Relational semigroupoids: Abstract relation-algebraic interfaces for finite relations between infinite types

Wolfram Kahl *

Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario, Canada L8S 4K1

Available online 13 February 2008

Abstract

Finite maps or finite relations between infinite sets do not even form a category, since the necessary identities are not finite. We show relation-algebraic extensions of semigroupoids where the operations that would produce infinite results have been replaced with variants that preserve finiteness, but still satisfy useful algebraic laws. The resulting theories allow calculational reasoning in the relation-algebraic style with only minor sacrifices; our emphasis on generality even provides some concepts in theories where they had not been available before.

The semigroupoid theories presented in this paper also can directly guide library interface design and thus be used for principled relation-algebraic programming; an example implementation in Haskell allows manipulating finite binary relations as data in a point-free relation-algebraic programming style that integrates naturally with the current Haskell collection types. This approach enables seamless integration of relation-algebraic formulations to provide elegant solutions of problems that, with different data organisation, are awkward to tackle.

© 2007 Elsevier Inc. All rights reserved.

AMS classification: 18B40; 18B10; 03G15; 04A05; 68R99

Keywords: Finite relations; Semigroupoids; Determinacy; Restricted residuals; Direct products; Functional programming; Haskell

1. Introduction

The motivation for this paper arose from the desire to use well-defined algebraic interfaces for collection datatypes implementing sets, partial functions, and binary relations. Even though set libraries typically provide natural algebraic operations like union, intersection, and difference, libraries providing partial function datatypes (frequently called “map” or “dictionary”) apparently rarely even provide function composition, arguably the most natural operation on functions.
Continuing in this vein, a mathematically minded user might expect a Boolean algebra interface for sets, a category interface for functions, and a relation algebra interface for binary relations.

However, most collection libraries are inherently limited to representing only finite collections, even if the types involved are infinite. Many “natural” algebraic operations on the types of collections we are interested in are not closed on finite arguments. For example, the complement of a finite subset of an infinite type will be infinite. Identity functions and identity relations on infinite types are infinite. Residuals of relations, arising from translations of predicate-logic implications into the language of relation algebra, introduce infinity, just like complement of relations.

Reasoning in the presence of all these operations therefore becomes encumbered with side-conditions asserting finiteness of the results of operations.

The better solution is to abandon such “dangerous” operations and replace them with “well-behaved” relatives that still satisfy a coherent set of algebraic laws, just as, for example, set libraries do not provide a unary complement operation, but do provide a binary difference.

By carefully “weakening” the algebraic theories in this way, and by abstracting common subtheories into more general interfaces, libraries can achieve a high degree of “conceptual coherence”, which can significantly help correctness arguments for abstract, polymorphic algorithms.

This paper contributes an exploration of the theories most likely to be useful for reasoning in an abstract way about finite relations, and relation-like structures, between (potentially) infinite types. These theories are adaptations of the various allegories of Freyd and Scedrov [18], and of different variants of Kleene algebras as axiomatised by Kozen [28,29], and by Desharnais, Möller, and Struth [15,16]. The general pattern of our adaptations is that we omit the operations that obviously pose problems of infinity with binary relations, and add finiteness-preserving operations and properties that produce sufficient information even when applied only to finite arguments. Since the common subtype to the original theories are categories, and, as mentioned above, we do not in general have identities in our setting, the appropriate algebraic structure to form the basis of our investigations is that of semi-groupoids (see for example [42,43], and Definition 2.1 below), which are related to categories in the same way as semigroups are related to monoids, that is, identities are omitted.\footnote{One might prefer to use the name “semicategory”, but this is already used for “categories with identities, but with only partial composition” [39], and the name “semi-groupoid” seems to be well-established in the mathematical literature.}

To convey a first flavour of how to we approach this problem, and to introduce a pattern recurring a few times in the body of the paper, we now quickly present residuation in general (Section 1.1), and as first instances on the one hand complement concepts for finite sets (Section 1.2), and on the other hand residuals of composition for binary relations (Section 1.3). In Section 2, we present the definitions of basic semigroupoids, including those with converse and with inclusion ordering, and the important concept of sub-identity. Section 3, in a certain sense, revolves around various applications of sub-identities, in particular to the concepts of domain and determinacy, and shows how allegories almost “do not need identities”. We then round of the morphism-centric theories by generalising Kleene algebras in Section 4, and residuals of composition in Section 5.

Moving, in a sense, to objects means to consider datatype construction—the challenge here is that we want a library to be able to provide an interface to the fact that its infinite-precision natural number type is the subobject of its infinite-precision integer type “containing” exactly the non-negative integers, but the interface to do this can only expect finite relations as arguments, and only produce finite relations as result, and still should satisfy a set of algebraic laws that uniquely determine the desired subobject. We discuss this particular example in much detail in Section 6, together with the solutions we propose for subobjects and for quotients. In Section 7, we then gear the mechanisms acquired in the previous section up to cope with the more complex situation of direct products and projections.

