker version of the stability property holds.

Voigtländer [Voi09] proposes a different approach that is purely semantic. Generic polymorphic forward transformations are passed to a high-order function, that semantically calculates in run-time the backward transformation, without any syntactic analysis of the forward transformation. In [VHMW10] the authors from the semantic technique [Voi09] and the syntactic technique in [MHN+07] combine the two approaches, shifting part of the syntactic analysis of the latter to run-time semantic analysis.

Diskin [Dis08] analyzed the properties of the lens framework under a generic setting, comparing the implications of different bidirectional properties. The totality requirements are relaxed, since the transformations are only required to be correct. Later, Diskin et al. [DXC10] propose an operation-based system as a different solution for the alignment problem. The backward transformation is divided in two steps: the first calculates the difference, or the update, between the views; the second propagates that update back to the source. Unlike [BCF+10], the updates are seen as first-class entities of the system, reflected in the fact that the round-trip laws are defined in terms of updates.

In the database community, Melnik et al. [MAB08] propose the bidirectionalization of mappings between entity-relationship models and the database. The bidirectional transformations are derived from the mapping, after checking if the mapping preserves round-tripping.

Maintainers Constraint maintainers were first proposed by Meertens [Mee98] in the context of constrained user interfaces. Transformations are defined in order to satisfy a consistency relation. Meertens defines several combinators that can be used to create such transformations, but many do not allow compositional reasoning about correctness. Later in the same article, the maintainers are extended to an operation-based setting, in order to deal with lists.

In [Ste10, Ste08], Stevens reasons about bidirectional properties for the QVT standard [Obj08] and how it can be represented by the constraint maintainers framework. In order to be correct, maintainers must possess the additional property of undoability.

In the maintainers definition of Diskin [Dis08], correct systems are slightly stronger than the ones from Meertens, with an additional domain restriction. Stevens' hippocratiness property is also defined and analyzed. In the same work, Diskin also proposes a third framework, called *trigonal synchronization systems*, that consist of operation-based maintainers. The transformations now take an update as an input. The motivation of such systems is incremental synchronization, since in certain situations there is no need to recalculate the whole model.

Ennals and Gay [EG07] propose the JT system, that is able to transform programs written in C and Jekyll in both directions. The transformations are non-deterministic, and take both the file in the original language and the updated file. Admittedly, the bidirectionality properties are not enforced, but authors only expect a "good enough" result.

Based on the idea of lenses and complements, Hofmann et al. proposed a framework for symmetric lenses [HPW11], where both sides may contain information that does not exist in the other. As interesting results, they prove that there does not exist the categorical split and either operators in the symmetric lens framework. Operators for more complex datatypes, like lists, are also presented, based on the definition of catamorphisms and anamorphisms. The resulting system is similar to a constraint maintainer where the consistency relation is disregarded, its role being played by the complement.

Other frameworks Other bidirectional approaches that do not fit in the defined frameworks have been proposed. In the graph community, the *triple graph grammars* (TGGs) created by Schürr [Sch94, SK08] have long been used to model simultaneous evolution of synchronized graphs. The technique consists of two graphs, connected by a third graph that defines the correspondences between the first two. When operations are performed, the three graphs co-evolve, preserving consistency. Later Konigs and Schürr [KS06] adapted the framework to situations where the graphs did not evolve at the same time: a graph is individually updated, and a posterior transformation updates the second graph to a consistent state. In [EEE+07], Ehrig analyzed the condition in which these transformations preserve certain round-trip properties. Although it might seem that TGGs would fit in the maintainers framework, the comparison is not so direct, since the gluing graph of TGGs is not really a consistency relation, as it co-evolves with the graphs.

Framework	Article	Updates	Total	Properties		Bidir.		Application
				Roundtrip	Other	Technique	Calculated	
$\rightleftharpoons$	[MHT04]	O_H	-	$\substack{\longrightarrow \\ \dashleftarrow}$		C	M	T
	[BMS08]	S	+	$\substack{\leadsto \\ \leftsquigarrow}$		S	S	S
	[KH06]	S	+	M		S	S	S
	[Wad87]	S	$\ominus$	$\substack{\longrightarrow \\ \longleftarrow}$		$\times$	U	T
	[Ken04]	S	$\ominus$	$\substack{\leadsto \\ \longleftarrow}$		C	U+M	T
	[WGMH11]	S	+	$\longrightarrow$		C	S	T
	[BCPV07]	S	$\oplus$	$\longleftarrow$		C	M	T
$\trianglerighteq$	[FGM+07]	S	+	$\substack{\longrightarrow \\ \longleftarrow}$		C	M	T
	[BPV06]	S	+	$\substack{\longrightarrow \\ \longleftarrow}$		C	M	R
	[BFP+08]	S	+	$\substack{\longrightarrow \\ \leftsquigarrow}$		C	M	S
	[FPP08]	S	+	$\substack{\leadsto \\ \leftsquigarrow}$		C	M	S
	[BCF+10]	O_C	+	$\substack{\boldsymbol{\longrightarrow} \\ \boldsymbol{\longleftarrow}}$		C	M	S
	[PC10]	S	+	$\substack{\longrightarrow \\ \longleftarrow}$		C	M	T
	[HMT08]	O_H	$\ominus$	$\substack{\dashrightarrow \\ \dashleftarrow}$		C	M	T
	[MHN+07]	S	$\ominus$	$\substack{\longrightarrow \\ \longleftarrow}$	$\curvearrowleft$,$\circlearrowleft$	S	C+s	T
	[LHT07]	O_H	$\ominus$	$\boldsymbol{\longleftarrow}$		C	M+s	T
	[HHI+10]	O_H	$\ominus$	$\substack{\longrightarrow \\ \dashrightarrow}$		C	M+s	G
	[Voi09]	S	$\ominus$	$\substack{\longrightarrow \\ \longleftarrow}$	$\curvearrowleft$,$\circlearrowleft$	s	s	T
	[VHMW10]	S	$\ominus$	$\substack{\longrightarrow \\ \longleftarrow}$		S+s	C+s	T
	[Dis08]	S	$\oplus$	$\substack{\longrightarrow \\ \longleftarrow}$		$\times$	$\times$	$\times$
	[DXC10]	O_C	$\oplus$	$\boldsymbol{\longleftarrow}$	$\circlearrowleft$	$\times$	$\times$	$\times$
	[MAB08]	O_H	+	$\longrightarrow$		S	S	R
$\lesseqgtr$	[Mee98]	S	T		$\downdownarrows$,R	C	M	R
	[Mee98]	O_H	T		$\downdownarrows$,R	C	M	T
	[Ste10]	S	+		$\curvearrowleft$,$\downdownarrows$,R	C	M	G
	[Dis08]	S	$\oplus$	$\substack{\longrightarrow \\ \longleftarrow}$	R	$\times$	$\times$	$\times$
	[EG07]	S	+	M		$\times$	S	T
	[HPW11]	S	+	$\substack{\longrightarrow \\ \longleftarrow}$		C	M	T

Table 3.1: A summary of bidirectional transformation approaches.

Chapter 4

RAs and Bidirectional Transformations

The high level of freedom of the calculus in RAs also enables easier and simpler reasoning about the properties of the systems and derivation of inverses. In this Chapter we take advantage of those characteristics and apply the calculus, first, in the redefinition of an existing framework (the quotient lenses) under the relational setting, demonstrating the simplicity by presenting some proofs; second, in the analysis of possible techniques for the automatic derivation of correct backward transformations.

4.1 Redefining Quotient Lenses using RAs

RAs can be used to express most existing approaches. Such shift in the underlying framework provides not only the simplicity of calculus associated with the PF notation, but also allows the comparison of previously incomparable approaches. As an example, we will now show how RAs can be used to redefine and simplify the quotient lens framework [FPP08].

Quotient lenses laws are defined as a relaxation of the original round-trip properties, in order to allow equivalences modulo equivalence relations. The goal is to allow transformations to be well-behaved up to inessential details, like whitespaces. Equivalence relations define which values should be considered equivalent when checking the properties. For a lens $l : S \trianglerighteq V$, with equivalence relations $\sim_S$ and $\sim_V$ on S and V respectively, the round-trip properties of quotient lenses can simply be defined as:

$$\begin{aligned} get \cdot put &\subseteq \sim_V \cdot \pi_1 \\ put \cdot (get \triangle id) &\subseteq \sim_S \\ create \cdot get &\subseteq \sim_S \end{aligned}$$

Additional laws guaranteeing that the operations treat equivalent elements equiv-

alently are also needed:

$$\begin{aligned} get \cdot \sim_S &\subseteq \sim_V \cdot get \\ put \cdot (\sim_V \times \sim_S) &\subseteq \sim_S \cdot put \\ create \cdot \sim_V &\subseteq \sim_S \cdot create \end{aligned}$$

With these laws we are able to trivially prove the well-behavedness of the quotient lenses presented in the original work. As an example, let's consider the *lquot* combinator. A *canonizer* is a structure that bidirectionally transforms elements into its canonical representative, comprising functions *canonize* and *choose*, such that:

$$canonize \cdot choose \subseteq \sim_A \tag{4.1}$$

As such, by definition, any equivalence relation $\sim_C$ can be derived by a canonizer:

$$\sim_C = canonize^\circ \cdot \sim_A \cdot canonize \tag{4.2}$$

The combinator *lquot* takes a canonizer and a lens and relaxes the original source domain, giving origin to the following lens:

$$\begin{aligned} get &= get_l \cdot canonize \\ put &= choose \cdot put_l \cdot (id \times canonize) \\ create &= choose \cdot create \end{aligned}$$

As an example of the PF calculus, we present the proof of the acceptability law (where *canonize* and *choose* are abbreviated to *can* and *cho* respectively):

$$\begin{aligned} & put_l \cdot (get_l \triangle id) \subseteq \sim_A \\ \Rightarrow & \{\cdot \text{ - MONOTONICITY}\} \\ & \sim_A \cdot put_l \cdot (get_l \triangle id) \cdot can \subseteq \sim_A \cdot can \\ \Rightarrow & \{\subseteq \text{ - TRANSITIVITY}, (4.1), \triangle \text{ - FUSION}\} \\ & can \cdot cho \cdot put_l \cdot (get_l \cdot can \triangle can) \subseteq \sim_A \cdot can \\ \Leftrightarrow & \{\text{ SHUNTING}, \triangle \text{ - ABSORPTION}\} \\ & cho \cdot put_l \cdot (id \times can) \cdot (get_l \cdot can \triangle id) \subseteq can^\circ \cdot \sim_A \cdot can \\ \Leftrightarrow & \{(4.2)\} \\ & cho \cdot put_l \cdot (id \times can) \cdot (get_l \cdot can \triangle id) \subseteq \sim_S \end{aligned}$$

4.2 Calculating Backward Transformations

Like has already been stated, the round-trip laws of the lens framework are more of a guideline than actually rules for the creation of the backward transformation. Even the authors recognized this, by stating that they are "intended more as a loose guide than a specification of correctness" [Fos09].

As they are, the laws allow the definition of *Put* operations that do not produce the desired outcome. For example, consider the lens π_1. The expected behavior of the *Put* operation is to select the updated view as the new first element, and retrieve the second element from the original source, defined as:

$$Put(a', (a, b)) = (a', b)$$

Or the PF equivalent $id \times \pi_2$. However, the acceptability property $Put(v, Get\, v) = v$ only enforces round-tripping when the view is not modified. As such, the following definition of Put is also well-behaved:

$$Put(a', (a, b)) = \begin{cases} (a', b) & \text{if } a = a' \\ (a', \bot) & \text{if } a \neq a' \end{cases}$$

Since the lens framework is combinatorial, this kind of problems does not occur, since the combinators are defined with the desired behavior by the developer. This is confirmed by Table 3.1, where most combinatorial approaches have the transformations defined manually. However, nothing guarantees that the provided definition is in fact the "best" one, considering that a "good" backward definition is one that preserves as much information from the original source as possible.

A possible solution, proposed in [HMT08] is to define an ordering $\preceq$ on views defining how "updated" they are. By enforcing that $v \preceq Get(Put(v, s))$, guaranteeing that Put and Get are update preserving, some of the absurd Put definitions are avoided. However, calculating the correct transformation is still not possible, unless we define an even stronger imposition.

Given the "update" orders $\sqsubseteq : S \to S$ on sources and $\leq : V \to V$ on views, the transformations are optimal if the following law holds:

$$(Get \triangle id)^\circ \cdot (\leq \times \sqsubseteq) = \sqsubseteq \cdot Put \cap (Get^\circ \cdot \leq \cdot Get \cdot \pi_2) \tag{4.3}$$

The law guarantees multiple properties. First, it guarantees that a source s' is more "updated" than a source s if, and only if, their views are as well:

$$s \sqsubseteq s' \Leftrightarrow Get\, s \leq Get\, s'$$

Second, given the same sources s and s', and an updated view view v', the backward transformation $Put\,(v', s')$ is always more "updated" than s, i.e.:

$$s \sqsubseteq s' \wedge Get\, s \leq v' \Rightarrow s \sqsubseteq Put\,(v', s')$$

Lastly, if s' is more "updated" than s, and the backward transformation $Put\,(v', s')$ results in a source more updated than s, then v' must be more updated than the view of s:

$$s \sqsubseteq s' \wedge s \sqsubseteq Put\,(v', s') \Rightarrow Get\, s \leq v'$$

With this particular law, it is possible to calculate the correct Put operation for the π_1 lens, assuming that $\sqsubseteq = \leq \times \leq$. However, this approach has also a limitation: the relation between the two orders must be defined, which may not always be possible. Even in this case, although we defined the order on pairs as the product of the order on the elements, that is not necessarily the best definition.

Another technique that has been studied is based in the view complement approach from [MHN+07]. The main idea is to define a complement view function $Comp : S \to C$ such that the split $Get \triangle Comp$ is an injective function. In such case, the Put could be automatically calculated as the inverse of that operation as:

$$(Get \triangle Comp)^\circ \cdot (id \times Comp) \subseteq Put$$

In this case, calculating the best backward transformation consists of calculating the best complement view function, since Put directly depends of $Comp$. Since the goal is to achieve injectivity, we want $Comp$ to be as injective as possible. The level of injectivity of a function is defined by its kernel, and thus the injectivity order $\preceq$ can be defined as:

$$R \preceq S \equiv \ker S \subseteq \ker R$$

Such order has already been studied in the relational PF calculus, and possesses very interesting properties [Oli05].

With this order, we can define the desired property that guarantees that $Comp$ is as injective as possible:

$$\langle \forall R : (Get, R)^\circ \cdot (id \times R) \subseteq Put : R \leq Comp \rangle$$

This law states that any other complement view R that fits the definition of Put is less injective than $Comp$. A direct consequence of this framework is that, if S is isomorphic to $V \times C$, then the transformations are ideal. Considering the π_1 lens, if $Comp = \pi_2$ we achieve the desired isomorphism, which directly defines Put as the expected $id \times \pi_2$. Another example is the product lens $f \times g$. Since the ideal view complement for a product $A \times B$ is $Comp_A \times Comp_B$, calculating the Put operation results in $(Put_f \times Put_g) \cdot distp$, as expected.

An issue with this approach is that it involves a second-order quantification on the law, which complicates the calculations. We are still studying other ways to express this property without resorting to such quantifications.

Chapter 5

Relational lenses

Invariants play an important role in program construction and verification, since they provide a more refined way to predict the behavior of the system. In particular, state invariants, conditions that are defined on the domains of application, can be seen as an enhancement of the data types and an extension of the standard type system, resulting in what is known as *extended static checking*.

Moreover, in the case of lenses the existence of invariants allows the definition of combinators that can not be defined in a purely functional setting, since invariants will restrict the domains to subsets where the transformations are known to be well-behaved. Such is the case of the duplication of data operation $id \triangle id$, or the McCarthy's conditional operator, essential constructions for a useful programming language. By introducing as primitives operators that needed an ad hoc definition in previous frameworks, the expressiveness of the language is greatly improved, resulting in a more complete language, able to encode general functional programming languages.

Let's use the duplication of data, a well known problem of bidirectional transformations, as an example to roughly present our framework. In fact, the lens framework does not have a categorical product, and the split operator $f \triangle g$ is not a well-behaved lens in general (although the product operator $f \times g$ is). It is simple to see why data duplication does not preserve the acceptability law. Let's assume a variable a is duplicated into (a, a), and then the first is changed to a'. The *put* operation would either return first value of the pair a' or the second element a. Applying *get* again would either return (a, a) or (a', a'), breaking the acceptability property. In fact, since the split operator is usually not surjective, it could never form a lens.

Different authors have proposed different solutions to overcome this limitation. In [MHT04] and [HMT08] the round-trip laws are weakened in order to allow duplication, being able to preform the backward transformation if only one value is changed, or if they are equal to each other. The laws were also weakened in [LHT07], where variable duplication is implicitly allowed due to multiple variable occurrences. In [FPP08] data duplication is allowed by relaxing round-trip properties up to an equivalence relation, but the lens is valid only for the string framework. In [PC10] the split operator $f \triangle g$ is defined in case the following side condition holds:

$$put_f \cdot (id \times put_g) \cdot assocr = put_g \cdot (id \times put_f) \cdot assocr \cdot (swap \times id)$$

Where $assocr$ is the isomorphism $(A \times B) \times C \to A \times (B \times C)$. Examples of such lenses are the swap operation $\pi_2 \triangle \pi_1$ and the product operator $f \times g$.

Techniques where the domain of the application is restricted have already been proposed. In [MHN+07], a *view update checker* is generated for each transformation, that checks on run-time if the views are valid, i.e., the backward transformation correctly propagates the updates. However, they still require variable occurrences to be *affine*, i.e., they can appear only once in a transformation, and thus data duplication is not allowed.

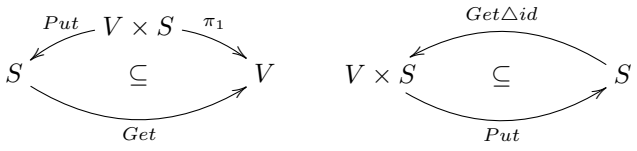
The generalization of the calculus to a relational setting provides a framework where partial and non-deterministic operations, as well as invariants on the domains of the transformations, arise as natural artifacts. Extended static checking in RAs has already been studied in [Oli09], providing a simple technique for the verification of state-invariants. Such invariants are defined as coreflexives, restricting the possible values. In the case of lenses, domains are restricted to values that guarantee the correctness of the transformation, and implicitly obtain total and surjective operations.

As a result we are able to define arbitrary split operators, since each split operator will induce an invariant on the generated pair, which must be preserved in order to enable a correct backward transformation. For instance let's consider the split $len \triangle head : [A] \trianglerighteq N \times A$ that calculates the length and the head of a lists. This operation would induce an invariant on the type $N \times A$ stating that a pair (n, a) is valid only if there is a list l such that $len\ l = n$ and $head\ l = a$, that is, $n > 0$. In such case, the backward transformation could obvious return that l.

In this Chapter a novel framework for relational lenses enhanced with invariants is presented, which will be denoted by *r-lenses*. First, the properties which a r-lens must possess are analyzed and defined. Then, the basic r-lenses are defined, followed by the product operators, including the split combinator. In the end, the remaining lenses are presented, including the constant combinators and the coproduct r-lenses.

5.1 Defining R-Lenses

In [Oli07], the author makes an interesting remark that total well-behaved lenses meet the connectivity requirements of two refinement/abstraction pairs. In particular, the following diagrams are true:



[Oli07] also shows that these diagrams are in fact equivalent to the round-tripping properties with equalities in place of inclusion:

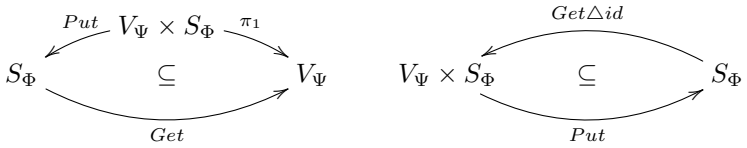
$$Get \cdot Put = \pi_1$$
$$Put \cdot (Get \triangle id) = id$$

Note, however, that total well-behaved lenses are still stronger: the diagrams merely require that Put is left- and right-total, which does not imply that it is

"fully" total.

A *relational lens*, or *r-lens*, is a lens where the transformations are no longer required to be total functions. As such, the combinators will only be well-behaved for a defined subset of the domain of the transformations, which is defined by coreflexives that filter out invalid values. For simplification purposes, we will refer to lens operations assuming their respective invariants. For example, when we say that "*Get* is total" we actually mean to say that "*Get* is total for the values that satisfy the invariants".

Since the well-behavedness of classic lenses is equivalent to defining the transformations as refinement/abstraction patterns, the round-trip properties of r-lens should also emerge from the respective generalized diagrams. Consider two coreflexives $\Phi : S \to S$ and $\Psi : V \to V$ that represent the invariants on sources and views respectively. By restricting the types of the transformation, the previous diagrams can be directly generalized as:



defined by the following laws [Oli08]:

$$\text{img}(Get \cdot \Phi) = \Psi \qquad (Get \text{ simple and surjective}) \tag{5.1}$$

$$\ker(Put \cdot (\Psi \times \Phi) \cdot \pi_1^\circ) = \Psi \qquad (Put \text{ left-total and left-injective}) \tag{5.2}$$

$$Put \cdot (\Psi \times \Phi) \subseteq (Get \cdot \Phi)^\circ \cdot \pi_1 \qquad (Put \cdot \pi_1^\circ \text{ and } Get \text{ connected}) \tag{5.3}$$

$$\text{img}(Put \cdot (\Psi \times \Phi)) = \Phi \qquad (Put \text{ simple and surjective}) \tag{5.4}$$

$$\ker((Get \triangle id) \cdot \Phi) = \Phi \qquad (Get \triangle id \text{ total and injective}) \tag{5.5}$$

$$(Get \triangle id) \cdot \Phi \subseteq (Put \cdot (\Psi \times \Phi))^\circ \qquad (Put \text{ and } Get \triangle id \text{ connected}) \tag{5.6}$$

As before, the above laws entail the round-tripping properties (with equalities) for r-lenses:

$$Get \cdot Put \cdot (\Psi \times \Phi) = \Psi \cdot \pi_1 \tag{5.7}$$

$$Put \cdot (Get \triangle id) \cdot \Phi = \Phi \tag{5.8}$$

Unfortunately, they are implying subtly more. Although it is reasonable for *Get* to be an abstraction, and consequently simple, so that it uniquely determines a view, we would like *Put* to remain non-deterministic, to account for the different possible view updates. Nevertheless, expressing the stability property as an abstraction diagram forces *Put* to uniquely determine a view update (law (5.4)), and can be seen as an over-specification. We can relax this condition by requiring only the weaker:

$$\text{img}(Put \cdot (\Psi \times \Phi)) \supseteq \Phi \qquad (Put \text{ surjective})$$

but making *Put* non-simple also implies that the stability law (5.8) is no longer a consequence of the diagram's connectivity. As such, since $Put/Get \triangle id$ does not form a valid abstraction pair, the stability of r-lenses will be directly imposed by the slightly stronger law (5.8).

The pair $Get/Put \cdot \pi_1^\circ$ is in fact an abstraction pair on total well-behaved r-lenses (which guarantees that the view type V_Ψ contains less information that the source type S_Φ and is crucial for proper view-updating), implying acceptability. As a side-effect of (5.8) however, (5.2) happens to become redundant. As such, total well-behaved r-lenses are defined by laws (5.1), (5.3) and (5.8).

These three laws are equivalent to the following set of more "intuitive" laws, that allow a simpler reasoning about the lens properties:

Invariant preservation laws that guarantee that Get and Put preserve the invariants, which consist of checking that the transformations fall into the pre- and post-conditions [Oli09], i.e., the invariants on the types:

$$Get \cdot \Phi \subseteq \Psi \cdot Get \tag{5.9}$$
$$Put \cdot (\Psi \times \Phi) \subseteq \Phi \cdot Put \tag{5.10}$$

Round-tripping laws that guarantee that the lens is an abstraction and preserves empty source updates, which are similar to the original round-trip laws, but restricted by the invariants:

$$Get \cdot Put \cdot (\Psi \times \Phi) \subseteq \Psi \cdot \pi_1 \tag{5.11}$$
$$Put \cdot (Get \triangle id) \cdot \Phi \subseteq \Phi \tag{5.12}$$

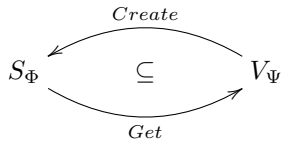
Totality laws that guarantee that a r-lens is total:

$$\Phi \subseteq \delta Get \tag{5.13}$$
$$\Psi \times \Phi \subseteq \delta Put \tag{5.14}$$

Where the first two guarantee r-lens well-behavedness and the third one that it is total. Note that the laws only define the behavior of the operation for values inside the respective invariants; if the input of the operations is outside the invariant, the behavior is unknown. Summarizing, for total well-behaved r-lenses we have:

- Get total, simple and surjective;
- Put total, left-injective and surjective.

In situations when performing the backward transformation is not possible, a default transformation is applied that generates an arbitrary valid value, denoted by $Create$. A similar rationale was applied to this operation, which is represented by the following refinement diagram:



The laws entailed by the diagram are (this time) all desired properties:

$$\mathrm{img}(Get \cdot \Phi) = \Psi \qquad (Get \text{ simple and surjective}) \tag{5.15}$$
$$\ker(Create \cdot \Psi) = \Psi \qquad (Create \text{ total and injective}) \tag{5.16}$$
$$Create \cdot \Psi \subseteq (Get \cdot \Phi)^\circ \qquad (Create \text{ and } Get \text{ connected}) \tag{5.17}$$

$id : A_\Phi \trianglerighteq A_\Phi$
$\cdot : (C_\Omega \trianglerighteq B_\Phi) \to (B_\Phi \trianglerighteq A_\Psi) \to (C_\Omega \trianglerighteq A_\Psi)$
$^\Theta : (A_\Phi \trianglerighteq B_\Psi) \to (A_{\Phi \cap \delta(\Theta \cdot f)} \trianglerighteq B_{\Psi \cap \Theta})$
$_\Omega : (A_\Phi \trianglerighteq B_\Psi) \to (A_{\Phi \cap \Omega} \trianglerighteq B_{\Psi \cap \rho(f \cdot \Omega)})$
$\pi_1 : (A \times B)_{\prod_R} \trianglerighteq A_{\delta R}$
$\pi_2 : (A \times B)_{\prod_R} \trianglerighteq B_{\rho R}$
$\triangle : (A_\Phi \trianglerighteq B_\Omega) \to (A_\Phi \trianglerighteq C_\Lambda) \to (A_\Phi \trianglerighteq (B \times C)_{\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega}})$
$\times : (A_\Phi \trianglerighteq C_\Omega) \to (B_\Psi \trianglerighteq D_\Lambda) \to ((A \times B)_{\Phi \times \Psi} \trianglerighteq (C \times D)_{\rho(\Omega \cdot f \cdot \Phi) \times \rho(\Lambda \cdot g \cdot \Psi)})$
$i_1 : A_\Phi \trianglerighteq (A + B)_{\Phi + id}$
$i_2 : B_\Psi \trianglerighteq (A + B)_{id + \Psi}$
$\nabla : (A_\Phi \trianglerighteq C_\Omega) \to (B_\Psi \trianglerighteq C_\Lambda) \to ((A + B)_{\Phi + \Psi} \trianglerighteq C_{\Omega \cup \Lambda})$
$+ : (A_\Phi \trianglerighteq C_\Omega) \to (B_\Psi \trianglerighteq D_\Lambda) \to ((A + B)_{\Phi + \Psi} \trianglerighteq (C + D)_{\Omega + \Lambda})$
$\Phi? : A_\Psi \trianglerighteq (A + A)_{\Phi \cap \Psi + \Phi_\neg \cap \Psi}$
$! : A_\Phi \trianglerighteq 1$
$\underline{a} : A_\Phi \trianglerighteq A_{\rho \underline{a}}$

Table 5.1: Defined and expected r-lenses.

which is equivalent to showing that:

$$\begin{aligned} Create \cdot \Psi &\subseteq \Phi \cdot Create \\ Get \cdot Create \cdot \Psi &\subseteq \Psi \\ \Psi &\subseteq \delta Create \end{aligned}$$

However, we are defining $Create$ as a non-deterministic operation, in order to be able to retrieve all possible valid values. As such, it is simply defined as the inverse of Get, restricted by the respective invariant to filter out invalid values:

$$Create \triangleq (Get \cdot \Phi)^\circ = \Phi \cdot Get^\circ \tag{5.18}$$

This definition trivially satisfies the laws (5.15)-(5.17), and so $Create$ requires no additional proofs. Note that the restriction by the invariant is needed because it is not guaranteed that all valid outputs of Get are produced by valid inputs, i.e., $\Psi \cdot Get \subseteq Get \cdot \Phi$ does not necessarily hold.

5.2 Basic R-Lenses

The primitives and combinators needed in order to obtain a complete framework will now be defined. Table 5.1 presents the already defined and proved to be correct r-lens on top, and the remaining lenses that have not been checked for correctness yet, but whose shape can already be inferred, on bottom.

The simplest lens is the identity, which can be trivially lifted to the r-lens $id : A_\Phi \trianglerighteq A_\Phi$:

$$\boxed{\begin{array}{c} id : A_\Phi \trianglerighteq A_\Phi \\ \\ Get = id \\ Put = \pi_1 \\ Create = id \end{array}}$$

Proving that the identity is in fact a r-lens is trivial for all laws.

Next, we will analyze the composition of r-lenses. In the functional setting, the composition of two lenses $f : B \trianglerighteq A$ and $g : C \trianglerighteq B$, $f \cdot g : C \trianglerighteq A$ is also a lens defined as:

$$\begin{aligned} get &= get_f \cdot get_g \\ put &= put_g \cdot (put_f \cdot (id \times get_g) \triangle \pi_2) \\ create &= create_g \cdot create_f \end{aligned}$$

In the case when the pre-condition of f is exactly the post-condition of g, the functional lens can be trivially lifted to a r-lens $f \cdot g : C_\Omega \trianglerighteq A_\Psi$ with similar operations:

$$\frac{f : B_\Phi \trianglerighteq A_\Psi \qquad g : C_\Omega \trianglerighteq B_\Phi}{f \cdot g : C_\Omega \trianglerighteq A_\Psi}$$

$$\begin{aligned} Get &= Get_f \cdot Get_g \\ Put &= Put_g \cdot (Put_f \cdot (id \times Get_g) \triangle \pi_2) \\ Create &= Create_g \cdot Create_f \end{aligned}$$

Since the invariants are equivalent, the lens transforms all values in Ω to values in Ψ. Since this is the first r-lens presented, the full proof of the properties will be shown. For the rest of the lenses, the proofs are delegated to Appendix A. For simplification purposes, during the proofs Get_f will be denoted simply by the lens name f.

Proof. Let's start by proving that Get preserves the invariant, i.e., $Get_f \cdot Get_g \cdot \Omega \subseteq \Psi \cdot Get_f \cdot Get_g$:

$$\begin{aligned} & \{f, g \text{ lenses}\} \\ & g \cdot \Omega \cdot g^\circ \subseteq \Phi \wedge \Phi \subseteq f^\circ \cdot \Psi \cdot f \\ \Rightarrow & \{\subseteq \text{ - TRANSITIVITY}\} \\ & g \cdot \Omega \cdot g^\circ \subseteq f^\circ \cdot \Psi \cdot f \\ \Leftrightarrow & \{\text{SHUNTING}\} \\ & f \cdot g \cdot \Omega \subseteq \Psi \cdot f \cdot g \end{aligned}$$

Next, proving that the domain of Get is total for the invariant, $\Omega \subseteq \delta(Get_f \cdot Get_g)$:

$$\begin{aligned} & \{f, g \text{ lenses}\} \\ & \Omega \subseteq g^\circ \cdot \Phi \cdot g \wedge \Phi \subseteq f^\circ \cdot \Psi \cdot f \\ \Rightarrow & \{\subseteq \text{ - TRANSITIVITY}, \Phi \text{ - LARGER}\} \\ & \Omega \subseteq g^\circ \cdot f^\circ \cdot f \cdot g \\ \Rightarrow & \{\cap \text{ - MONOTONICITY}, \delta \text{ - DEF}\} \\ & \Omega \subseteq \delta(f \cdot g) \end{aligned}$$

Next, we prove that Put preserves the invariant, $Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2) \cdot$

$(\Psi \times \Omega) \subseteq \Omega \cdot Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2)$:

$$
\begin{aligned}
& Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2) \cdot (\Psi \times \Omega) \\
= & \{\triangle \text{ - FUSION}, \triangle \text{ - ABSORPTION}\} \\
& Put_g \cdot (Put_f \cdot (\Psi \times g \cdot \Omega) \triangle \Omega \cdot \pi_2 \cdot \delta(\Psi \cdot \pi_2)) \\
\subseteq & \{\Phi \text{ - LARGER}, g \text{ lens}\} \\
& Put_g \cdot (Put_f \cdot (\Psi \times \Phi \cdot g) \triangle \Omega \cdot \pi_2) \\
\subseteq & \{f, g \text{ lenses}\} \\
& \Omega \cdot Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2)
\end{aligned}
$$

And finally, the totality of Put, $(\Psi \times \Omega) \subseteq \delta(Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2))$. We start by simplifying the right-hand side of the inequation:

$$
\begin{aligned}
& \delta(Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2)) \\
\supseteq & \{\delta \text{ - COMPOSITION}, g \text{ lens}\} \\
& \delta((\Phi \times \Omega) \cdot (Put_f \cdot (id \times g) \triangle \pi_2)) \\
\supseteq & \{\triangle \text{ - ABSORPTION}, f \text{ lens}\} \\
& \delta(Put_f \cdot (\Psi \times \Phi) \cdot (id \times g) \triangle \Omega \cdot \pi_2) \\
\supseteq & \{\delta \text{ - COMPOSITION}, f \text{ lens}\} \\
& \delta((\Psi \times \Phi) \cdot (id \times g) \triangle \Omega \cdot \pi_2) \\
\supseteq & \{\triangle \text{ - ABSORPTION}, g \text{ lens}\} \\
& \delta((id \times g) \cdot (\Psi \times \Omega) \triangle \Omega \cdot \pi_2) \\
= & \{\triangle \text{ - DOMAIN}\} \\
& \delta((id \times g) \cdot (\Psi \times \Omega)) \cap \delta(\Omega \cdot \pi_2)
\end{aligned}
$$

As such, by ($\subseteq$ - TRANSITIVITY) and ($\cap$ - UNIVERSAL), it suffices to prove:

$$
\begin{aligned}
& \Psi \times \Omega \subseteq \delta((id \times g) \cdot (\Psi \times \Omega)) \wedge \Psi \times \Omega \subseteq \delta(\Omega \cdot \pi_2) \\
\Leftrightarrow & \{\Phi \text{ - IN DOMAIN/RANGE}, \ker \text{ - DEF}, \text{SHUNTING}\} \\
& (id \times g) \cdot (\Psi \times \Omega) \subseteq (id \times g) \cdot (\Psi \times \Phi) \wedge \pi_2 \cdot (\Psi \times \Omega) \subseteq \Omega \cdot \pi_2 \\
\Leftrightarrow & \{\triangle \text{ - CANCELLATION}, \Phi \text{ - SMALLER}\} \\
& true
\end{aligned}
$$