Finally, we discuss the design of our Haskell implementation in Section 8.

1.1. Residuation

Given an ordering \(\leq\) and a binary operator \(\otimes\), an element \(R\) is called right-residual of \(S\) with respect to \(Q\) iff for all \(X\) (of appropriate type),

\[
Q \otimes X \leq S \iff X \leq R
\]
Analogously, \( Q \) is called \textit{left-residual of} \( S \) \textit{with respect to} \( R \) \textit{iff} for all \( X \) (of appropriate type),
\[
X \otimes R \leq S \iff X \leq Q
\]

For commutative operators, one just talks about “residuals”.

Where a certain kind of residual always exists, this gives rise to a Galois connection; this fact makes many useful laws available, see for example \([1,37]\).

As an example, \textit{relative pseudo-complement} (implication) \( R \Rightarrow S \) in a lattice is the residual of meet (conjunction) with respect to the lattice ordering:
\[
(X \land R) \leq S \iff X \leq (R \Rightarrow S)
\]

Residuals of continuous operations exist automatically if the relevant ordering is complete. If this is not the case, they are frequently introduced axiomatically as existing for certain operators.

The plain names “left-residual” and “right-residual” are used for composition, see below in Section 1.3.

\subsection*{1.2. Difference of finite sets}

Finite subsets of an infinite set do \textit{not} form a Boolean algebra: there is no largest finite set, and therefore the concept of complement does not make sense.

Only relative complements exist—the difference function normally available in libraries calculates \( A - B = A \cap \overline{B} \), which can be defined in several ways without referring to complement:

1. The \textit{complement} of \( B \) in the sublattice of finite sets contained in \( A \): \textit{the} finite set \( X \subseteq A \) for which \( X \cup B = A \) and \( X \cap B = \emptyset \) hold.
2. The \textit{pseudo-complement} of \( B \) in the sublattice of finite sets contained in \( A \): \textit{the largest} finite set \( X \subseteq A \) for which \( X \cap B = \emptyset \) hold.
   (This is the \textit{relative pseudo-complement} \( B \Rightarrow \emptyset \) of \( B \) with respect to \( \emptyset \), taken in the sublattice below \( A \).)
3. The \textit{relative semi-complement} (dual concept to pseudo-complement) of \( B \) with respect to \( A \): \textit{the smallest} finite set \( X \) for which \( A \subseteq X \cup B \) holds.

Algebraists are more used to the first two definitions—this corresponds to the habit of defining a complete lattice as a lattice in which all joins (least upper bounds) exist.

However, the last definition is conceptually simpler, since it only involves \( A \) and \( B \), and does not need a restriction to a sub-space.

Also note that, in the Boolean lattice of all subsets of some infinite set, unrestricted pseudo-complement i.e., Boolean implication, does not preserve finiteness, since union preserves infiniteness:
\[
B \Rightarrow C = \overline{B} \cup C
\]

However, restricted implication can easily be expressed using semi-complement, i.e., difference:
\[
A \cap (B \Rightarrow C) = A \cap (\overline{B} \cup C) = A \cap \overline{B} \cap \overline{C} = A - (B - C)
\]

The problem of missing universal and complement sets of course immediately applies to sets of pairs, i.e., relations, too.

\subsection*{1.3. Residuals of composition for relations}

Turning to relations, we use the notation \( \mathcal{A} \leftrightarrow \mathcal{B} := \mathcal{P}(\mathcal{A} \times \mathcal{B}) \) for the set of \textit{all} (concrete) relations from set \( \mathcal{A} \) to set \( \mathcal{B} \), and \( \mathcal{A} \leftrightarrow \mathcal{B} \) for the set of all \textit{finite} relations from \( \mathcal{A} \) to \( \mathcal{B} \). (We use the term “concrete relation” to refer to a subset of a binary Cartesian product of sets, in contrast with the “abstract” use of the word “relation” for a morphism of some heterogeneous relation algebra or indeed also of some category or semigroupoid that is perceived to generalise properties of relations.)
A (heterogeneous) relation algebra, which is a category where every homset is a Boolean algebra and some additional laws hold [38], can be obtained from any collection of sets by taking these sets as objects and arbitrary relations between them as morphisms.

If we move to finite relations, then we again loose complements, largest elements, and identities, so where infinite sets are involved, we have no category of finite relations, only a semigroupoid.