Regarding the round-trip properties, we will resort to the following lemma:

Lemma 5.1. *Given a well-behaved functional lens, if the operations are lifted to the relational setting and they preserve the invariant, the corresponding r-lens is also well-behaved.*

Proof. The lemma is trivially proved. Starting with stability:

$$
\begin{aligned}
& Put \cdot (Get \triangle id) \subseteq id \\
\Rightarrow & \{\cdot \text{ - MONOTONICITY}\} \\
& Put \cdot (Get \triangle id) \cdot \Phi \subseteq \Phi
\end{aligned}
$$

Followed by acceptability:

$$
\begin{aligned}
& Get \cdot Put \subseteq \pi_1 \\
\Rightarrow & \{\cdot \text{ - MONOTONICITY}\} \\
& \Psi \cdot Get \cdot Put \subseteq \Psi \cdot \pi_1 \\
\Rightarrow & \{\text{Invariant preservation}, \subseteq \text{ - TRANSITIVITY}\} \\
& Get \cdot \Phi \cdot Put \subseteq \Psi \cdot \pi_1 \\
\Rightarrow & \{\text{Invariant preservation}, \subseteq \text{ - TRANSITIVITY}\} \\
& Get \cdot Put \cdot (\Psi \times \Phi) \subseteq \Psi \cdot \pi_1
\end{aligned}
$$

□

Following Lemma 5.1, since the operations of the r-lens composition combinator are lifted from their functional counter-parts, it is also stable and acceptable. □

In order to make the composition more generic, it is desired that any two lenses can be combined, including those whose invariants do not match. There exist two interesting r-lenses that restrict the domains of the operations with stronger invariants. Although useful by themselves, these lenses also enable the composition of lenses with different invariants, by casting them down to similar coreflexives. The first one, which will be denoted by *v-restrict*, for an existing r-lens $f : A_\Phi \trianglerighteq B_\Psi$, given a coreflexive $\Theta : B \to B$, further restricts the view's invariant as $f^\Theta : A_{\Phi \cap \delta(\Theta \cdot f)} \trianglerighteq B_{\Psi \cap \Theta}$:

$$
\frac{f : A_\Phi \trianglerighteq B_\Psi \qquad \Theta : B \to B}{f^\Theta : A_{\Phi \cap \delta(\Theta \cdot f)} \trianglerighteq B_{\Psi \cap \Theta}}
$$

$$
\begin{aligned}
Get &= \Theta \cdot Get_f \\
Put &= Put_f \\
Create &= Create_f
\end{aligned}
$$

The invariant of the sources in this new lens is also restricted to values from which Get_f reaches values in Θ. The Get transformation of this lens simply restricts the values to belong to Θ. An interesting property that is essential to validate this lens is the following, that defines how the restriction of the invariant on the views is reflected in the Put operation:

Lemma 5.2. *For a r-lens $f : A_\Phi \trianglerighteq B_\Psi$, if the invariant on the view is further restricted by a coreflexive Ω, the range of Put is restricted to sources whose views are in Ω,i.e., $\delta(\Omega \cdot get)$:*

$$put \cdot ((\Psi \cap \Omega) \times \Phi) \subseteq (\Phi \cap \delta(\Omega \cdot get)) \cdot put$$

The dual combinator *s-restrict* takes a r-lens $f : A_\Phi \trianglerighteq B_\Psi$ and a coreflexive $\Omega : A \to A$ and further restricts the source's invariant as $f_\Omega : A_{\Phi \cap \Omega} \trianglerighteq B_{\Psi \cap \rho(f \cdot \Omega)}$:

$$\frac{f : A_\Phi \trianglerighteq B_\Psi \qquad \Omega \supseteq \rho(Put_f)}{f_\Omega : A_{\Phi\cap\Omega} \trianglerighteq B_{\Psi\cap\rho(f\cdot\Omega)}}$$

$$\begin{aligned} Get &= Get_f \\ Put &= \Omega \cdot Put_f \\ Create &= \Omega \cdot Create_f \end{aligned}$$

The additional condition over Ω is needed to guarantee the totality of Put. The rationale of the combinator is dual to the previous one. This time, the values produced by Put are restricted to those belonging to Ω.

With these two operators, composition of r-lenses with different invariants can be simply defined. For two lenses $f : B_\Phi \trianglerighteq A_\Psi$ and $g : C_\Omega \trianglerighteq B_\Lambda$, provided that $\rho(Put_f) \subseteq \Lambda$, we can restrict the invariants that need to match in the composition to $\Phi \cap \Lambda$, and thus compose them as:

$$f_\Lambda \cdot g^\Phi : C_{\Omega\cap\delta(\Phi\cdot g)} \trianglerighteq A_{\Psi\cap\rho(f\cdot\Lambda)}$$

In the case when $\Phi \subseteq \Lambda$ or $\Lambda \subseteq \Phi$, the composition can be further simplified to $f \cdot g^\Phi$ and $f_\Lambda \cdot g$ respectively.

5.3 Product of R-Lenses

In this Section we show that with the domains restricted to invariants, it is possible to define the split operation, and thus data duplication, as a valid r-lens.

5.3.1 Invariants on Pairs

In order to define the product operators, there is the need to define invariants on pairs. Since every pair is constructed by a split operation, it can be assumed that invariants on pairs are defined by the range of a split that generated them. From ($\triangle$ - RANGE) the range of a split operation $\rho(R \triangle S)$ is $(\pi_1^\circ \cdot R \cdot S^\circ \cdot \pi_2) \cap id$. The expression $R \cdot S^\circ$ is defined for values in A and B that reach the same value in C by R and S respectively. As such, invariants on pairs can always take the shape:

$$(\pi_2^\circ \cdot R \cdot \pi_1) \cap id = (\pi_1^\circ \cdot R^\circ \cdot \pi_2) \cap id$$

Which will be denoted by $\prod_R$, where $R : A \to B$ defines the validity relation between elements of A and B that are contained in the invariant.

If we see the invariant as the decomposed expression $(\pi_2^\circ \cdot \rho R \cdot R \cdot \delta R \cdot \pi_1) \cap id$, the coreflexives δR and ρR define the individual invariants on A and B respectively. In the case when there is no relation between A and B, but there are invariants Φ and Ψ on each type, respectively, the coreflexive can be simplified as:

$$(\pi_2^\circ \cdot \Phi \cdot \top \cdot \Psi \cdot \pi_1) \cap id = \Phi \times \Psi$$

The cancellation laws for the product invariant, that will be useful in the analysis of the r-lenses, are:

$$\pi_1 \cdot \prod_R = (id \triangle R)^\circ \wedge \pi_2 \cdot \prod_R = (R^\circ \triangle id)^\circ \tag{5.19}$$

Also, the invariants created from disjoint relations is empty:

$$\prod_{R} \cap \prod_{\overline{R}} = \bot \tag{5.20}$$

5.3.2 The π R-Lenses

The first primitives on the products are the projections. The $\pi_1 : (A \times B)_{\prod_R} \trianglerighteq A_{\delta R}$ r-lens is defined for any pair related by R, and is defined by:

$$\begin{array}{c} \pi_1 : (A \times B)_{\prod_R} \trianglerighteq A_{\delta R} \\ \\ Get = \pi_1 \\ Put = \prod_R \cdot (id \times \pi_2) \cup Create \cdot \pi_1 \cdot \prod_{\overline{R}} \cdot (id \times \pi_2) \\ Create = \prod_R \cdot (id \triangle \top) \end{array}$$

Recalling the discussion from Section 5.3.1, the invariant $\prod_R$ on a pair induces the individual invariant on the first element δR, from which the post-condition of the π_1 r-lens arises. The $Create$ operation $(id \triangle \top) = \pi_1^\circ$ yields the expected result, since it creates a pair by generating an arbitrary second element. The Put_{π_1} operation consists on the application of $id \times \pi_2$, the typical *put* operation in the functional setting, if the updated view is still related with the second element of the original source by R; otherwise, $Create$ is applied, generating an arbitrary valid pair.

Let's demonstrate the necessity of the $Create$ case by analyzing an example. Consider a pair whose invariant states that the first element must be greater than the second, i.e., $(A \times A)_{\prod_{\leq}}$. By projecting the first element, we obtain the $(A \times A)_{\prod_{\leq}} \trianglerighteq A_{id}$ lens. This means that any element of A is valid as a view. However, in the pair resulting from the backward transformation, the first element must be greater than the second. As such, there is the need to compare the new element with the original second element, and to generate a new arbitrary second value in case it fails.

The $\pi_2 : (A \times B)_{\prod_R} \trianglerighteq A_{\rho R}$ is similarly defined by:

$$\begin{array}{c} \pi_2 : (A \times B)_{\prod_R} \trianglerighteq A_{\rho R} \\ \\ Get = \pi_2 \\ Put = \prod_R \cdot swap \cdot (id \times \pi_1) \cup Create \cdot \pi_2 \cdot \prod_{\overline{R}} \cdot swap \cdot (id \times \pi_1) \\ Create = \prod_R \cdot (\top \triangle id) \end{array}$$

The reasoning about $\pi_2 : (A \times B)_{\prod_R} \trianglerighteq A_{\rho R}$ is similar to that of the r-lens π_1.

5.3.3 The $f \triangle g$ Combinator

For two lenses $f : A_\Phi \trianglerighteq B_\Omega$ and $g : A_\Phi \trianglerighteq C_\Lambda$, their split is defined by the r-lens

$$f \triangle g : A_\Phi \trianglerighteq (B \times C)_{\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega}} \tag{5.21}$$

defined by:

$$\frac{f : A_\Phi \trianglerighteq B_\Omega \qquad g : A_\Phi \trianglerighteq C_\Lambda}{f \triangle g : A_\Phi \trianglerighteq (B \times C)_{\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega}}}$$

$$\begin{aligned} Get &= Get_f \triangle Get_g \\ Put &= (Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f) \cup \\ &\quad Create \cdot \pi_1 \cdot \delta(Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f)_\neg \\ Create &= Create_f \cdot \pi_1 \cap Create_g \cdot \pi_2 \end{aligned}$$

Where $R \circledcirc S = R \cdot (\pi_1 \cdot \pi_1 \triangle S \cdot (\pi_2 \times id))$ and $R \circledcirc S = R \cdot (\pi_2 \cdot \pi_1 \triangle S \cdot (\pi_1 \times id))$. We will denote by $R \bullet S$ the expression $R \circledcirc S \cap S \circledcirc R$.

The invariant induced by the split invariant states that for any $(b, c) \in B \times C$, then $b \in \Omega$ and $c \in \Lambda$, and there exists an $a \in \Phi$ such that $Get_f(a) = b$ and $Get_g(b) = c$. This is expected, since the pairs are created by $Get_f \triangle Get_g$ from the invariant Φ. The Put operation has two cases: the first consists on the sequential application of Put_f and Put_g; the second case is applied if the first fails, when $Create$ is applied instead. The $Create$ operation yields the expected result, since for a pair (b, c), $Create_f \cdot \pi_1 \cap Create_g \cdot \pi_2$ produces an arbitrary value a such that $Get_f(a) = b$ and $Get_g(a) = c$. Note that the split non-determinism of the split $Create$ arises only from the non-determinism of f and g.

The $Create$ case is never applied if the first component is total (for the invariant). In particular, this happens if the result is independent of the order of application of the Put operations, since in this case $Put_f \circledcirc Put_g = Put_g \circledcirc Put_f$, and thus $Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f = Put_f \circledcirc Put_g = Put_g \circledcirc Put_f$ and the Put operations are total. Note that this is the condition defined in [PC10] for a split operation to be well-behaved.

The split combinator as defined accepts only r-lenses with the same pre-condition. In order to allow different invariants, the same mechanism that was used in the composition is applied, resorting to the s-restrict combinator. For two lenses $f : A_\Phi \trianglerighteq B_\Omega$ and $g : A_\Psi \trianglerighteq C_\Lambda$, their split is defined by:

$$f_\Psi \triangle g_\Phi : A_{\Phi \cap \Psi} \trianglerighteq (B \times C)_{\prod_{\Lambda \cdot \cdot g \cdot (\Phi \cap \Psi) \cdot f^\circ \cdot \cdot \Omega}} \tag{5.22}$$

With the split combinator, the duplication operation $id \triangle id$ can now be defined, resulting in the following r-lens:

$$\begin{aligned} &id \triangle id : A_\Phi \trianglerighteq (A \times A)_{\prod_\Phi} \\ &\\ &Get = id \triangle id \\ &Put = (\pi_1 \cap \pi_2) \cdot \pi_1 \\ &Create = \Phi \cdot (\pi_1 \cap \pi_2) \end{aligned}$$

Note that the consistency relation between the elements of the resulting pair is a coreflexive, meaning that they must always be equal. This way the backward transformation is able to preserve the correctness of the system. The Put operation states that the new source must be equal to the first and second elements of the view, as expected. Since this is total for the invariant $\prod_\Phi$, the $Create$ case was dropped.

Another important result to validate the split combinator is to check that the reflection law $\pi_1 \triangle \pi_2 = id$ holds, an important property of the categorical product. Using the defined combinators, the following r-lens is obtained:

$$\pi_1 \triangle \pi_2 : (A \times B)_{\prod_R} \trianglerighteq (A \times B)_{\prod_R}$$

$$Get = id$$
$$Put = \pi_1$$
$$Create = \prod_R$$

The behavior of this lens is as expected: the forward transformation is the identity function, while Put simply retrieves the new pair. Once again, the $Create$ case drops because $Put_{\pi_1} \bullet Put_{\pi_2}$ is functional.

5.3.4 The $f \times g$ Combinator

We now have all the combinators needed to define the product of lenses as $f \times g = f \cdot \pi_1 \triangle g \cdot \pi_2$. Given two lenses $f : A_\Phi \trianglerighteq C_\Omega$ and $g : B_\Psi \trianglerighteq D_\Lambda$, the expressions inside the split have types $f \cdot \pi_1 : (A \times B)_{\Phi \times \Psi} \trianglerighteq C_\Omega$ and $g \cdot \pi_2 : (A \times B)_{\Phi \times \Psi} \trianglerighteq D_\Lambda$, and thus, the product of r-lenses will have the type $f \cdot \pi_1 \triangle g \cdot \pi_2 : (A \times B)_{\Phi \times \Psi} \trianglerighteq (C \times D)_{\prod_{\Lambda \cdot g \cdot \pi_2 \cdot (\Phi \times \Psi) \cdot \pi_1^\circ \cdot f^\circ \cdot \Omega}}$. With some simplifications, we obtain the following r-lens:

$$\frac{f : A_\Phi \trianglerighteq C_\Omega \qquad g : B_\Psi \trianglerighteq D_\Lambda}{f \times g : (A \times B)_{\Phi \times \Psi} \trianglerighteq (C \times D)_{\rho(\Omega \cdot f \cdot \Phi) \times \rho(\Lambda \cdot g \cdot \Psi)}}$$

$$Get = f \times g$$
$$Put = (Put_f \times Put_g) \cdot (\pi_1 \times \pi_1 \triangle \pi_2 \times \pi_2)$$
$$Create = Create_f \times Create_g$$

The Get operation is directly defined by $f \cdot \pi_1 \triangle g \cdot \pi_2$. As for the Put operation, the expected definition is obtained, since it is exactly the functional product combinator defined in [PC10].

A more generic definition assumes that there exists a relation R on the source pair. In such case, for two lenses $f : A_{\delta R} \trianglerighteq C_\Omega$ and $g : B_{\rho R} \trianglerighteq D_\Lambda$, we obtain the r-lens:

$$f \times g : (A \times B)_{\prod_R} \trianglerighteq (C \times D)_{\prod_{\Psi \cdot g \cdot R \cdot f^\circ \cdot \Phi}}$$

Although the definition of Get remains the same, the Put operation becomes more complex. The Puts of the expressions inside the split are:

$$Put_{f \cdot \pi_1} = (\prod_R \cup \prod_R \cdot \ker \pi_1 \cdot \prod_{\overline{R}}) \cdot (Put_f \cdot (id \times \pi_1) \triangle \pi_2 \cdot \pi_2)$$
$$Put_{g \cdot \pi_2} = (\prod_R \cup \prod_R \cdot \ker \pi_2 \cdot \prod_{\overline{R}}) \cdot swap \cdot (Put_f \cdot (id \times \pi_2) \triangle \pi_1 \cdot \pi_2)$$

As such, the sequential application of the $Put_{f \cdot \pi_1} \bullet Put_{g \cdot \pi_2}$ takes the form:

$$Put_{f \cdot \pi_1} \cdot (\pi_1 \cdot \pi_1 \triangle \cdot Put_{g \cdot \pi_2} \cdot (\pi_2 \times id))$$
$$\cap$$
$$Put_{g \cdot \pi_2} \cdot (\pi_2 \cdot \pi_1 \triangle \cdot Put_{f \cdot \pi_1} \cdot (\pi_1 \times id))$$

Finally, by adding the $Create$ case, the Put is defined as:

$$Put_{f \cdot \pi_1} \bullet Put_{g \cdot \pi_2} \cup \prod_R \cdot (f^\circ \times g^\circ) \cdot \pi_1 \cdot \delta(Put_{f \cdot \pi_1} \bullet Put_{g \cdot \pi_2})_\neg$$

5.4 Remaining Combinators

In order to have a complete framework, there is the need to define some other combinators, including the constant lenses and the ones related with the coproduct. Although yet to be proven, these r-lenses are considerably simpler than the previously presented, and, as such, it is possible to infer their shape and invariants.

5.4.1 Coproduct of R-Lenses

Regarding the invariant on coproducts, we follow the same rationale as the one applied to the products. While in the products, the invariant was defined by the range of the split operator, since pairs are created by splits, the invariant on coproducts will be defined as the range of the either operator, since coproducts are consumed by eithers. The domain of an either $R \nabla S$ is defined as:

$$\delta(R \cdot i_1^\circ) \cup \delta(S \cdot i_2^\circ) \cup (i_1 \cdot R^\circ \cdot S \cdot i_2^\circ) \cup (i_2 \cdot S^\circ \cdot R \cdot i_1^\circ)$$

However, this definition would assume, like in the product invariant, the existence of a consistency relation between the elements of A and B, which has no purpose when dealing with coproducts since only one of the elements may exist at a time. Thus, that component of the expression is dropped, and the invariant is defined as $\delta(\Phi \cdot i_1^\circ) \cup \delta(\Psi \cdot i_2^\circ) = \Phi + \Psi$.

The first primitives of the coproducts are the injections. The r-lens $i_1 : A_\Phi \trianglerighteq (A + B)_{\delta(\Phi \cdot i_1^\circ)}$ is defined as:

$$\boxed{\begin{array}{c} i_1 : A_\Phi \trianglerighteq (A + B)_{\Phi + id} \\ \\ Get = i_1 \\ Put = Create^\circ \cdot \pi_1 \\ Create = \Phi \cdot i_1^\circ \end{array}}$$

When a value is injected on the left side of the coproduct, naturally there is no invariant on the right side, i.e., it is the identity relation. The injection i_2 can be similarly defined.

The either combinator $f \nabla g : (A + B)_{\Phi + \Psi} \trianglerighteq C_{\Omega \cup \Lambda}$ takes two lenses and applies them on either side of the product:

$$\boxed{\begin{array}{c} \dfrac{f : A_\Phi \trianglerighteq C_\Omega \qquad g : B_\Psi \trianglerighteq C_\Lambda}{f \nabla g : (A + B)_{\Phi + \Psi} \trianglerighteq C_{\Omega \cup \Lambda}} \\ \\ Get = f \nabla g \\ Put = ((i_1 \cdot Put_f \cdot (\Omega \times id) \cup i_2 \cdot Create_g \cdot \Omega_\neg \cdot \pi_1) \nabla \\ \qquad (i_1 \cdot Create_f \cdot \Lambda_\neg \cdot \pi_1 \cup i_2 \cdot Put_g \cdot (\Lambda \times id))) \cdot distr \\ Create = (i_1 \cdot Create_f) \cup (i_2 \cdot Create_g) \end{array}}$$

The invariant on the views is $\Omega \cup \Lambda$ since the value may have originated from f or g. Regarding the Put operation, its input type is $C \times (A + B)$. After distributing it with the isomorphism $distr : A \times (B + C) \to (A \times B) + (A \times C)$, either the Put_f or the Put_g is applied, depending on which side the coproduct is injected, if the value is within the respective invariant; otherwise, the respective $Create$ is applied to generate an arbitrary valid source.

Finally, the coproduct combinator $f + g : (A + B)_{\Phi+\Psi} \trianglerighteq (C + D)_{\Omega+\Lambda}$, that enables parallel transformation of coproduct:

$$\frac{f : A_\Phi \trianglerighteq C_\Omega \qquad g : B_\Psi \trianglerighteq D_\Lambda \qquad R : A_\Phi \to B_\Psi}{f + g : (A + B)_{\Phi+\Psi} \trianglerighteq (C + D)_{\Omega+\Lambda}}$$

$$\begin{gathered} Get = f + g \\ Put = ((Put_f \nabla Put_f \cdot (id \times R^\circ)) + (Put_g \cdot (id \times R) \nabla Put_g)) \cdot dists \\ Create = Create_f + Create_g \end{gathered}$$

Where $dists = distr \cdot (distl + distl)$, and $distl$ is the isomorphism $(A+B) \times C \to (A \times C) + (B \times C)$. This is the expected behavior of the coproduct of r-lenses, with a definition directly lifted from the functional setting [PC10]. However, it can also be defined by combining previously defined primitives as $f + g = i_1 \cdot f \nabla i_2 \cdot g$. It remains to analyze the behavior of such constructor, and its comparison with the expected one.

5.4.2 Conditional Combinator

The conditional combinator $\Phi? : A_\Psi \trianglerighteq (A + A)_{(\Phi \cap \Psi)+(\Phi_\neg \cap \Psi)}$, where Φ is the condition, produces a coproduct where the original input is passed to either right-side, if it belongs to Φ, or to the left-side otherwise:

$$\frac{\Phi : A \to A}{\Phi? : A_\Psi \trianglerighteq (A + A)_{(\Phi \cap \Psi)+(\Phi_\neg \cap \Psi)}}$$

$$\begin{gathered} Get = i_1 \cdot \Phi \cup i_2 \cdot \Phi_\neg \\ Put = Create \cdot \pi_1 \\ Create = \Psi \cdot (\Phi \nabla \Phi_\neg) \end{gathered}$$

The *Get* operation tests the value with the coreflexives, injecting them on the respective side. *Create* checks the condition on either side of the view, returning the value if it is valid on the correct side of the coproduct. The *Put* operation performs the same transformation, after discarding the original source. On the views, the original invariant on A is further restricted by a coreflexive defining wether the value must pass the condition or not.

5.4.3 Constant R-Lenses

The bang $! : C_\Phi \trianglerighteq 1$ r-lens and the constant r-lens $\underline{a} : C_\Phi \trianglerighteq A_{\underline{a} \cap id}$ are defined in similar way:

$! : C_\Phi \trianglerighteq 1$	$\underline{a} : C_\Phi \trianglerighteq A_{\rho \underline{a}}$
$Get = !$	$Get_{\underline{a}} = \underline{a}$
$Put = \pi_2$	$Put = \pi_2$
$Create = \Phi \cdot \top$	$Create = \Phi \cdot \top$

The *Get* operation applied the constant transformation, while the *Put* simply retrieves the original source. *Create* generates an arbitrary value within the invariant. Note that, as expected, the invariants are lost with the forward transformation.

Chapter 6

Future Work and Discussion

In this Chapter the planned future work on r-lenses and its expected applications are presented. Lastly, we analyze the results of the relational approach so far, as well as the viability of the remaining goals of the thesis.

6.1 Future Work on R-Lenses

Once the combinators are fully defined, the r-lens framework must be applied to concrete problems. However, before the implementation, r-lenses must still undergo a "functionalization" process, since at the moment backward transformations are possibly non-deterministic. Non-determinism arises solely by the application of the *Create* operation, and consequently, from the *Put* operations that resort to it. In particular, at the moment only the *Put* operations of the π and the constant r-lenses may introduce non-determinism. Many times, the operations will probably naturally degenerate in functions, like the $\pi_1 \triangle \pi_2$ r-lens presented. However, in the other cases, the need to somehow choose a single value for the backward transformation will emerge. Such is the case of the π r-lenses when the new view is not related to the original other element, since the new generated value is completely arbitrary.

There are various solutions to this problem. For instance, in [KH06] there is no concern about which transformation is chosen, as long as the system remains consistent. Another possibility is to let the user decide which transformation he deems best when non-determinism occurs. This is the solution applied, for instance, in [PC10], where the π lenses take as input the concrete value to be returned when *Create* is applied. A more generic solution, instead of a concrete value, takes an order on transformations that identifies "better" transformations, which will be used to choose the correct one.

This kind of optimization can be performed by the *shrink* operator introduced and analyzed in [MO11], which possesses very interesting properties. Given a relation $R : A \to B$ and a criterion $C : B \to B$, the shrink operator is defined by the universal law:

$$X \subseteq R \upharpoonright C \equiv X \subseteq R \wedge X \cdot R^{\circ} \subseteq C$$

Meaning that X is at most R, and that for any output x of X, it is a maximum in R in regard to the criterion defined by C. The operator can also be directly defined as:

$$R \upharpoonright C = R \cap C/R^{\circ}$$

As such, in our particular case, provided we defined a criterion C for preferred sources, we could define the function *create* as:

$$create \subseteq Create \upharpoonright C$$

With the r-lens framework we expect to define a complete PF language for bidirectional transformation. An essential step towards that goal is the definition of recursive patterns (folds and unfolds) on r-lenses and support for inductive types, following the already studied for functional lenses [PC10], further improving the expressivity of the language.

Following that, future work on r-lenses involves the lifting of generic functional languages to a bidirectional setting, as well as providing formal bases for existing transformation languages, like XPath [CD99]. Moreover, r-lenses can be used to redesign existing frameworks, like the lenses for databases from [BPV06] by using the invariants to represent functional dependencies. The framework also allows a more precise way to analyze specifications, and leads to further and more natural optimizations, following the style of [PC11]. In fact, the r-lenses can be used as the underlying setting for most of the goals of the thesis: since graphs can be defined using invariants and inductive types, the r-lens framework can be used to encode graph transformations.

6.2 Conclusions

A taxonomy defined in terms of RAs was presented, as well as the state-of-the-art review, classified under that same taxonomy; an existing framework with particular characteristics was rewrote under the relational setting, further validating RAs as the unifying framework; by studying possible techniques for the correct derivation by calculation of the backward transformation we propose mechanisms for the optimization of existing approaches; by developing a framework where lenses are generalized to a relational settings and the types are restricted by invariants, it has been shown that the relational setting can be used to improve existing bidirectional techniques with new features.

The positive results motivate us to pursue the thesis objectives. RAs have proven not only to be able to provide a unifying perspective upon existing approaches, but also to enable their enhancement. Moreover, its simplicity and ease of calculation provide simple analysis and proof of properties. The r-lens framework in particular, already at an advanced stage of development, is expected to result in a robust implementation with interesting applications. In particular, we believe that it can be used to define a novel framework for bidirectional graph transformations, simpler and easier to manipulate than existing approaches.

As such, we believe that the overall objectives of the thesis initially defined are achievable with the relational approach, and in particular the r-lens framework as the underlying formalism. The proposed methodology is to finnish the development of the r-lenses framework, including the introduction of recursion,

and subsequently the enhancement of the state invariants in order to support inductive types, allowing the representation of more complex data structures, (and in particular graphs). The next step is to use the r-lenses to define the semantics of high-level model transformations languages (like QVT). Finally, the framework will be implemented, and case studies performed to validate the formal foundations of the system, including the encoding of existing transformation languages, or the lifting of general purpose functional languages to a bidirectional setting.

Bibliography

[Bac78] John Backus. Can programming be liberated from the von Neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[Bac04] Roland C. Backhouse. Mathematics of program construction. Draft of book in preparation, 2004.

[BCF+05] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language. W3C Working Draft, 2005.

[BCF+10] Davi M. J. Barbosa, Julien Cretin, J. Nate Foster, Michael Greenberg, and Benjamin C. Pierce. Matching lenses: alignment and view update. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010*, pages 193–204. ACM, 2010.

[BCPV07] Pablo Berdaguer, Alcino Cunha, Hugo Pacheco, and Joost Visser. Coupled schema transformation and data conversion for XML and SQL. In *PADL*, volume 4354 of *LNCS*, pages 290.–304. Springer, 2007.

[BdM96] Richard Bird and Oege de Moor. *Algebra of Programming*, volume 100 of *International Series in Computer Science*. Prentice-Hall, Inc., 1996.

[BFP+08] Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: resourceful lenses for string data. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008*, pages 407–419. ACM, 2008.

[BFS00] Peter Buneman, Mary F. Fernández, and Dan Suciu. UnQL: A query language and algebra for semistructured data based on structural recursion. *VLDB J.*, 9(1):76–110, 2000.

[BMS08] Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. Dual syntax for XML languages. *Inf. Syst.*, 33(4–5):385–406, 2008.

[BPV06] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: a language for updatable views. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium*

on Principles of Database Systems, PODS 2006, pages 338–347. ACM, 2006.

[BS81] François Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981.

[CD99] James Clark and Steve DeRose. *XML path language (XPath)*, 1999.

[CFH+09] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Theory and Practice of Model Transformations, Second International Conference, ICMT 2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.

[CPP06] Alcino Cunha, Jorge S. Pinto, and José Proença. A framework for point-free program transformation. In *Revised Papers of the 17th International Workshop on Implementation and Application of Functional Languages (IFL'05)*, volume 4015 of *LNCS*, pages 1–18. Springer, 2006.

[DB82] Umeshwar Dayal and Philip A. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. Database Syst.*, 7(3):381–416, 1982.

[Dis08] Zinovy Diskin. Algebraic models for bidirectional model synchronization. In *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008*, volume 5301 of *LNCS*, pages 21–36. Springer, 2008.

[DXC10] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. From state- to delta-based bidirectional model transformations. In *Theory and Practice of Model Transformations, Third International Conference, ICMT 2010*, volume 6142 of *LNCS*, pages 61–76. Springer, 2010.

[EEE+07] Hartmut Ehrig, Karsten Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information preserving bidirectional model transformations. *Fundamental Approaches to Software Engineering*, pages 72–86, 2007.

[EG07] Robert Ennals and David Gay. Multi-language synchronization. In *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007*, volume 4421 of *LNCS*, pages 475–489. Springer, 2007.

[EM45] Samuel Eilenberg and Saunders MacLane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2):231–294, 1945.

[FGM+07] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3), 2007.

[Fos09] J. Nathan Foster. *Bidirectional programming languages.* PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2009.

[FPP08] J. Nathan Foster, Alexandre Pilkiewicz, and Benjamin C. Pierce. Quotient lenses. In *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP 2008*, pages 383–396. ACM, 2008.

[FPZ09] J. Nathan Foster, Benjamin C. Pierce, and Steve Zdancewic. Updatable security views. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009*, pages 60–74. IEEE Computer Society, 2009.

[Fv90] Peter J. Freyd and Andre Ščedrov. *Categories, Allegories*, volume 39 of *North Holland Mathematical Library.* North Holland, 1990.

[GPZ88] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and update semantics of consistent views. *ACM Trans. Database Syst.*, 13(4):486–524, 1988.

[HHI+10] Soichiro Hidaka, Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato, Kazutaka Matsuda, and Keisuke Nakano. Bidirectionalizing graph transformations. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010*, pages 205–216. ACM, 2010.

[HHKN09] Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. Towards a compositional approach to model transformation for software development. In *Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009*, pages 468–475. ACM, 2009.

[HMT08] Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. *Higher-Order and Symbolic Computation*, 21(1–2):89–118, 2008.

[HPW11] Martin Hofmann, Benjamin C. Pierce, and Daniel Wagner. Symmetric lenses. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 371–384. ACM, 2011.

[Ken04] Andrew Kennedy. Pickler combinators. *J. Funct. Program.*, 14(6):727–739, 2004.

[KH06] Shinya Kawanaka and Haruo Hosoya. biXid: a bidirectional transformation language for XML. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006*, pages 201–214. ACM, 2006.

[KS06] Alexander Königs and Andy Schürr. Tool integration with triple graph grammars - a survey. *Electr. Notes Theor. Comput. Sci.*, 148(1):113–150, 2006.

[LÏ5] Leopold Löwenheim. Uber möglichkeiten im relativkalkul. *Mathematische Annalen*, 76:447–470, 1915.

[LHT07] Dongxi Liu, Zhenjiang Hu, and Masato Takeichi. Bidirectional interpretation of XQuery. In *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2007*, pages 21–30. ACM, 2007.

[Lut08] David Lutterkort. Augeas – a configuration API. In *Linux Symposium, Ottawa, ON*, pages 47–56, 2008.

[Lyn50] Roger Lyndon. The representation of relational algebras. *Annals of Mathematics*, 51:707–729, 1950.

[MAB08] Sergey Melnik, Atul Adya, and Philip A. Bernstein. Compiling mappings to bridge applications and databases. *ACM Trans. Database Syst.*, 33(4), 2008.

[Mad91] Roger D. Maddux. The origin of relation algebras in the development and axiomatization of the calculus of relations. *Studia Logica*, 50(3):421–455, 1991.

[Mad06] Roger D. Maddux. *Relation Algebras*, volume 150 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006.

[Mee98] Lambert Meertens. Designing constraint maintainers for user interaction. Unpublished manuscript, 1998.

[MHN+07] Kazutaka Matsuda, Zhenjiang Hu, Keisuke Nakano, Makoto Hamana, and Masato Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007*, pages 47–58. ACM, 2007.

[MHT04] Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. An algebraic approach to bi-directional updating. In *Programming Languages and Systems: Second Asian Symposium, APLAS 2004*, volume 3302 of *LNCS*, pages 2–20. Springer, 2004.

[MO11] Shin-Cheng Mu and José N. Oliveira. Programming from galois connections. In *Relational and Algebraic Methods in Computer Science - 12th International Conference, RAMICS 2011*, volume 6663 of *LNCS*, pages 294–313. Springer, 2011.

[Mor60] Augustus De Morgan. On the syllogism, no. IV. and on the logic of relations. *Transactions of the Cambridge Philosophical Society*, 10:331 – 358, 1860.

[Obj08] Object Management Group (OMG). *Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification*, 2008.

[OCV06] José N. Oliveira, Alcino Cunha, and Joost Visser. Type-safe two-level data transformation. In *14th International Symposium on Formal Methods*, volume 4085 of *LNCS*, pages 284–299. Springer, 2006.