Finiteness is also not preserved by the residuals of composition; this is obvious from the complement expressions that calculate these residuals in full relation algebras (shown in the right-most column):

\[
\begin{align*}
X \subseteq (Q \setminus S) & \iff Q ; X \subseteq S \hspace{1cm} \text{right-residual} \hspace{1cm} Q \setminus S = \overline{Q \cdot \overline{S}} \\
X \subseteq (S/R) & \iff X ; R \subseteq S \hspace{1cm} \text{left-residual} \hspace{1cm} S/R = \overline{S \cdot R} \\
\end{align*}
\]

Residuals are important since they provide the standard means to translate predicate logic formulae involving universal quantification into complement-free relational formalisms:

\[
\begin{align*}
(x, y) \in S/R & \iff \forall z. (y, z) \in R \Rightarrow (x, z) \in S \\
(y, z) \in Q \setminus S & \iff \forall x. (x, y) \in Q \Rightarrow (x, z) \in S \\
\end{align*}
\]

Altogether, we have so far identified three problems with relation-algebraic treatment of finite relations:

- No complement (negation): This is usually not a real problem, but perhaps only somewhat of a nuisance that can be worked around by using difference.
- No identities: Many relational properties are normally defined using identities, so their absence will make itself felt a bit more significantly.
- No residuals: To the newcomer to relational algebra, residuals may appear to be a rather strange construction; however, the fact that they are the tool to translate universal quantifications into relation-algebraic formulae frequently makes them indispensable for point-free formulations.

In the next sections, we show how most relational formalisations can be adapted into a generalised framework that avoids these problems.

2. Basic semigroupoids

Above, we showed how set difference can be defined in a point-free way, without reference to the complement operation available only in the superstructure of arbitrary sets, only in terms of the theory of finite sets. This way of directly defining concepts that are better-known as derived in more general settings has the advantage that it guarantees a certain “inherent conceptual coherence”.

In order to achieve this coherence also for the interface to our finite relation library, we now take a step back from concrete finite relations and consider instead a hierarchy of semigroupoid theories geared towards relational concepts in a similar way as Freyd and Scedrov’s hierarchy of allegories [18] does this for category theory. Our exposition will, however, be structured more as a generalisation of the theory organisation of [26] from categories to semigroupoids. Proofs that carry over from the categorical case without significant change will be omitted without special mention.

2.1. Semigroupoids, identities, and categories

Semigroupoids are to categories as semigroups are to monoids: no identities are assumed:

**Definition 2.1.** A **semigroupoid** $(\text{Obj}, \text{Mor}, \text{src}, \text{trg}, \cdot)$ is a graph with a set $\text{Obj}$ of objects as vertices, a set $\text{Mor}$ of morphisms as edges, with $\text{src}, \text{trg} : \text{Mor} \to \text{Obj}$ assigning source and target object to each morphism (we write
“\( f : A \to B \)” instead of “\( f \in \text{Mor} \) and \( \text{src} \ f = A \) and \( \text{trg} \ f = B \)”), and an additional partial operation “\( \_ \land \_ \)” of composition such that the following hold:

- For \( f : A \to B \) and \( g : B' \to C \), the composition \( f : g \) is defined iff \( B = B' \), and if it is defined, then \( (f : g) : A \to C \).
- Composition is associative, i.e., if one of \( (f : g) : h \) and \( f : (g : h) \) is defined, then so is the other and they are equal.

For two objects \( A \) and \( B \), the collections of morphisms \( f : A \to B \) is also called the homset from \( A \) to \( B \), and written \( \text{Hom}(A, B) \).

A morphism is called an endomorphism iff its source and target objects coincide; an endomorphism \( R \) is called idempotent if \( R : R = R \).

In typed relation algebras, endomorphisms are often called homogeneous relations.

**Definition 2.2.** In a semigroupoid, an endomorphism \( I : A \to A \) is called an identity on \( A \) (or just an identity) iff it is a local left- and right-identity for composition, i.e., iff for all objects \( B \) and for all morphisms \( F : A \to B \) and \( G : B \to A \) we have \( I : F = F \) and \( G : I = G \).

If there is an identity on \( A \), we write it \( \mathbb{1}_A \).

A category is a semigroupoid where each object has an identity.

As in semigroups, identities are unique where they exist, and whenever we write \( \mathbb{1}_A \) without further comment, we imply the assumption that it exists.

Self-duality is an important property in the context of relations:

**Definition 2.3.** An semigroupoid with converse (SGC), also called a self-dual semigroupoid, is a semigroupoid where each morphism \( R : A \to B \) has a converse \( R^\circ : B \to A \), and for all \( R : A \to B \) and \( S : B \to C \), the involution equations \( (R^\circ)^\circ = R \) and \( (R : S)^\circ = S^\circ : R^\circ \) hold.

### 2.2. Ordered semigroupoids

A hallmark of the relational flavour of reasoning is the use of inclusion; the abstract variant of this directly corresponds to the categorical version:

**Definition 2.4.** A locally ordered semigroupoid, or just ordered semigroupoid, is a semigroupoid in which on each homset \( \text{Hom}(A, B) \), there is an ordering \( \preceq_{A,B} \), and composition is monotonic in both arguments.

We will normally omit the subscripts, as they can be deduced from the context, and we will write “\( \sqsubseteq \)” for the strict-order associated with \( \preceq \), that is, “\( R \sqsubseteq S \)” means “\( R \sqsubseteq S \land R \neq S \)”.