[Oli05] José N. Oliveira. First steps in pointfree functional dependency theory. Unpublished manuscript, 2005.

[Oli07] José N. Oliveira. Data transformation by calculation. GTTSE'07 Lecture, Universidade do Minho, 2007.

[Oli08] José N. Oliveira. Transforming data by calculation. In *Generative and Transformational Techniques in Software Engineering II*, volume 5235 of *LNCS*, pages 134–195. Springer, 2008.

[Oli09] José N. Oliveira. Extended static checking by calculation using the point-free transform. In *LerNet ALFA Summer School 2008*, volume 5520 of *LNCS*, pages 195–251. Springer, 2009.

[OR06] José N. Oliveira and César Rodrigues. Point-free factorization of operation refinement. In *14th International Symposium on Formal Methods*, volume 4085 of *LNCS*, pages 236–251. Springer-Verlag, 2006.

[Ore44] Oystein Ore. Galois connexions. *Transactions of the American Mathematical Society*, 55:493–513, 1944.

[PC10] Hugo Pacheco and Alcino Cunha. Generic point-free lenses. In *Mathematics of Program Construction, 10th International Conference, MPC 2010*, volume 6120 of *LNCS*, pages 331–352. Springer, 2010.

[PC11] Hugo Pacheco and Alcino Cunha. Calculating with lenses: optimising bidirectional transformations. In *Proceedings of the 2011 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2011*, pages 91–100. ACM, 2011.

[Pei33] Charles S. Peirce. Note B. the logic of relatives. In *Studies in logic by members of the Johns Hopkins University*, pages 187 – 203. Little, Brown and Company, Boston, 1833.

[Sch95] Ernst Schröder. *Vorlesungen uber die Algebra der Logik (exakte Logik)*, volume 3, part 1: Algebra und Logik der Relative. Leipzig, 1895.

[Sch94] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94*, volume 903 of *LNCS*, pages 151–163. Springer, 1994.

[SK08] Andy Schürr and Felix Klar. 15 years of triple graph grammars. In *Graph Transformations, 4th International Conference, ICGT 2008*, volume 5214 of *LNCS*, pages 411–425. Springer, 2008.

[Ste08] Perdita Stevens. Towards an algebraic theory of bidirectional transformations. In *Graph Transformations, 4th International Conference, ICGT 2008*, volume 5214 of *LNCS*, pages 1–17. Springer, 2008.

[Ste10] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and System Modeling*, 9(1):7–20, 2010.

[Tar41] Alfred Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73 – 89, 1941.

[TG87] Alfred Tarski and Steven Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, 1987.

[VHMW10] Janis Voigtländer, Zhenjiang Hu, Kazutaka Matsuda, and Meng Wang. Combining syntactic and semantic bidirectionalization. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010*, pages 181–192. ACM, 2010.

[Voi09] Janis Voigtländer. Bidirectionalization for free! (pearl). In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 165–176. ACM, 2009.

[Wad87] Philip Wadler. Views: A way for pattern matching to cohabit with data abstraction. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, POPL 1987*, pages 307–313. ACM, 1987.

[WGMH11] Meng Wang, Jeremy Gibbons, Kazutaka Matsuda, and Zhenjiang Hu. Refactoring pattern matching. *Science of Computer Programming*, 2011.

Appendix A

Proofs

$$\boxed{\begin{array}{c} id : A_\Phi \trianglerighteq A_\Phi \\ \\ Get = id \\ Put = \pi_1 \\ Create = id \end{array}}$$

Proof.

Get **total** $\Phi \subseteq \delta id$

Trivial by (Φ - DEF).

Put **total** $\Phi \times \Phi \subseteq \delta \pi_1$

Trivial by (π - DOMAIN) and (Φ - DEF).

Acceptable $\pi_1 \cdot (\Phi \times \Phi) \subseteq \pi_1$

Trivial by (Φ - SMALLER).

Stable $\pi_1 \cdot (id \triangle id) \cdot \Phi \subseteq id$

$$\begin{array}{ll} & \pi_1 \cdot (id \triangle id) \cdot \Phi \subseteq id \\ \Leftrightarrow & \{\triangle \text{ - CANCELLATION}\} \\ & \Phi \subseteq id \\ \Leftrightarrow & \{\Phi \text{ - DEF}\} \\ & true \end{array}$$

Get **preserves invariant** $\Phi \subseteq \Phi$

Trivial.

Put **preserves invariant** $\pi_1 \cdot (\Phi \times \Phi) \subseteq \Phi \cdot \pi_1$

$$\begin{array}{ll} & \pi_1 \cdot (\Phi \times \Phi) \subseteq \Phi \cdot \pi_1 \\ \Leftrightarrow & \{\triangle \text{ - CANCELLATION}\} \\ & \Phi \cdot \pi_1 \cdot \delta(\Phi \cdot \pi_2) \subseteq \Phi \cdot \pi_1 \\ \Leftrightarrow & \{\Phi \text{ - SMALLER}\} \\ & true \end{array}$$

□

$$\frac{f : B_\Phi \trianglerighteq A_\Psi \qquad g : C_\Omega \trianglerighteq B_\Phi}{f \cdot g : C_\Omega \trianglerighteq A_\Psi}$$

$$\begin{array}{c} Get = Get_f \cdot Get_g \\ Put = Put_g \cdot (Put_f \cdot (id \times Get_g) \triangle \pi_2) \\ Create = Create_g \cdot Create_f \end{array}$$

Proof. *Get* **total** $\Omega \subseteq \delta(Get_f \cdot Get_g)$

$$\begin{array}{ll} & \{f, g \text{ lenses}\} \\ & \Omega \subseteq g^\circ \cdot \Phi \cdot g \wedge \Phi \subseteq f^\circ \cdot \Psi \cdot f \\ \Rightarrow & \{\subseteq \text{ - TRANSITIVITY}, \Phi \text{ - LARGER}\} \\ & \Omega \subseteq g^\circ \cdot f^\circ \cdot f \cdot g \\ \Rightarrow & \{\cap \text{ - MONOTONICITY}, \delta \text{ - DEF}\} \\ & \Omega \subseteq \delta(f \cdot g) \end{array}$$

Put **total** $(\Psi \times \Omega) \subseteq \delta(Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2))$

Starting by simplifying the right-hand side of the inequation:

$$\begin{array}{ll} & \delta(Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2)) \\ \supseteq & \{\delta \text{ - COMPOSITION}, g \text{ lens}\} \\ & \delta((\Phi \times \Omega) \cdot (Put_f \cdot (id \times g) \triangle \pi_2)) \\ \supseteq & \{\triangle \text{ - ABSORPTION}, f \text{ lens}\} \\ & \delta(Put_f \cdot (\Psi \times \Phi) \cdot (id \times g) \triangle \Omega \cdot \pi_2) \\ \supseteq & \{\delta \text{ - COMPOSITION}, f \text{ lens}\} \\ & \delta((\Psi \times \Phi) \cdot (id \times g) \triangle \Omega \cdot \pi_2) \\ \supseteq & \{\triangle \text{ - ABSORPTION}, g \text{ lens}\} \\ & \delta((id \times g) \cdot (\Psi \times \Omega) \triangle \Omega \cdot \pi_2) \\ = & \{\triangle \text{ - DOMAIN}\} \\ & \delta((id \times g) \cdot (\Psi \times \Omega)) \cap \delta(\Omega \cdot \pi_2) \end{array}$$

As such, by ($\subseteq$ - TRANSITIVITY) and ($\cap$ - UNIVERSAL), it suffices to prove:

$$\begin{array}{ll} & \Psi \times \Omega \subseteq \delta((id \times g) \cdot (\Psi \times \Omega)) \wedge \Psi \times \Omega \subseteq \delta(\Omega \cdot \pi_2) \\ \Leftrightarrow & \{\Phi \text{ - IN DOMAIN/RANGE}, \text{ker - DEF}, \text{SHUNTING}\} \\ & (id \times g) \cdot (\Psi \times \Omega) \subseteq (id \times g) \cdot (\Psi \times \Phi) \wedge \pi_2 \cdot (\Psi \times \Omega) \subseteq \Omega \cdot \pi_2 \\ \Leftrightarrow & \{\triangle \text{ - CANCELLATION}, \Phi \text{ - SMALLER}\} \\ & true \end{array}$$

Acceptable and Stable Trivial from Lemma 5.1.

Get **preserves invariant** $Get_f \cdot Get_g \cdot \Omega \subseteq \Psi \cdot Get_f \cdot Get_g$

$$
\begin{array}{ll}
& \{f, g \text{ lenses}\} \\
& g \cdot \Omega \cdot g^\circ \subseteq \Phi \wedge \Phi \subseteq f^\circ \cdot \Psi \cdot f \\
\Rightarrow & \{\subseteq \text{ - TRANSITIVITY}\} \\
& g \cdot \Omega \cdot g^\circ \subseteq f^\circ \cdot \Psi \cdot f \\
\Leftrightarrow & \{\text{SHUNTING}\} \\
& f \cdot g \cdot \Omega \subseteq \Psi \cdot f \cdot g
\end{array}
$$

Put **preserves invariant**

$$Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2) \cdot (\Psi \times \Omega) \subseteq \Omega \cdot Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2)$$

$$
\begin{array}{ll}
& Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2) \cdot (\Psi \times \Omega) \\
= & \{\triangle \text{ - FUSION}, \triangle \text{ - ABSORPTION}\} \\
& Put_g \cdot (Put_f \cdot (\Psi \times g \cdot \Omega) \triangle \Omega \cdot \pi_2 \cdot \delta(\Psi \cdot \pi_2)) \\
\subseteq & \{\Phi \text{ - LARGER}, g \text{ lens}\} \\
& Put_g \cdot (Put_f \cdot (\Psi \times \Phi \cdot g) \triangle \Omega \cdot \pi_2) \\
\subseteq & \{f, g \text{ lenses}\} \\
& \Omega \cdot Put_g \cdot (Put_f \cdot (id \times g) \triangle \pi_2)
\end{array}
$$

□

$$
\boxed{
\begin{array}{c}
\dfrac{f : A_\Phi \trianglerighteq B_\Psi \qquad \Theta : B \to B}{f^\Theta : A_{\Phi \cap \delta(\Theta \cdot f)} \trianglerighteq B_{\Psi \cap \Theta}} \\
\\
Get = \Theta \cdot Get_f \\
Put = Put_f \\
Create = Create_f
\end{array}
}
$$

Proof.

Get **total** $\Phi \cap \delta(\Theta \cdot f) \subseteq \delta(\Theta \cdot f)$

Trivial from ($\cap$ - SMALLER).

Put **total** $(\Psi \cap \Theta) \times (\Phi \cap \delta(\Theta \cdot f)) \subseteq \delta Put_f$

$$
\begin{array}{ll}
& \{f \text{ lens}\} \\
& \Psi \times \Phi \subseteq \delta Put_f \\
\Rightarrow & \{\Phi \text{ - SMALLER}\} \\
& (\Psi \cap \Theta) \times (\Phi \cap \delta(\Theta \cdot f)) \subseteq \delta Put_f
\end{array}
$$

Acceptable $\Theta \cdot f \cdot Put_f \cdot (\Psi \times \Phi) \cdot (\Theta \times \delta(\Theta \cdot f)) \subseteq (\Psi \cap \Theta) \cdot \pi_1$

$$
\begin{array}{ll}
& \{f \text{ lens}\} \\
& f \cdot Put_f \cdot (\Psi \times \Phi) \subseteq \Psi \cdot \pi_1 \\
\Rightarrow & \{\Phi \text{ - SMALLER}, \cdot \text{ - MONOTONICITY}, \Phi \text{ - COMPOSITION}\} \\
& \Theta \cdot f \cdot Put_f \cdot (\Psi \times \Phi) \cdot (\Theta \times \delta(\Theta \cdot f)) \subseteq (\Psi \cap \Theta) \cdot \pi_1
\end{array}
$$

Stable $Put_f \cdot ((\Theta \cdot f) \triangle id) \cdot (\Phi \cap \delta(\Theta \cdot f)) \subseteq id$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad Put_f \cdot (f \triangle id) \cdot \Phi \subseteq id \\
\Rightarrow \{\Phi \text{ - SMALLER}\} \\
\quad Put_f \cdot ((\Theta \cdot f) \triangle id) \cdot (\Phi \cap \delta(\Theta \cdot f)) \subseteq id
\end{array}
$$

Get **preserves invariant** $\Theta \cdot f \cdot (\Phi \cap \delta(\Theta \cdot f)) \subseteq (\Psi \cap \Theta) \cdot \Theta \cdot f$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad f \cdot \Phi \subseteq \Psi \cdot f \\
\Rightarrow \{\Phi \text{ - LARGER}, \cdot \text{ - MONOTONICITY}\} \\
\quad \Theta \cdot f \cdot \Phi \cdot \delta(\Theta \cdot f) \subseteq \Theta \cdot \Psi \cdot f \\
\Leftrightarrow \{\Phi \text{ - COMPOSITION}, \Phi \text{ - REFLEXIVITY}\} \\
\quad \Theta \cdot f \cdot (\Phi \cap \delta(\Theta \cdot f)) \subseteq (\Psi \cap \Theta) \cdot \Theta \cdot f
\end{array}
$$

Put **preserves invariant**

$$Put_f \cdot ((\Psi \cap \Theta) \times (\Phi \cap \delta(\Theta \cdot f))) \subseteq (\Phi \cap \delta(\Theta \cdot f)) \cdot Put_f$$

$$
\begin{array}{l}
\quad \{\text{Lemma 5.2}\} \\
\quad Put_f \cdot ((\Psi \cap \Theta) \times \Phi) \subseteq \delta(\Theta \cdot f) \cdot Put_f \cdot ((\Psi \cap \Theta) \times \Phi) \\
\Rightarrow \{\Phi \text{ - SMALLER}, f \text{ lens}\} \\
\quad Put_f \cdot ((\Psi \cap \Theta) \times (\Phi \cap \delta(\Theta \cdot f))) \subseteq \delta(\Theta \cdot f) \cdot \Phi \cdot Put_f \cdot (\Theta \times id) \\
\Leftrightarrow \{\Phi \text{ - COMPOSITION}, \Phi \text{ - LARGER}\} \\
\quad Put_f \cdot ((\Psi \cap \Theta) \times (\Phi \cap \delta(\Theta \cdot f))) \subseteq (\Phi \cap \delta(\Theta \cdot f)) \cdot Put_f
\end{array}
$$

□

$$
\frac{f : A_\Phi \trianglerighteq B_\Psi \qquad \Omega \supseteq \rho(Put_f)}{f_\Omega : A_{\Phi \cap \Omega} \trianglerighteq B_{\Psi \cap \rho(f \cdot \Omega)}}
$$

$$
\begin{array}{c}
Get = Get_f \\
Put = \Omega \cdot Put_f \\
Create = \Omega \cdot Create_f
\end{array}
$$

Proof.

Get **total** $\Phi \cap \Omega \subseteq \delta f$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad \Phi \subseteq \delta f \\
\Rightarrow \{\Phi \text{ - SMALLER}\} \\
\quad \Phi \cap \Omega \subseteq \delta f
\end{array}
$$

Put **total** $(\Psi \cap \rho(f \cdot \Omega)) \times (\Phi \cap \Omega) \subseteq \delta(\Omega \cdot Put_f)$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad \Psi \times \Phi \subseteq \delta Put_f \\
\Rightarrow \{\cdot \text{ - MONOTONICITY}, \Phi \text{ - COMPOSITION}. \triangle \text{ - ABSORPTION}\} \\
\quad (\Psi \cap \rho(f \cdot \Omega)) \times (\Phi \cap \Omega) \subseteq \delta(Put_f \cdot (\rho(f \cdot \Omega) \times \Omega)) \\
\Rightarrow \{\rho(Put_f) \subseteq \Omega, \subseteq \text{ - TRANSITIVITY}, \Phi \text{ - LARGER}\} \\
\quad (\Psi \cap \rho(f \cdot \Omega)) \times (\Phi \cap \Omega) \subseteq \delta(\Omega \cdot Put_f)
\end{array}
$$

Acceptable $f \cdot \Omega \cdot Put_f \cdot (\Psi \cap \rho(f \cdot \Omega) \times \Phi \cap \Omega) \subseteq (\Psi \cap \rho(f \cdot \Omega)) \cdot \pi_1$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad f \cdot Put_f \cdot (\Psi \times \Phi) \subseteq \Psi \cdot \pi_1 \\
\Rightarrow \{\Phi \text{ - SMALLER}, \cdot \text{ - MONOTONICITY}\} \\
\quad \rho(f \cdot \Omega) \cdot f \cdot \Omega \cdot Put_f \cdot (\Psi \times \Phi) \cdot (\rho(f \cdot \Omega) \times \Omega)) \subseteq \rho(f \cdot \Omega) \cdot \Psi \cdot \pi_1 \\
\Leftrightarrow \{\rho \text{ - NEUTRAL}, \Phi \text{ - COMPOSITION}, \triangle \text{ - ABSORPTION}\} \\
\quad f \cdot \Omega \cdot Put_f \cdot (\Psi \cap \rho(f \cdot \Omega) \times \Phi \cap \Omega) \subseteq (\Psi \cap \rho(f \cdot \Omega)) \cdot \pi_1
\end{array}
$$

Stable $\Omega \cdot Put_f \cdot (f \triangle id) \cdot (\Phi \cap \Omega) \subseteq id$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad Put_f \cdot (f \triangle id) \cdot \Phi \subseteq id \\
\Rightarrow \{\Phi \text{ - SMALLER}, \Phi \text{ - COMPOSITION}\} \\
\quad \Omega \cdot Put_f \cdot (f \triangle id) \cdot (\Phi \cap \Omega) \subseteq id
\end{array}
$$

Get **preserves invariant** $f \cdot (\Phi \cap \Omega) \subseteq (\Psi \cap \rho(f \cdot \Omega)) \cdot f$

$$
\begin{array}{l}
\quad \{f \text{ lens}\} \\
\quad f \cdot \Phi \subseteq \Psi \cdot f \\
\Rightarrow \{\cdot \text{ - MONOTONICITY}\} \\
\quad f \cdot \Phi \cdot \Omega \subseteq \Psi \cdot f \cdot \Omega \\
\Rightarrow \{\rho \text{ - NEUTRAL}, \Phi \text{ - COMPOSITION}, \Phi \text{ - LARGER}\} \\
\quad f \cdot (\Phi \cap \Omega) \subseteq (\Psi \cap \rho(f \cdot \Omega)) \cdot f
\end{array}
$$

Put **preserves invariant**

$$\Omega \cdot Put_f \cdot ((\Psi \cap \rho(f \cdot \Omega)) \times (\Phi \cap \Omega)) \subseteq (\Phi \cap \Omega) \cdot \Omega \cdot Put_f$$

$$
\begin{array}{l}
\quad Put_f \cdot (\Psi \times \Phi) \subseteq \Phi \cdot Put_f \\
\Rightarrow \{\cdot \text{ - MONOTONICITY}\} \\
\quad \Omega \cdot Put_f \cdot (\Psi \times \Phi) \subseteq \Omega \cdot \Phi \cdot Put_f \\
\Rightarrow \{\Phi \text{ - COMPOSITION}, \Phi \text{ - REFLEXIVITY}, \Phi \text{ - SMALLER}\} \\
\quad \Omega \cdot Put_f \cdot ((\Psi \cap \rho(f \cdot \Omega)) \times (\Phi \cap \Omega)) \subseteq (\Phi \cap \Omega) \cdot \Omega \cdot Put_f
\end{array}
$$

□

$$\pi_1 : (A \times B)_{\prod_R} \trianglerighteq A_{\delta R}$$

$$Get = \pi_1$$
$$Put = \prod_R \cdot (id \times \pi_2) \cup Create \cdot \pi_1 \cdot \prod_{\overline{R}} \cdot (id \times \pi_2)$$
$$Create = \prod_R \cdot (id \triangle \top)$$

Proof.