Some familiar concepts are available unchanged:

**Definition 2.5.** In an ordered semigroupoid, a morphism \( R : A \to A \) is called transitive iff \( R : R \sqsubseteq R \), and co-transitive iff \( R \sqsubseteq R : R \).

If the homset \( \text{Hom}(A, B) \) has a greatest element, then this will be denoted \( \sqcup_{A,B} \), and if it has a least element, this will be denoted \( \sqcap_{A,B} \). Existence of least morphisms is usually assumed together with the zero law:

**Definition 2.6.** An ordered semigroupoid with zero morphisms is an ordered semigroupoid such that each homset \( \text{Hom}(A, B) \) has a least element \( \sqcap_{A,B} \), and each least element \( \sqcap_{A,B} \) is a left- and right-zero for composition.

Frequently, in the literature, the inclusion ordering \( \subseteq \) is not primitive, but defined using meet, for example in the allegory definition of Freyd and Scedrov [18]. In such contexts, meet-subdistributivity of composition,

\[
Q : (R \sqcap S) \subseteq Q : R \sqcap Q : S
\]

is usually listed as an axiom, whereas here it follows from monotonicity of composition.
Definition 2.7. A lower semilattice semigroupoid is an ordered semigroupoid such that each homset is a lower semilattice with binary meet \( \sqcap \).

By demanding strict distributivity of composition over join, upper semilattice semigroupoids are not completely dual to lower semilattice semigroupoids:

Definition 2.8. An upper semilattice semigroupoid is an ordered semigroupoid such that each homset is an upper semilattice with binary join \( \sqcup \), and composition distributes over joins from both sides.

An upper semilattice semigroupoid is called bounded-complete if it has arbitrary joins \( \sqcup S \) over bounded\(^2\) subsets \( S \) of a single homset, and composition distributes over these joins from both sides.

A complete upper semilattice semigroupoid has joins \( \sqcup S \) over arbitrary subsets \( S \) of a single homset, and composition distributing over them from both sides.

For finite relations between infinite sets, we obviously have bounded completeness, but not completeness.

If we consider upper or lower semilattice semigroupoids with converse, i.e., upper respectively lower semilattice semigroupoids that are at the same time OSGCs, then the involution law for join respectively meet follows from isotony of converse.

2.3. Subidentities

The usual definitions of reflexivity and co-reflexivity, however, involve an identity; we work around this by essentially defining the concept “included in an identity” without actually referring to identity morphisms:

Definition 2.9. In an ordered semigroupoid, let an endomorphism \( p : \mathcal{A} \rightarrow \mathcal{A} \) be given. If for all objects \( \mathcal{B} \) and all morphisms \( R : \mathcal{A} \rightarrow \mathcal{B} \) and \( S : \mathcal{B} \rightarrow \mathcal{A} \) we have

- \( p \cdot R \sqsubseteq R \) and \( S \cdot p \sqsubseteq S \), then \( p \) is called a subidentity, and if we have
- \( R \sqsubseteq p \cdot R \) and \( S \sqsubseteq S \cdot p \), then \( p \) is called a superidentity.

Obviously, for each object \( \mathcal{A} \), each subidentity \( p \) on \( \mathcal{A} \) is contained in each superidentity \( Q \) on \( \mathcal{A} \):

\[
p \sqsubseteq p : Q \quad Q \text{ is superidentity}
\]

\[
\sqsubseteq Q \quad p \text{ is subidentity}
\]

Furthermore, Definition 2.9 directly implies, due to antisymmetry of \( \sqsubseteq \), that if \( p \) is both a superidentity and a subidentity, then \( p \) is an identity.

In ordered categories (or monoids), subidentities are normally defined as elements included in identities, see e.g. [16]. If the identity morphism \( \mathbb{1}_\mathcal{A} \) exists, then each subidentity \( p : \mathcal{A} \rightarrow \mathcal{A} \) is indeed contained in the identity, since \( p = p : \mathbb{1}_\mathcal{A} \sqsubseteq \mathbb{1}_\mathcal{A} \). Therefore, we also call subidentities co-reflexive, and, dually, we call superidentities reflexive.

2.4. Ordered semigroupoids with converse (OSGCs)

We now consider ordered semigroupoids with converse without assuming the availability of join or meet, and we shall see that the resulting theory is quite expressive.

Definition 2.10. An ordered semigroupoid with converse (OSGC) is an ordered semigroupoid that is also self-dual, and where conversion is monotonic with respect to \( \sqsubseteq \).

Because of involution, conversion is in fact isotonic with respect to \( \sqsubseteq \).

\(^2\) A subset \( S \) of some set \( X \) is called bounded iff there is an element \( x \in X \) which is an upper bound for \( S \), i.e., for all elements \( s \in S \) we have \( s \sqsubseteq x \).
Lemma 2.11. Conversion preserves subidentities and superidentities.

Proof. For any relation $p$ we obtain:

$$p^{-1} \circ R \sqsubseteq R \iff (p^{-1} \circ R)^{-1} \subseteq R \iff R^{-1} \circ p \sqsubseteq R^{-1},$$

and analogously $S \circ p^{-1} \sqsubseteq S$ iff $p \circ S^{-1} \subseteq S^{-1}$, so if $p$ is a subidentity, then $p^{-1}$ is one, too. Preservation of superidentities is shown in the same way. □

Many standard properties of relations can be characterised in the context of OSGCs—not significantly hindered by the absence of identities. Those relying on superidentities are, of course, only of limited use in semigroupoids of finite relations between potentially infinite sets.

Definition 2.12. For a morphism $R : A \to B$ in an OSGC we define:

- $R$ is univalent iff $R^{-1} \circ R$ is a subidentity.
- $R$ is injective iff $R \circ R^{-1}$ is a subidentity.
- $R$ is difunctional iff $R \circ R^{-1} \subseteq R$.
- $R$ is co-difunctional iff $R \subseteq R \circ R^{-1}$.
- $R$ is strictly difunctional iff $R \circ R^{-1} = R$.
- $R$ is total iff $R^{-1} \circ R$ is a superidentity.
- $R$ is surjective iff $R^{-1} \circ R$ is a superidentity.
- $R$ is a mapping iff $R$ is univalent and total.
- $R$ is bijective iff $R$ is injective and surjective.

If $R$ is univalent or injective, then $R$ is obviously difunctional.

Subidentities are univalent and injective, while superidentities are total and surjective.

All concrete relations, including all finite relations, are co-difunctional.

For endomorphisms, there are a few additional properties of interest:

Definition 2.13. For a morphism $R : A \to A$ in an OSGC we define:

- $R$ is symmetric iff $R^{-1} \subseteq R$.
- $R$ is a co-equivalence iff $R$ is co-reflexive, co-transitive, and symmetric.
- $R$ is a partial equivalence iff $R$ is symmetric and idempotent.
- $R$ is an equivalence iff $R$ is reflexive, transitive, and symmetric.

Symmetry is self-dual, therefore, the concept of co-equivalence is the strict $\sqsubseteq$-dual of the concept of equivalence.

In the categorical context, a number of connections between the properties introduced above has been shown in [26, Section 3.4]; is easy to see these all carry over directly to the semigroupoid-based definitions presented here.

3. Allegory topics

Allegories were originally introduced by Freyd and Scedrov [18] to be “to binary relations between sets as categories are to functions between sets”. Their axiomatisation adds meet and converse to the signature of categories, and appropriate laws.

Since Freyd and Scedrov did not consider any locally ordered categories with converse weaker than allegories, allegories became the natural home for the definition of univalence listed in Definition 2.12. Allegories are, in that context, also the natural home for domain, which can be defined there using identities, meet, and converse:

$$\text{dom } R := \mathbb{I} \cap R \circ R^{-1}.$$

The predomain and domain axioms of Definition 3.1 then all become theorems.

Recently, Desharnais, Möller and Struth performed in-depth explorations of both determinacy [15] and domain [16] in the context of Kleene algebras (with different extensions).
It turns out that much of this material can already be introduced in weaker structures; we discuss domain in the setting of ordered semigroupoids in Section 3.1; then we use domain in our definition of semi-allegories in Section 3.2, and finally move between the different settings to discuss different determinacy concepts in Section 3.4.

3.1. Domain

Related to the introduction of “Kleene algebras with tests” [29], which allow the study of pre- and postconditions in a Kleene algebra setting, domain (and range) operators have been studied in the Kleene algebra setting [34,16]. Much of the material there can be transferred into our much weaker setting of ordered semigroupoids by replacing preservation of joins with monotonicity and using our subidentity concept of Definition 2.9 instead of the special set of “test” endomorphisms used in [16]. The definition of “predomain” is given as a special residual of composition with respect to the ordering $\sqsupseteq$:

**Definition 3.1.** An ordered semigroupoid with predomain is an ordered semigroupoid where for every $R : A \to B$ there is a subidentity $\text{dom} \ R : A \to A$ such that for every subidentity $q : A \to A$, we have $q \cdot R \sqsupseteq R$ iff $q \sqsupseteq \text{dom} \ R$.

In an ordered semigroupoid with domain, additionally the locality condition $\text{dom} \ (R ; S) = \text{dom} \ (R \cdot \text{dom} \ S)$ has to hold.