Get **total** $\prod_R \subseteq \delta\pi_1$

Trivial since by (π - DOMAIN).

Put **total**

$$\delta R \times \prod_R \subseteq \delta((\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot id \times \pi_2)$$

Let's start by analyzing the subexpression:

$$\begin{aligned}
&\delta(\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \\
=&\{\delta/\rho \text{ - JOIN}, (5.19)\} \\
&\delta \prod_R \cup \delta((id \triangle R) \cdot (id \triangle \overline{R})^\circ) \\
=&\{\delta \text{ - COMPOSITION}, \triangle \text{ - DOMAIN}, \Phi \text{ - DOMAIN/RANGE}\} \\
&\prod_R \cup \delta(\delta R \cdot (id \triangle \overline{R})^\circ) \\
=&\{^\circ \text{ - COMPOSITION}, \delta/\rho \text{ - CONVERSE}\} \\
&\prod_R \cup \rho((id \triangle \overline{R}) \cdot \delta R) \\
=&\{\triangle \text{ - FUSION}, \triangle \text{ - RANGE}\} \\
&(\pi_2^\circ \cdot R \cdot \pi_1 \cap id) \cup (\pi_2^\circ \cdot \overline{R} \cdot \delta R \cdot \pi_1 \cap id) \\
=&\{\cup \text{ - DISTRIBUTIVITY}, \delta \text{ - NEUTRAL}\} \\
&\pi_2^\circ \cdot (R \cdot \delta R \cup \overline{R} \cdot \delta R) \cdot \pi_1 \cap id \\
=&\{\cup \text{ - DISTRIBUTIVITY}, \cup \text{ - COMPLEMENT}\} \\
&\pi_2^\circ \cdot \top \cdot \delta R \cdot \pi_1 \cap id
\end{aligned}$$

Considering $\pi_2^\circ \cdot \top \cdot \delta R \cdot \pi_1 \cap id = \prod_{\top \cdot \delta R}$, we have:

$$\begin{aligned}
&\delta R \times \prod_R \subseteq \delta(\prod_{\top \cdot \delta R} \cdot id \times \pi_2) \\
\Leftrightarrow&\{\delta \text{ - DEF}, \Phi \text{ - IN DOMAIN/RANGE}\} \\
&\delta R \times \prod_R \subseteq id \times \pi_2^\circ \cdot \prod_{\top \cdot \delta R} \cdot id \times \pi_2 \\
\Leftrightarrow&\{\text{SHUNTING}, \triangle \text{ - ABSORPTION}\} \\
&\delta R \times (\pi_2 \cdot \prod_R \cdot \pi_2^\circ) \subseteq \prod_{\top \cdot \delta R} \\
\Leftrightarrow&\{(5.19), \Phi \text{ - IN DOMAIN/RANGE}, ^\circ \text{ - MONOTONICITY}\} \\
&\delta R \times \rho R \subseteq \pi_1^\circ \cdot \delta R \cdot \top \cdot \pi_2
\end{aligned}$$

$\Leftarrow$ {SHUNTING, $\triangle$ - CANCELLATION, Φ - SMALLER}

$\delta R \cdot \pi_1 \subseteq \delta R \cdot \top \cdot \pi_2$

$\Leftarrow$ {π - DOMAINS, $\cdot$ - MONOTONICITY}

true

Acceptable $\pi_1 \cdot (\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R) \subseteq \pi_1$

$\pi_1 \cdot (\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R) \subseteq \pi_1$

$\Leftarrow$ {$\cup$ - DISTRIBUTIVITY, (5.19), Φ - SMALLER}

$\pi_1 \cdot ((id \times \pi_2) \cup \pi_1^\circ \cdot (id \triangle \overline{R})^\circ \cdot (id \times \pi_2)) \cdot (\delta R \times \prod_R) \subseteq \pi_1$

$\Leftrightarrow$ {$\cup$ - DISTRIBUTIVITY}

$(\pi_1 \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R)) \cup (\pi_1 \cdot \pi_1^\circ \cdot (id \triangle \overline{R})^\circ \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R)) \subseteq \pi_1$

$\Leftrightarrow$ {$\times$ - CANCELLATION, π - RANGE}

$\pi_1 \cdot (\delta R \times \prod_R) \cup ((id \triangle \overline{R})^\circ \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R)) \subseteq \pi_1$

$\Leftrightarrow$ {$\cup$ - UNIVERSAL, Φ - SMALLER, $\triangle$ - ABSORPTION}

$(id \triangle \overline{R})^\circ \cdot (\delta R \times \pi_2 \cdot \prod_R) \subseteq \pi_1$

$\Leftrightarrow$ {$\triangle$ - CONVERSE}

$\delta R \cdot \pi_1 \cap \overline{R}^\circ \cdot \pi_2 \cdot \prod_R \cdot \pi_2 \subseteq \pi_1$

$\Leftarrow$ {$\cap$ - SMALLER, Φ - SMALLER}

true

Stable $(\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot (id \times \pi_2) \cdot (\pi_1 \triangle id) \cdot \prod_R \subseteq id$

$(\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot (id \times \pi_2) \cdot (\pi_1 \triangle id) \cdot \prod_R \subseteq id$

$\Leftrightarrow$ {$\cup$ - DISTRIBUTIVITY, $\triangle$ - ABSORPTION, $\triangle$ - REFLECTION}

$\prod_R \cup (\prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}} \cdot \prod_R) \subseteq id$

$\Leftrightarrow$ {$\bot$ - COMPOSITION, (5.20)}

$\prod_R \cup \bot \subseteq id$

$\Leftrightarrow$ {Φ - DEF, $\bot$ - NEUTRAL}

true

Get **preserves invariant** $\pi_1 \cdot \prod_R \subseteq \delta R \cdot \pi_1$

$$\pi_1 \cdot \prod_R \subseteq \delta R \cdot \pi_1$$

$$
\begin{array}{l}
\Leftrightarrow\{5.19,\ \text{SHUNTING}\} \\
\quad (id \triangle R)^\circ \cdot \pi_1^\circ \subseteq \delta R \\
\Leftrightarrow\{\triangle\ \text{-}\ \text{CANCELLATION},\ \Phi\ \text{-}\ \text{CONVERSE}\} \\
\quad \delta R \subseteq \delta R
\end{array}
$$

Put **preserves invariant**

$$
(\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R) \subseteq \prod_R \cdot (\prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}}) \cdot (id \times \pi_2)
$$

By ($\cup$ - DISTRIBUTIVITY) and ($\cup$ - UNIVERSAL), the proof can be split in two:

$$
\begin{array}{l}
\prod_R \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R) \subseteq \prod_R \cdot (id \times \pi_2) \cup \prod_R \cdot \pi_1^\circ \cdot (id \triangle \overline{R})^\circ \cdot (id \times \pi_2) \\
\Leftarrow\{\cup\ \text{-}\ \text{LARGER},\ \Phi\ \text{-}\ \text{SMALLER}\} \\
\quad true
\end{array}
$$

$$
\begin{array}{l}
\prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}} \cdot (id \times \pi_2) \cdot (\delta R \times \prod_R) \subseteq \prod_R \cup \prod_R \cdot \pi_1^\circ \cdot \pi_1 \cdot \prod_{\overline{R}} \cdot (id \times \pi_2) \\
\Leftarrow\{\cup\ \text{-}\ \text{LARGER},\ \Phi\ \text{-}\ \text{SMALLER}\} \\
\quad true
\end{array}
$$

□

$$
\frac{f : A_\Phi \rhd B_\Omega \qquad g : A_\Phi \rhd C_\Lambda}{f \triangle g : A_\Phi \rhd (B \times C)_{\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega}}}
$$

$$
\begin{array}{rl}
Get &= Get_f \triangle Get_g \\
Put &= (Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f) \cup \\
&\quad Create \cdot \pi_1 \cdot \delta(Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f)_\neg \\
Create &= Create_f \cdot \pi_1 \cap Create_g \cdot \pi_2
\end{array}
$$

Proof.

Get **total** $\Phi \subseteq \delta(f \triangle g)$

$$
\begin{array}{l}
\quad \{f, g \text{ lenses}\} \\
\quad \Phi \subseteq \delta f \wedge \Phi \subseteq \delta g \\
\Leftrightarrow\{\cap\ \text{-}\ \text{UNIVERSAL}\} \\
\quad \Phi \subseteq \delta f \cap \delta g \\
\Leftrightarrow\{\triangle\ \text{-}\ \text{DOMAIN}\} \\
\quad \Phi \subseteq \delta(f \triangle g)
\end{array}
$$

Put **total**

$$
\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \subseteq \delta(Put_f \bullet Put_g \cup Create \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g))_\neg)
$$

Analyzing the right-hand subexpression individually:

$$
\begin{aligned}
& \delta(Put_f \bullet Put_g \cup Create \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g)_\neg) \\
= & \{\rho \text{ - COMPOSITION}, \Phi \text{ - COMPOSITION}\} \\
& \delta(Put_f \bullet Put_g \cup \delta(\delta(Create \cdot \pi_1) \cap \delta(Put_f \bullet Put_g)_\neg) \\
= & \{\delta/\rho \text{ - JOIN}, \Phi \text{ - DOMAIN/RANGE}\} \\
& \delta(Put_f \bullet Put_g) \cup (\delta(Create \cdot \pi_1) \cap \delta(Put_f \bullet Put_g)_\neg) \\
= & \{\cup/\cap \text{ - DISTRIBUTIVITY}, \Phi \text{ - NOT}\} \\
& \delta(Put_f \bullet Put_g) \cup \delta(Create \cdot \pi_1)
\end{aligned}
$$

, it suffices to prove, by ($\cup$ - LARGER) that $\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \subseteq \delta(Create \cdot \pi_1)$:

$$
\begin{aligned}
& \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \subseteq \delta(\Phi \cdot f^\circ \cdot \pi_1 \cdot \pi_1 \cap \Phi \cdot g^\circ \cdot \pi_2 \cdot \pi_1) \\
\Leftrightarrow & \{\delta/\rho \text{ - MEET}\} \\
& \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \subseteq \pi_1^\circ \cdot \pi_1^\circ \cdot f \cdot \Phi \cdot g^\circ \cdot \pi_2 \cdot \pi_1 \\
\Leftrightarrow & \{\times \text{ - TOP}\} \\
& \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \subseteq \prod_{g \cdot \Phi \cdot f^\circ} \times \top \\
\Leftarrow & \{\times \text{ - MONOTONICITY}\} \\
& true
\end{aligned}
$$

Acceptable

$$
(f \triangle g) \cdot ((Put_f \bullet Put_g) \cup Create \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g)_\neg) \cdot (\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi) \subseteq \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \cdot \pi_1
$$

By ($\cup$ - DISTRIBUTIVITY) and ($\cup$ - UNIVERSAL), the proof can be split in two. For simplification purposes, $\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega$ will be denoted by R.

$$
\begin{aligned}
& (f \triangle g) \cdot (Put_f \bullet Put_g) \cdot (\prod_R \times \Phi) \subseteq \prod_R \cdot \pi_1 \\
\Leftrightarrow & \{\triangle \text{ - FUSION}\} \\
& (f \cdot (Put_f \bullet Put_g) \triangle g \cdot (Put_f \bullet Put_g)) \cdot \prod_R \times \Phi \subseteq \prod_R \cdot \pi_1 \\
\Leftarrow & \{\cap \text{ - SMALLER}\} \\
& (f \cdot (Put_f \circledcirc Put_g) \cdot \prod_R \times \Phi) \triangle (g \cdot (Put_g \circledcirc Put_f) \cdot \prod_R \times \Phi) \subseteq \prod_R \cdot \pi_1
\end{aligned}
$$

Analyzing the sub-expressions individually, we have:

$$
\begin{aligned}
& f \cdot Put_f \cdot (\pi_1 \cdot \pi_1 \triangle Put_g \cdot \pi_2 \times id) \cdot \prod_R \times \Phi \\
= & \{\triangle \text{ - ABSORPTION}, \triangle \text{ - CANCELLATION}, (5.19)\} \\
& f \cdot Put_f \cdot ((id \triangle R)^\circ \cdot \pi_1 \cdot \delta(\Phi \cdot \pi_2) \times \Phi \triangle Put_g \cdot (R^\circ \triangle id)^\circ \times \Phi)
\end{aligned}
$$

$$
\begin{aligned}
&=\{\delta \text{ - NEUTRAL}, \rho \text{ - NEUTRAL}, {}^\circ \text{ - COMPOSITION}, \Phi \text{ - CONVERSE}\}\\
&\quad f \cdot Put_f \cdot (\delta R \cdot (id \triangle R)^\circ \cdot \pi_1 \cdot \delta(\Phi \cdot \pi_2) \triangle Put_g \cdot \rho R \cdot (R^\circ \triangle id)^\circ \times \Phi)\\
&\subseteq\{\delta \text{ - COMPOSITION}, \rho \text{ - COMPOSITION}\}\\
&\quad f \cdot Put_f \cdot (\Omega \cdot (id \triangle R)^\circ \cdot \pi_1 \cdot \delta(\Phi \cdot \pi_2) \triangle Put_g \cdot \Lambda \cdot (R^\circ \triangle id)^\circ \times \Phi)\\
&\subseteq\{\triangle \text{ - ABSORPTION}, g \text{ lens}\}\\
&\quad f \cdot Put_f \cdot (\Omega \cdot (id \triangle R)^\circ \cdot \pi_1 \cdot \delta(\Phi \cdot \pi_2) \triangle \Phi \cdot Put_g \cdot (R^\circ \triangle id)^\circ \times id)\\
&\subseteq\{\triangle \text{ - ABSORPTION}, f \text{ lens}\}\\
&\quad \Omega \cdot \pi_1 \cdot ((id \triangle R)^\circ \cdot \pi_1 \cdot \delta(\Phi \cdot \pi_2) \triangle Put_g \cdot (R^\circ \triangle id)^\circ \times id)\\
&\subseteq\{\triangle \text{ - CANCELLATION}, \Phi \text{ - SMALLER}\}\\
&\quad (id \triangle R)^\circ \cdot \pi_1
\end{aligned}
$$

The sub-expression $g \cdot (Put_g \circledcirc Put_f) \cdot (\prod_R \times \Phi)$ can be similarly analyzed. Returning to the full expression:

$$
\begin{aligned}
&(f \cdot (Put_f \circledcirc Put_g) \cdot \prod_R \times \Phi \triangle g \cdot (Put_g \circledcirc Put_f) \cdot \prod_R \times \Phi) \subseteq \prod_R \cdot \pi_1\\
&\Leftarrow\{\subseteq \text{ - TRANSITIVITY}\}\\
&\quad (id \triangle R)^\circ \cdot \pi_1 \triangle (R^\circ \triangle id)^\circ \cdot \pi_1 \subseteq \prod_R \cdot \pi_1\\
&\Leftrightarrow\{(5.19), \triangle \text{ - FUSION}\}\\
&\quad (\pi_1 \cdot \prod_R \triangle \pi_2 \cdot \prod_R) \cdot \pi_1 \subseteq \prod_R \cdot \pi_1\\
&\Leftrightarrow\{\triangle \text{ - FUSION}, \triangle \text{ - REFLECTION}\}\\
&\quad \prod_R \cdot \pi_1 \subseteq \prod_R \cdot \pi_1
\end{aligned}
$$