The following properties carry easily over to the semigroupoid setting from the semiring setting of [16]:

**Lemma 3.2.** Choosing morphisms $R$ and $S$ and subidentities $p$ in each item from an ordered semigroupoid with predomain so that all expressions are defined, we have

1. monotonicity: $R \subseteq S \Rightarrow \text{dom} \ R \subseteq \text{dom} \ S$
2. identity on subidentities: $\text{dom} \ p = p$
3. $(\text{dom} \ R) \cdot R = R$
4. import/export law: $\text{dom} \ (p \cdot R) = p \cdot \text{dom} \ R$
5. sublocality: $\text{dom} \ (R ; S) \subseteq \text{dom} \ (R \cdot \text{dom} \ S)$

In addition, we obtain the existence of left-identities from the existence of superidentities:

**Lemma 3.3.** In an ordered semigroupoid with predomain, if $Q$ is a superidentity on $A$, then $\text{dom} \ Q$ is a left-identity on $A$.

**Proof.** For any $R : A \to B$, its domain $\text{dom} \ R$ is a subidentity; since $Q$ is a superidentity, we therefore have $\text{dom} \ R \subseteq Q$. With idempotence and monotonicity of predomain (Lemma 3.2), we obtain

$\text{dom} \ R = \text{dom} \ (\text{dom} \ R) \subseteq \text{dom} \ Q$

With the definition of predomain, we then have:

$R \subseteq (\text{dom} \ R) \cdot R \subseteq (\text{dom} \ Q) \cdot R$

Since $\text{dom} \ Q$ is a subidentity by definition, we have equality:

$(\text{dom} \ Q) \cdot R = R \quad \square$

Range can be defined analogously to domain; an OSGC with domain also has range, and range is then related with domain via converse: $\text{ran} \ R = (\text{dom} \ (R^*))$ — note that subidentities are not necessarily symmetric.

---

3 It is important not to confuse these domain and range operations, which only make sense in ordered semigroupoids, with the semigroupoid (or categorical) concepts of source and target of a morphism!
Domain and range allow us to define “rectangles”, which will be useful later:

**Definition 3.4.** If, in an ordered semigroupoid with predomain and prerange, for any two subidentities \( p : A \to A \) and \( q : B \to B \) there is a largest morphism \( T : A \to B \) such that \( \text{dom} \, T \subseteq p \) and \( \text{ran} \, T \subseteq q \), then we call \( T \) a **restricted top morphism** and denote it by \( \top_{p,q} \).

The existence of restricted top morphisms does not follow from bounded completeness, but is useful in its context. If one of \( p \) and \( q \) is a zero morphism, then \( \top_{p,q} \) is a zero-morphism, too, so in general we cannot expect \( \text{dom} \, \top_{p,q} = p \), in the same way as we cannot expect \( \text{dom} \, \top_{A,B} = I_A \) for an unrestricted top morphism \( \top_{A,B} \). (For concrete relations, this fails when \( B \) is the empty set and \( A \) is non-empty.) However, \( \text{dom} \) does distribute over join:

**Lemma 3.5.** In an upper semilattice semigroupoid with predomain, \( \text{dom} \) distributes over joins.

**Proof.** From monotonicity of \( \text{dom} \) and the definition of join we obtain \( \text{dom} \, R \sqcup \text{dom} \, S \subseteq \text{dom} \, (R \sqcup S) \).

The converse inclusion follows from the predomain property with

\[
\begin{align*}
(\text{dom} \, R \sqcup \text{dom} \, S) : (R \sqcup S) &= (\text{dom} \, R) : R \sqcup (\text{dom} \, S) : R \sqcup (\text{dom} \, S) : S \sqcup (\text{dom} \, R) : S \\
&= R \sqcup S.
\end{align*}
\]

This result is trivial for the domain definition of Backhouse et al. [1], since it employs a Galois connection; however, that Galois connection requires the presence of greatest morphisms, and can therefore not be used in our setting.

### 3.2. Semi-allegories

In the allegories of Freyd and Scedrov [18], domain in the sense of Definition 3.1 can be obtained from identities, meet, and converse: \( \text{dom} \, R = I \cap R : R' \) (see also Lemma 3.8 below). Nevertheless, Dougherty and Gutiérrez [17] proposed to add the domain operator to the signature of allegories, in part motivated by the fact that domain corresponds to a very basic operation in their graphical calculus for allegory equations. From the results below, we will see that including predomain provides the appropriate starting point for defining “allegories without identities”:

**Definition 3.6.** A **semi-allegory** is a lower semilattice semigroupoid with converse and predomain such that for all \( Q : A \to B, R : B \to C, \) and \( S : A \to C, \) the **Dedekind rule** holds:

\[
Q : R \cap S \subseteq (Q \cap S : R') : (R \cap Q' : S)
\]

As in allegories, the Dedekind rule is equivalent to either of the **modal rules** occasionally used in the axiomatisation in its place:

\[
\begin{align*}
Q : R \cap S \subseteq Q : (R \cap Q' : S) \\
Q : R \cap S \subseteq (Q \cap S : R') : R
\end{align*}
\]

Examples of concrete semi-allegories can be produced from any algebraic signature \( \Sigma \) (with function symbols of arbitrary finite arity), by taking \( \Sigma \)-algebras as objects and **finite relational \( \Sigma \)-algebra homomorphisms** (see e.g. [25]) as morphisms.