Now, the second part of the disjunction:

$$
\begin{aligned}
&(f \triangle g) \cdot \Phi \cdot (f \triangle g)^\circ \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g)_\neg) \cdot \prod_R \times \Phi \subseteq \prod_R \cdot \pi_1\\
&\Leftarrow\{\Phi \text{ - SMALLER}\}\\
&\quad (f \triangle g) \cdot (f \triangle g)^\circ \cdot \pi_1 \cdot \prod_R \times \Phi \subseteq \prod_R \cdot \pi_1\\
&\Leftarrow\{f, g \text{ lenses}, \triangle \text{ - CANCELLATION}, \Phi \text{ - SMALLER}\}\\
&\quad \prod_R \cdot \pi_1 \subseteq \prod_R \cdot \pi_1
\end{aligned}
$$

Stable

$$((Put_f \bullet Put_g) \cup Create \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g)_\neg) \cdot ((f \triangle g) \triangle id) \cdot \Phi \subseteq \Phi$$

Once again, by ($\cup$ - DISTRIBUTIVITY) and ($\cup$ - UNIVERSAL), the proof is

split in two:

$$
\begin{array}{ll}
& (Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f) \cdot ((f \triangle g) \triangle id) \cdot \Phi \subseteq \Phi \\
\Leftrightarrow & \{\cap \text{ - DISTRIBUTIVITY}\} \\
& (Put_f \circledcirc Put_g \cdot ((f \triangle g) \triangle id)) \cap (Put_g \circledcirc Put_f \cdot ((f \triangle g) \triangle id)) \cdot \Phi \subseteq \Phi \\
\Leftarrow & \{\cap \text{ - SMALLER}\} \\
& Put_f \cdot (\pi_1 \cdot \pi_1 \triangle Put_g \cdot (\pi_2 \times id)) \cdot ((f \triangle g) \triangle id) \cdot \Phi \subseteq \Phi \\
\Leftrightarrow & \{\triangle \text{ - FUSION}, \triangle \text{ - ABSORPTION}, \triangle \text{ - CANCELLATION}\} \\
& Put_f \cdot (f \cdot \delta g \triangle Put_g \cdot (g \cdot \delta f \triangle id)) \cdot \Phi \subseteq \Phi \\
\Leftarrow & \{\Phi \text{ - SMALLER}, \triangle \text{ - FUSION}\} \\
& Put_f \cdot (f \cdot \Phi \triangle Put_g \cdot (g \triangle id) \cdot \Phi) \subseteq \Phi \\
\Leftarrow & \{g \text{ lens}\} \\
& Put_f \cdot (f \cdot \Phi \triangle \Phi) \subseteq \Phi \\
\Leftarrow & \{f \text{ lens}\} \\
& \Phi \subseteq \Phi
\end{array}
$$

Since in this particular case, $Put_f \bullet Put_g$ is total, the *Create* case is never applied.

Get **preserves invariant** $(f \triangle g) \cdot \Phi \subseteq \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \cdot (f \triangle g)$

$$
\begin{array}{ll}
& (f \triangle g) \cdot \Phi \subseteq \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \cdot (f \triangle g) \\
\Leftrightarrow & \{\text{SHUNTING}, \cap \text{ - UNIVERSAL}, \Phi \text{ - DEF}\} \\
& (f \triangle g) \cdot \Phi \cdot (f \triangle g)^\circ \subseteq \pi_2^\circ \cdot \Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega \cdot \pi_1 \\
\Leftrightarrow & \{\text{SHUNTING}, \triangle \text{ - CANCELLATION}\} \\
& g \cdot \delta f \cdot \Phi \cdot \delta g \cdot f^\circ \subseteq \Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega \\
\Leftarrow & \{\Phi \text{ - SMALLER}, f, g \text{ lenses}\} \\
& g \cdot \Phi \cdot f^\circ \subseteq g \cdot \Phi \cdot f^\circ
\end{array}
$$

Put **preserves invariant**

$$
((Put_f \bullet Put_g) \cup Create \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g)_\neg) \cdot \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \subseteq
$$
$$
\Phi \cdot ((Put_f \bullet Put_g) \cup Create \cdot \pi_1 \cdot \delta(Put_f \bullet Put_g)_\neg)
$$

Once again, by ($\cup$ - DISTRIBUTIVITY) and ($\cup$ - UNIVERSAL), the proof is split in two. Moreover, since the right-hand side is also a disjunction, by ($\cup$ - LARGER) it suffices to prove:

$$
(Put_f \bullet Put_g) \cdot (\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi) \subseteq \Phi \cdot (Put_f \bullet Put_g)
$$

By similar reasons, this proof can be split in two ($\cap$ - UNIVERSAL). For the first one, by ($\cap$ - SMALLER) it suffices to prove:

$$
(Put_f \circledcirc Put_g) \cdot (\prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi) \subseteq \Phi \cdot (Put_f \circledcirc Put_g)
$$

For simplifications purposes, $\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega$ will be denoted by R.

$$
\begin{aligned}
& Put_f \cdot (\pi_1 \cdot \pi_1 \cdot (\prod_R \times \Phi) \triangle Put_g \cdot (\pi_2 \times id)) \cdot (\prod_R \times \Phi) \\
= & \{\triangle \text{ - FUSION}, \triangle \text{ - ABSORPTION}, (5.19)\} \\
& Put_f \cdot ((id \triangle R)^\circ \cdot \delta(\Phi \cdot \pi_2) \triangle Put_g \cdot ((R^\circ \triangle id)^\circ \times \Phi)) \\
\subseteq & \{\delta \text{ - NEUTRAL}, \rho \text{ - NEUTRAL}, \delta \text{ - COMPOSITION}, \rho \text{ - COMPOSITION}\} \\
& Put_f \cdot (\Omega \cdot (id \triangle R)^\circ \cdot \delta(\Phi \cdot \pi_2) \triangle Put_g \cdot (\Lambda \cdot (R^\circ \triangle id)^\circ \times \Phi)) \\
\subseteq & \{\triangle \text{ - ABSORPTION}, g \text{ lens}\} \\
& Put_f \cdot (\Omega \cdot (id \triangle R)^\circ \cdot \delta(\Phi \cdot \pi_2) \triangle \Phi \cdot Put_g \cdot ((R^\circ \triangle id)^\circ \times \Phi)) \\
\subseteq & \{\triangle \text{ - ABSORPTION}, f \text{ lens}\} \\
& \Phi \cdot Put_f \cdot ((id \triangle R)^\circ \cdot \delta(\Phi \cdot \pi_2) \triangle Put_g \cdot ((R^\circ \triangle id)^\circ \times \Phi)) \\
= & \{\triangle \text{ - ABSORPTION}, (5.19)\} \\
& \Phi \cdot Put_f \cdot (\pi_1 \cdot \pi_1 \cdot (\prod_R \times \Phi) \triangle Put_g \cdot (\pi_2 \cdot \prod_R \times \Phi)) \\
\subseteq & \{\Phi \text{ - SMALLER}\} \\
& \Phi \cdot Put_f \cdot (\pi_1 \cdot \pi_1 \triangle Put_g \cdot (\pi_2 \times id))
\end{aligned}
$$

The proof of the second part of the intersection is similar.

As for the second part of the join, we have:

$$
\begin{aligned}
& \Phi \cdot (f \triangle g)^\circ \cdot \pi_1 \cdot \delta(Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f)_\neg \cdot \prod_{\Lambda \cdot g \cdot \Phi \cdot f^\circ \cdot \Omega} \times \Phi \\
\subseteq & \{\Phi \text{ - SMALLER}\} \\
& \Phi \cdot (f \triangle g)^\circ \cdot \pi_1 \cdot \delta(Put_f \circledcirc Put_g \cap Put_g \circledcirc Put_f)_\neg
\end{aligned}
$$

□

Lemma A.1. *For a r-lens $f : A_\Phi \trianglerighteq B_\Psi$, if the invariant on the view is further restricted by a coreflexive Ω, the range of Put is restricted to sources whose views are in Ω,i.e., $\delta(\Omega \cdot Get)$:*

$$Put \cdot ((\Psi \cap \Omega) \times \Phi) \subseteq (\Phi \cap \delta(\Omega \cdot Get)) \cdot Put \tag{A.1}$$

Proof. Since

$$
\begin{aligned}
& \{f \text{ lens}\} \\
& Get \cdot Put \cdot (\Psi \times \Phi) \subseteq \Psi \cdot \pi_1 \\
\Rightarrow & \{\cdot \text{ - MONOTONICITY}, \triangle \text{ - ABSORPTION}\} \\
& Get \cdot Put \cdot ((\Psi \cap \Omega) \times \Phi) \subseteq \Psi \cdot \pi_1 \cdot (\Omega \times id) \\
\Rightarrow & \{\cdot \text{ - MONOTONICITY}, {}^\circ \text{ - MONOTONICITY}\} \\
& Get \cdot Put \cdot ((\Psi \cap \Omega) \times \Phi) \cdot Put^\circ \cdot Get^\circ \subseteq \Psi \cdot \pi_1 \cdot (\Omega \times id) \cdot \pi_1^\circ \cdot \Psi \\
\Rightarrow & \{\triangle \text{ - CANCELLATION}, \text{SHUNTING}\} \\
& Put \cdot ((\Psi \cap \Omega) \times \Phi) \cdot Put^\circ \subseteq Get^\circ \cdot (\Psi \cap \Omega) \cdot Get \\
\Rightarrow & \{\cap \text{ - MONOTONICITY}, \delta \text{ - DEF}, \rho \text{ - DEF}, \Phi \text{ - LARGER}\} \\
& \rho(Put \cdot ((\Psi \cap \Omega) \times \Phi)) \subseteq \delta(\Omega \cdot Get)
\end{aligned}
$$

and

$$
\begin{aligned}
&\{f \text{ lens}\}\\
&Put \cdot (\Psi \times \Phi) \subseteq \Phi \cdot Put\\
\Rightarrow&\{\Phi \text{ - SMALLER}, \rho \text{ - MONOTONICITY}\}\\
&\rho(Put \cdot ((\Psi \cap \Omega) \times \Phi)) \subseteq \rho(\Phi \cdot Put)\\
\Rightarrow&\{\rho \text{ - COMPOSITION}\}\\
&\rho(Put \cdot ((\Psi \cap \Omega) \times \Phi)) \subseteq \Phi
\end{aligned}
$$

we have

$$
\begin{aligned}
&\{\cap \text{ - UNIVERSAL}\}\\
&\rho(Put \cdot ((\Psi \cap \Omega) \times \Phi)) \subseteq \Phi \cap \delta(\Omega \cdot Get)\\
\Rightarrow&\{\cdot \text{ - MONOTONICITY}, \rho \text{ - NEUTRAL}\}\\
&Put \cdot ((\Psi \cap \Omega) \times \Phi) \subseteq (\Phi \cap \delta(\Omega \cdot Get)) \cdot Put \cdot ((\Psi \cap \Omega) \times \Phi)\\
\Rightarrow&\{\Phi \text{ - LARGER}\}\\
&Put \cdot ((\Psi \cap \Omega) \times \Phi) \subseteq (\Phi \cap \delta(\Omega \cdot Get)) \cdot Put
\end{aligned}
$$

□

Appendix B

RAs Laws

Inclusion $\subseteq$

$$R \subseteq S \wedge S \subseteq T \Rightarrow R \subseteq T \qquad (\subseteq \text{ - TRANSITIVITY})$$

$$R = S \equiv R \subseteq S \wedge S \subseteq R \qquad (= \text{ - DEF})$$
$$f \subseteq g \equiv f = g \equiv g \subseteq f \qquad (\subseteq \text{ - FUNCTIONAL})$$

Composition $\cdot$

$$R \subseteq S \wedge T \subseteq U \Rightarrow R \cdot T \subseteq S \cdot U \qquad (\cdot \text{ - MONOTONICITY})$$
$$R \subseteq S \Rightarrow R \cdot T \subseteq S \cdot T$$
$$R \subseteq S \Rightarrow T \cdot R \subseteq T \cdot S$$

$$f \cdot R \subseteq S \equiv R \subseteq f^{\circ} \cdot S \qquad (\text{SHUNTING})$$
$$R \cdot f^{\circ} \subseteq S \equiv R \subseteq \cdot S \cdot f$$

$$S \cdot R \subseteq T \equiv (\delta S) \cdot R \subseteq S^{\circ} \cdot T \Leftarrow \text{img } S \subseteq id \qquad (\text{SHUNTING (SIMPLE)})$$
$$R \cdot S^{\circ} \subseteq T \equiv R \cdot (\delta S) \subseteq T \cdot S \Leftarrow \text{img } S \subseteq id$$

Meet $\cap$

$$X \subseteq (R \cap S) \equiv (X \subseteq R) \wedge (X \subseteq S) \qquad (\cap \text{ - UNIVERSAL})$$

$$R \subseteq S \Rightarrow R \cap T \subseteq S \cap T \qquad (\cap \text{ - MONOTONICITY})$$

$$R \subseteq S \Rightarrow R \cap T \subseteq S \qquad (\cap \text{ - SMALLER})$$
$$R \cap S \subseteq R$$

$$R \cap \overline{R} = \bot \qquad (\cap \text{ - COMPLEMENT})$$

$$(R \cap S) \cdot T = (R \cdot T) \cap (S \cdot T) \Leftarrow R \cdot \text{img } T \subseteq R \vee S \cdot \text{img } T \subseteq S$$
$$(R \cap S) \cdot f = (R \cdot f) \cap (S \cdot f) \qquad (\cap \text{ - DISTRIBUTIVITY})$$
$$T \cdot (R \cap S) = (T \cdot R) \cap (T \cdot S) \Leftarrow \ker T \cdot R \subseteq R \vee \ker T \cdot S \subseteq S$$

Join $\cup$

$$R \cup S \subseteq T \equiv R \subseteq T \wedge S \subseteq T \qquad (\cup \text{ - UNIVERSAL})$$

$$(R \cup S) \cdot T = (R \cdot T) \cup (S \cdot T) \qquad (\cup \text{ - DISTRIBUTIVITY})$$
$$T \cdot (R \cup S) = (T \cdot R) \cup (T \cdot S)$$

$$R \subseteq S \Rightarrow R \subseteq S \cup T \qquad (\cup \text{ - LARGER})$$
$$R \subseteq R \cup S$$

$$R \cup \overline{R} = \top \qquad (\cup \text{ - COMPLEMENT})$$

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T) \qquad (\cap/\cup \text{ - DISTRIBUTIVITY})$$
$$(R \cup S) \cap T = (R \cap T) \cup (S \cap T)$$
$$R \cup (S \cap T) = (R \cup S) \cap (R \cup T) \qquad (\cup/\cap \text{ - DISTRIBUTIVITY})$$
$$(R \cap S) \cup T = (R \cup T) \cap (S \cup T)$$

Converse $^\circ$

$$R \subseteq S \Rightarrow R^\circ \subseteq S^\circ \qquad (^\circ \text{ - MONOTONICITY})$$

$$(R^\circ)^\circ = R \qquad (^\circ \text{ - INVOLUTION})$$
$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \qquad (^\circ \text{ - COMPOSITION})$$
$$(R \cap S)^\circ = R^\circ \cap S^\circ \qquad (^\circ \text{ - MEET})$$
$$(R \cup S)^\circ = R^\circ \cup S^\circ \qquad (^\circ \text{ - JOIN})$$

Domain δ **— Range** ρ

$$\ker R = R^\circ \cdot R \qquad (\ker \text{ - DEF})$$
$$\text{img } R = R \cdot R^\circ \qquad (\text{img - DEF})$$
$$\delta R = (R^\circ \cdot R) \cap id \qquad (\delta \text{ - DEF})$$
$$\rho R = (R \cdot R^\circ) \cap id \qquad (\rho \text{ - DEF})$$

$$\delta(R \cdot S) = \delta(\delta R \cdot S) \qquad (\delta \text{ - COMPOSITION})$$
$$\delta(R \cdot S) \subseteq \delta S$$
$$\rho(R \cdot S) = \rho(R \cdot \rho S) \qquad (\rho \text{ - COMPOSITION})$$
$$\rho(R \cdot S) \subseteq \rho R$$
$$\delta(R^\circ) = \rho R \quad \wedge \quad \rho(R^\circ) = \delta R \qquad (\delta/\rho \text{ - CONVERSE})$$

$$\delta(R \cup S) = \delta R \cup \delta S \quad \wedge \quad \rho(R \cup S) = \rho R \cup \rho S \qquad (\delta/\rho \text{ - JOIN})$$

$$\delta(R \cap (S \cdot T)) = (S^\circ \cdot R) \cap T \ \wedge \ \rho(R \cap (S \cdot T)) = (R \cdot T^\circ) \cap Serro?? \qquad (\delta/\rho \text{ - MEET})$$

$$\delta(R \cap S) = (R^\circ \cdot S) \cap id \wedge \rho(R \cap S) = (R \cdot S^\circ) \cap id$$

$$R \subseteq S \Rightarrow \delta R \subseteq \delta S \qquad (\delta \text{ - MONOTONICITY})$$
$$R \subseteq S \Rightarrow \rho R \subseteq \rho S \qquad (\rho \text{ - MONOTONICITY})$$
$$\rho R \cdot R = R \qquad (\rho \text{ - NEUTRAL})$$
$$R \cdot \delta R = R \qquad (\delta \text{ - NEUTRAL})$$