In a semi-allegory we can define \( R : A \to A \) to be **antisymmetric** iff \( R' \cap R \) is a subidentity.

One contributing factor to our inclusion of domain is that, in the absence of identities, we also need domain besides the Dedekind formula to be able to show that all morphisms are co-difunctional:

\[
R = \text{dom} \, R : R \cap R \subseteq (\text{dom} \, R \cap R : R') : R \subseteq R : R' : R.
\]

Co-difunctionality of all morphisms in semi-allegories implies symmetry of sub-identities:

\[
p \subseteq p ; p' ; p \subseteq p'.
\]

Symmetry then further implies idempotence of sub-identities; with Lemma 3.3 we also obtain that for each object \( A \), if there is a superidentity \( Q \) on \( A \), then the identity on \( I_A \) exists and is given by \( \text{dom} \, Q \).
With respect to subidentities, the following laws hold:

Lemma 3.7. In a semi-allegory, the following hold for morphisms \( Q : \mathcal{B} \to \mathcal{A} \), \( R, S : \mathcal{A} \to \mathcal{B} \), and subidentities \( p \) on \( \mathcal{B} \):

1. \( R \cap S : p = (R \cap S) : p \)
2. \( p \cap Q : (R \cap S) = p \cap (Q \cap R) : S \)
3. \( Q \cap Q : S \cap S = Q : (\text{ran } Q \cap S : S) \)

Proof

1. \( R \cap S : p \leq (R : p^\cap \cap S : p) \), modal rule

2. \( (R \cap S) : p \leq R \cap S : p \), \( p^\cap \) is subidentity

3. \( p \cap Q : (R \cap S) = p \cap (Q \cap R) : S \), modal rule

The opposite inclusion is shown analogously.

The opposite inclusion follows from meet-subdistributivity. \( \square \)

The domain formula of allegories can be regained from the following lemma by setting \( q := \mathbb{I} \); another important instance is obtained via \( q := \text{dom } R \).

Lemma 3.8. In a semi-allegory, given a morphism \( R : \mathcal{A} \to \mathcal{B} \) and a subidentity \( q \) on \( \mathcal{A} \) with \( q \sqsubseteq \text{dom } R \), then \( \text{dom } R = q \cap R : R \).

Proof. Since

\[
R = R \cap q : R
\]

\[

\]

we have \( q \cap R : R^\cap \sqsubseteq \text{dom } R \) according to Definition 3.1.

Furthermore, if \( p \) is a subidentity on \( \mathcal{A} \) with \( R \leq p : R \), then

\[
q \cap R : R^\cap \leq q \cap p : R : R^\cap
\]

\[

\]

Lemma 3.7.(1))

\[

\]

\[

\]

We only included predomain in the definition of semi-allegories since locality follows (indirectly) from the Dedekind rule.

Lemma 3.9. Semi-allegories have locality of domain.

Proof. Because of sublocality of predomain, we only need to show the following:

\[
\text{dom } (R : S) \equiv \text{dom } (R : \text{dom } S).
\]

Let \( p \) be a sub-identity on \( \text{src } R \), then the definition of predomain yields:

\[
p \equiv \text{dom } (R : S) \iff p : R : S \equiv R : S
\]

\[
p \equiv \text{dom } (R : \text{dom } S) \iff p : R : \text{dom } S \equiv R : \text{dom } S
\]
We show the latter using the former:

\[ R \cap \text{dom } S = R \cap (\text{dom } S \cap S^\perp) \]

Lemma 3.8

\[ \subseteq R \cap \text{dom } S \cap R \cap S^\perp \]

\[ \subseteq p \cap R \cap S \cap S^\perp \]

Lemma 3.7.(1)

\[ \subseteq p \cap R \cap \text{dom } S \]

3.3. Allegories

Allegories have been introduced by Freyd and Scedrov [18] essentially as lower semilattice categories with converse satisfying the modal rules. The relationship between allegories and semi-allegories is straightforward:

**Lemma 3.10.** Each allegory is a semi-allegory, with \( \text{dom } R = I \cap R \cap R^\perp \).

**Proof.** This definition of domain and its well-definedness as domain operator follow from Lemma 3.8.

Since the identity laws are the only allegory laws involving identities, we also have:

**Lemma 3.11.** A semi-allegory where all objects have identities is an allegory.

Since, with a modal rule, \( I \cap R = I \cap R \cap I \subseteq R \cap (I \cap R^\perp \cap I) \subseteq R \cap R^\perp \), we have

\[ I \cap R = I \cap R \cap R \cap R = \text{dom } R \cap R. \]

Furthermore, identities are simply reproduced by converse or domain, and disappear in compositions due to the identity laws, so identities can always be eliminated from allegory expressions except from those equal to an identity.

Therefore, when starting with a semi-allegory, formally adjoining an identity to all those objects that do not have one produces an allegory with no further additional elements.