Coreflexives Φ

$$\Phi \subseteq id \qquad (\Phi \text{ - DEF})$$

$$R \subseteq S \cdot \Phi \cdot T \Rightarrow R \subseteq S \cdot T \qquad (\Phi \text{ - LARGER})$$
$$R \cdot S \subseteq T \Rightarrow R \cdot \Phi \cdot S \subseteq T \qquad (\Phi \text{ - SMALLER})$$

$$\Phi \cdot \Psi = \Phi \cap \Psi \qquad (\Phi \text{ - COMPOSITION})$$
$$\Phi \cdot \Phi = \Phi \qquad (\Phi \text{ - REFLEXIVITY})$$
$$\Phi^\circ = \Phi \qquad (\Phi \text{ - CONVERSE})$$
$$\rho\Phi = \Phi = \delta\Phi \qquad (\Phi \text{ - DOMAIN/RANGE})$$
$$\Phi \cdot \Phi_\neg = \bot \qquad (\Phi \text{ - NOT})$$

$$\Phi \subseteq \delta R \equiv \Phi \subseteq \ker R \qquad (\Phi \text{ - IN DOMAIN/RANGE})$$
$$\Phi \subseteq \rho R \equiv \Phi \subseteq \text{img } R$$

$$R \cdot \Phi \subseteq S \equiv R \cdot \Phi \subseteq S \cdot \Phi \qquad (\Phi \text{ - SHUNTING})$$
$$\Phi \cdot R \subseteq S \equiv \Phi \cdot R \subseteq \Phi \cdot S$$

Bottom $\bot$

$$\bot \subseteq R \qquad (\bot \text{ - DEF})$$
$$R \cdot \bot = \bot = \bot \cdot R \qquad (\bot \text{ - COMPOSITION})$$
$$R \cup \bot = R \qquad (\bot \text{ - NEUTRAL})$$
$$R \cap \bot = \bot \qquad (\bot \text{ - ABSORPTION})$$

Top $\top$

$$R \subseteq \top \qquad (\top \text{ - DEF})$$
$$R \cup \top = \top \qquad (\top \text{ - ABSORPTION})$$
$$R \cap \top = R \qquad (\top \text{ - NEUTRAL})$$
$$\top \cdot \delta R = \top \cdot R \qquad (\delta \text{ - ELIMINATION})$$

$$\rho R \cdot \top = R \cdot \top \qquad (\rho \text{ - ELIMINATION})$$

Product $\triangle|\times$

$$X \subseteq R \triangle S \equiv \pi_1 \cdot X \subseteq R \wedge \pi_2 \cdot X \subseteq S \qquad (\triangle \text{ - UNIVERSAL})$$

$$R \triangle S \triangleq \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S \qquad (\triangle \text{ - DEF})$$
$$R \times S \triangleq R \cdot \pi_1 \triangle S \cdot \pi_2 \qquad (\times \text{ - DEF})$$
$$\pi_1 \triangleq (id \triangle \top)^\circ \quad \wedge \quad \pi_2 \triangleq (\top \triangle id)^\circ \qquad (\pi \text{ - DEF})$$

$$R \subseteq S \Rightarrow R \triangle T \subseteq S \triangle T \qquad (\triangle \text{ - MONOTONICITY})$$
$$R \subseteq S \wedge T \subseteq U \Rightarrow R \times T \subseteq S \times U \qquad (\times \text{ - MONOTONICITY})$$

$$(R \times S) \cdot (T \triangle U) = (R \cdot T) \triangle (S \cdot U) \qquad (\triangle \text{ - ABSORPTION})$$
$$(R \times S) \cdot (T \times U) = (R \cdot T) \times (S \cdot U)$$

$$\pi_1 \cdot (R \triangle S) = R \cdot \delta S \quad \wedge \quad \pi_2 \cdot (R \triangle S) = S \cdot \delta R \qquad (\triangle \text{ - CANCELLATION})$$
$$\pi_1 \cdot (f \triangle g) = f \quad \wedge \quad \pi_2 \cdot (f \triangle g) = g$$
$$\pi_1 \cdot (R \times S) = R \cdot \pi_1 \cdot (id \times \delta S) \qquad (\times \text{ - CANCELLATION})$$
$$\pi_2 \cdot (R \times S) = S \cdot \pi_2 \cdot (\delta R \times id)$$

$$\pi_1 \triangle \pi_2 = id \qquad (\triangle \text{ - REFLECTION})$$

$$(R \triangle S) \cdot T = R \cdot T \triangle S \cdot T \Leftarrow R \cdot \operatorname{img} T \subseteq R \vee S \cdot \operatorname{img} T \subseteq S \qquad (\triangle \text{ - FUSION})$$
$$(R \triangle S) \cdot f = (R \cdot f) \triangle (S \cdot f)$$
$$(R \triangle S) \cdot \Phi = (R \cdot \Phi) \triangle (S \cdot \Phi)$$

$$(R \triangle S)^\circ \cdot (T \triangle U) = (R^\circ \cdot T) \cap (S^\circ \cdot U) \qquad (\triangle \text{ - CONVERSE})$$

$$\delta \pi_1 = id = \delta \pi_2 \qquad (\pi \text{ - DOMAIN})$$
$$\delta(R \triangle S) = \delta R \cap \delta S \qquad (\triangle \text{ - DOMAIN})$$
$$\delta(R \times S) = \delta R \times \delta S \qquad (\times \text{ - DOMAIN})$$
$$\delta((R \times S) \cdot \pi_1^\circ) = \delta R \Leftarrow S \neq \bot \quad \wedge \quad \delta((R \times S) \cdot \pi_2^\circ) = \delta S \Leftarrow R \neq \bot \qquad \text{(B.1)}$$

$$\rho \pi_1 = id = \rho \pi_2 \qquad (\pi \text{ - RANGE})$$
$$\rho(R \triangle S) = (\pi_1^\circ \cdot R \cdot S^\circ \cdot \pi_2) \cap id \qquad (\triangle \text{ - RANGE})$$
$$\rho(R \times S) = \rho R \times \rho S \qquad (\times \text{ - RANGE})$$
$$\rho(\pi_1 \cdot (R \times S)) = \rho R \Leftarrow S \neq \bot \quad \wedge \quad \rho(\pi_2 \cdot (R \times S)) = \rho S \Leftarrow R \neq \bot \qquad \text{(B.2)}$$

$$\pi_1 \subseteq \top \cdot \pi_2 \quad \wedge \quad \pi_2 \subseteq \top \cdot \pi_1 \qquad (\pi \text{ - DOMAINS})$$
$$\pi_1 \cdot \pi_2^\circ = \top = \pi_2 \cdot \pi_1^\circ \qquad \text{(B.3)}$$

$$\pi_1^\circ \cdot R = R \triangle \top \ \wedge \ \pi_2^\circ \cdot R = \top \triangle R \qquad (\triangle \text{ - TOP})$$
$$\pi_1^\circ \cdot R \cdot \pi_1 = R \times \top \ \wedge \ \pi_2^\circ \cdot R \cdot \pi_2 = \top \times R \qquad (\times \text{ - TOP})$$

$$R \triangle S \cap P \triangle Q = (R \cap P) \triangle (S \cap Q)$$
$$(\triangle/\cap \text{ - DISTRIBUTIVITY})$$
$$R \times S \cap P \times Q = (R \cap P) \times (S \cap Q)$$
$$(\times/\cap \text{ - DISTRIBUTIVITY})$$
$$(R \cup S) \triangle (P \cup Q) = R \triangle P \cup R \triangle Q \cup S \triangle P \cup S \triangle Q$$
$$(R \cup S) \triangle P = (R \triangle P) \cup (S \triangle P)$$
$$(\triangle/\cup \text{ - DISTRIBUTIVITY})$$
$$R \triangle (P \cup Q) = (R \triangle P) \cup (R \triangle Q)$$
$$(R \cup S) \times (P \cup Q) = R \times P \cup R \times Q \cup S \times P \cup S \times Q$$
$$R \cup (S \times P) = R \times P \cup S \times P \quad (\times/\cup \text{ - DISTRIBUTIVITY})$$
$$R \times (P \cup Q) = R \times P \cup R \times Q$$

$$id \triangle R = \bot \equiv \delta R = \bot \quad \wedge \quad R \triangle id = \bot \equiv \delta R = \bot$$
$$R \triangle S = \bot \equiv R \cdot S^\circ = \bot \qquad (\triangle \text{ - BOTTOM})$$
$$R \triangle S = \bot \Leftarrow R = \bot \vee S = \bot$$
$$id \times R = \bot \equiv \delta R = \bot \quad \wedge \quad id \times R = \bot \equiv \delta R = \bot$$
$$R \times S = \bot \equiv R \cdot \top \cdot S^\circ = \bot \qquad (\times \text{ - BOTTOM})$$
$$R \times S = \bot \Leftarrow R = \bot \vee S = \bot$$

Product $\nabla|+$

$$X = R \nabla S \equiv X \cdot in_1 = R \wedge X \cdot in_2 = S \qquad (\nabla \text{ - UNIVERSAL})$$

$$R \nabla S \triangleq R \cdot in_1^\circ \cup S \cdot in_2^\circ \qquad (\nabla \text{ - DEF})$$
$$R + S \triangleq in_1 \cdot R \nabla in_2 \cdot S \qquad (+ \text{ - DEF})$$
$$in_1 \triangleq (id \triangle \bot)^\circ \quad \wedge \quad in_2 \triangleq (\bot \triangle id)^\circ \qquad (in \text{ - DEF})$$

$$R \subseteq S \Rightarrow R \nabla T \subseteq S \nabla T \qquad (\nabla \text{ - MONOTONICITY})$$
$$R \subseteq S \wedge T \subseteq U \Rightarrow R + T \subseteq S + U \qquad (+ \text{ - MONOTONICITY})$$

$$(R \nabla S) \cdot (T + U) = (R \cdot T) \nabla (S \cdot U) \qquad (\nabla \text{ - ABSORPTION})$$
$$(R + S) \cdot (T + U) = (R \cdot T) + (S \cdot U)$$

$$(R \nabla S) \cdot in_1 = R \quad \wedge \quad (R \nabla S) \cdot in_1 = S \qquad (\nabla \text{ - CANCELLATION})$$

$$(R+S)\cdot in_1 = in_1 \cdot R \quad \wedge \quad (R+S)\cdot in_2 = in_2 \cdot S \qquad (+ \text{ - CANCELLATION})$$

$$in_1 \nabla in_2 = id \qquad (\nabla \text{ - REFLECTION})$$

$$T \cdot (R \nabla S) \cdot T = T \cdot R \nabla T \cdot S \qquad (\nabla \text{ - FUSION})$$

$$(R \nabla S) \cdot (T \nabla U)^\circ = (R \cdot T^\circ) \cup (S \cdot U^\circ) \qquad (\nabla \text{ - CONVERSE})$$

$$\delta in_1 = id = \delta in_2 \qquad (in \text{ - DOMAIN})$$
$$\delta(R \nabla S) = \delta R + \delta S \qquad (\nabla \text{ - DOMAIN})$$
$$\delta(R+S) = \delta R + \delta S \qquad (+ \text{ - DOMAIN})$$

$$\rho in_1 = id + \bot \quad \wedge \quad \rho in_2 = \bot + id \qquad (in \text{ - RANGE})$$
$$\rho(R \triangle S) = \rho R \cup \rho S \qquad (\nabla \text{ - RANGE})$$
$$\rho(R+S) = \rho R + \rho S \qquad (+ \text{ - RANGE})$$

$$\rho in_1 \cap \rho in_2 = \bot \qquad \text{(B.4)}$$
$$in_1 \cap in_2 = \bot \qquad \text{(B.5)}$$
$$in_1^\circ \cdot in_2 = \bot = in_2^\circ \cdot in_1 \qquad \text{(B.6)}$$

$$R \cdot in_1^\circ = R \nabla \bot \ \wedge \ R \cdot in_2^\circ = \bot \nabla R \qquad (\nabla \text{ - TOP})$$
$$in_1 \cdot R \cdot in_1^\circ = R + \bot \ \wedge \ in_2 \cdot R \cdot in_2^\circ = \bot + R \qquad (+ \text{ - TOP})$$

$$(R \nabla S) \cap (P \nabla Q) = (R \cap P) \nabla (S \cap Q) \qquad (\nabla/\cap \text{ - DISTRIBUTIVITY})$$
$$(R+S) \cap (P+Q) = (R \cap P) + (S \cap Q) \qquad (+/\cap \text{ - DISTRIBUTIVITY})$$
$$(R \nabla S) \cup (P \nabla Q) = (R \cup P) \nabla (S \cup Q) \qquad (\nabla/\cup \text{ - DISTRIBUTIVITY})$$
$$(R+S) \cup (P+Q) = (R \cup P) + (S \cup Q) \qquad (+/\cup \text{ - DISTRIBUTIVITY})$$

$$R \nabla S = \bot \equiv R = \bot \wedge S = \bot \qquad (\nabla \text{ - BOTTOM})$$
$$R + S = \bot \equiv R = \bot \wedge S = \bot \qquad (+ \text{ - BOTTOM})$$

$$(R \nabla S) \triangle (T \nabla U) = (R \triangle T) \nabla (S \triangle U) \qquad (\triangle/\nabla \text{ - DISTRIBUTIVITY})$$

Division \|/

$$R \cdot S \subseteq T \equiv S \subseteq R \backslash T \qquad (\backslash \text{ - DEFINITION})$$
$$R \cdot S \subseteq T \equiv R \subseteq T / S \qquad (/ \text{ - DEFINITION})$$

$$(R/S)^\circ = S^\circ \backslash R^\circ \qquad (\backslash \text{ - CONVERSE})$$
$$(R \backslash S)^\circ = S^\circ / R^\circ \qquad (/ \text{ - CONVERSE})$$

$$R \subseteq S \Rightarrow T\backslash R \subseteq T\backslash S \qquad (\backslash \text{ - MONOTONIC})$$
$$S \subseteq R \Rightarrow R\backslash T \subseteq S\backslash T$$
$$R \subseteq S \Rightarrow R/T \subseteq S/T \qquad (/ \text{ - MONOTONIC})$$
$$S \subseteq R \Rightarrow T/R \subseteq T/S$$

$$R\backslash S \cap R\backslash T = R\backslash(S \cap T)$$
$$S\backslash R \cap T\backslash R = (S \cup T)\backslash R \qquad (\bigvee \cap / \cup \text{ - DISTRIBUTIVITY})$$
$$R\backslash U \cap T\backslash U \cap T\backslash S \cap R\backslash S = (R \cup T)\backslash(S \cap U)$$
$$S/R \cap T/R = (S \cap T)/R$$
$$R/S \cap R/T = R/(S \cup T) \qquad (// \cap / \cup \text{ - DISTRIBUTIVITY})$$
$$R/U \cap T/U \cap T/S \cap R/S = (R \cap T)/(S \cup U)$$

$$\overline{R\backslash S} = R^{\circ} \cdot \overline{S} \qquad (\backslash \text{ - COMPLEMENT})$$
$$\overline{R/S} = \overline{R} \cdot S^{\circ} \qquad (/ \text{ - COMPLEMENT})$$

$$id\backslash R = R \qquad (\backslash \text{ - NEUTRAL})$$
$$\bot\backslash R = \top \qquad (\backslash \text{ - ABSORPTION})$$
$$R\backslash\top = \top$$
$$R/id = R \qquad (/ \text{ - NEUTRAL})$$
$$R/\bot = \top \qquad (/ \text{ - NEUTRAL})$$
$$\top/R = \top$$

$$id \subseteq R\backslash R \qquad \text{(B.7)}$$
$$\mathrm{img}R = id \Rightarrow id = R\backslash R$$
$$id \subseteq R/R \qquad \text{(B.8)}$$
$$\ker R = id \Rightarrow id = R/R$$

$$R \cdot (R\backslash S) \subseteq S \qquad (\backslash \text{ - CANCELLATION})$$
$$S \subseteq R\backslash(R \cdot S)$$
$$(S/R) \cdot R \subseteq S \qquad (/ \text{ - CANCELLATION})$$
$$S \subseteq (S \cdot R)/R$$

$$R\backslash(S\backslash T) = (S \cdot R)\backslash T \qquad (\backslash \text{ - ASSOCIATIVITY})$$
$$(R/S)/T = R/(T/S) \qquad (/ \text{ - ASSOCIATIVITY})$$

$$(R\backslash S)\cdot f = R\backslash(S\cdot f) \qquad (\backslash \text{ - FUSION})$$
$$f^\circ\cdot(R\backslash S) = (R\cdot f)\backslash S$$
$$(R/S)\cdot f = R/(f^\circ\cdot S) \qquad (/ \text{ - FUSION})$$
$$f^\circ\cdot(R/S) = (f^\circ\cdot R)\backslash S$$

$$R\backslash S\cap S\backslash T\cap id\subseteq R\backslash T \qquad (\backslash \text{ - TRANSITIVITY})$$
$$R/S\cap S/T\cap id\subseteq R/T \qquad (/ \text{ - TRANSITIVITY})$$

$$R\backslash S = R\backslash(R\cdot(R\backslash S)) \tag{B.9}$$
$$S/R = ((S/R)\cdot R)/R \tag{B.10}$$
$$R\cdot(R\backslash(R\cdot S)) = R\cdot S \tag{B.11}$$
$$((S\cdot R)/R)\cdot R = S\cdot R \tag{B.12}$$
$$\phi\cdot((R\cdot\phi)\backslash S) = \phi\cdot(R\backslash S) \tag{B.13}$$
$$(R\backslash(\phi\cdot S))\cdot\phi = (R/S)\cdot\phi \tag{B.14}$$