This sheds some new light on Gutiérrez’ decision procedure for allegory equations, and in particular on the axiomatisation including domain [17]. Although identities appear to feature rather prominently there, too, closer inspection reveals that the many occurrences of intersections with identities correspond directly to our use of subidentities, for example in Lemma 3.7. Since the transformation rules there only identify (non-identity-)edges from their graph representation of allegory expressions, but never delete edges, identities, which correspond to edge-free one-node graphs, play no rôle at all in the equality decision procedure, since no other graphs rewrite to identities.

So one could state that Dougherty and Gutiérrez’ decision procedure is really a decision procedure for semi-allegory equations, which has been trivially extended to allegory equations by allowing edge-free graphs.

3.4. Determinacy

In Definition 2.12, the definition of univalence is essentially that given normally in an allegory context. It therefore does not apply in a Kleene algebra context—even in Kleene algebras with converse, it is not a useful concept, and does not map to the natural concept of determinacy in standard models.

Desharnais and Möller [15] studied different candidates for determinacy concepts in (mostly Boolean) Kleene algebras. However, except for the candidates directly involving complement (which are of minor importance anyway), they all can be presented or reformulated in weaker settings.

**Definition 3.12.** In a lower semilattice semigroupoid, a morphism \( F : A \to B \) is left-distributive iff for all objects \( C \) and for all morphisms \( R, S : B \to C \), we have

\[ F : (R \cap S) = F : R \cap F : S \]
The important part is the inclusion “⊒”, since “⊑” follows from monotonicity of composition and the definition of meet.

In semi-allegories, univalence implies left-distributivity:

\[ (R \sqcap S) \sqcap F = (R \sqcap (S \sqcap F)) \]

However, right factoring is directly equivalent with univalence: For “⇒”, due to self-duality, we need to show only one of the subidentity properties; we choose \( R \sqcap F \sqcap S = (R \sqcap R \sqcap F) \sqcap F \)

And univalence implies right factoring:

\[ (R \sqcap S) \sqcap F = (R \sqcap (S \sqcap F)) \sqcap F \]

The second major determinacy concept defined by Desharnais and Möller manages to leave meet behind, at the cost of using domain:

**Definition 3.13.** In a semi-allegory, a morphism \( F : B \to C \) is called right factoring iff for all morphisms \( R : A \to C \) and \( S : A \to B \),

\[ R \sqcap S \sqcap F = (R \sqcap R \sqcap S) \sqcap F. \]

However, right factoring is directly equivalent with univalence: For “⇒”, due to self-duality, we need to show only one of the subidentity properties; we choose \( R \sqcap F \sqcap R \sqcap F = (R \sqcap R \sqcap R \sqcap F) \sqcap F \)

And univalence implies right factoring:

\[ (R \sqcap S) \sqcap F = (R \sqcap (S \sqcap F)) \sqcap F \]

**Definition 3.14.** In an ordered semigroupoid with predomain, a morphism \( F : A \to B \) is domain-minimal iff

\[ \forall R : A \to B \bullet R \sqsubseteq F \Rightarrow \text{dom} R \sqsubseteq \text{dom} F. \]

The equivalent formulations listed in [15] include:

**Lemma 3.15.** In an ordered semigroupoid with predomain, domain-minimality of a morphism \( F : A \to B \) is equivalent to:

\[ \forall R : A \to B \bullet R \sqsubseteq F \Rightarrow R = \text{dom} R \sqsubseteq F. \]

Desharnais and Möller show compositionality of domain-minimality for atomic Boolean Kleene algebras [15]; here, we not only want to avoid complement, but we also prefer to be able to work in non-atomic lattices, which arise naturally in the context of graphs (see [25]).

As appropriate generalisation of atomic, an element \( A \) of an upper semilattice is called join-indecomposable if, whenever \( A = B \sqcup C \), then \( A = B \) or \( A = C \).

**Lemma 3.16.** In an upper semilattice semigroupoid with predomain (respectively pre-range), if a morphism \( F \) is join-indecomposable, then \( \text{dom} F \) (respectively \( \text{ran} F \)) is join-indecomposable, too.

**Proof.** If \( \text{dom} F = p \sqcup q \), then \( p \) and \( q \) are both sub-identities, and:

\[ F = \text{dom} F \sqsubseteq F = (p \sqcup q) \sqsubseteq F = p \sqcup q \sqsubseteq F \]

If \( F \) is join-indecomposable, then, without restriction of generality, \( p : F = F \), which together with \( \text{dom} F = p \sqcup q \) implies \( p = \text{dom} F \), so \( \text{dom} F \) is join-indecomposable, too.

The proof for \( \text{ran} \) is perfectly dual. □
To abbreviate the following arguments, we write $J(A)$ for the set of all join-indecomposable elements $B$ with $B \sqsubseteq A$. We call an upper semilattice decomposable if it is bounded complete and we have $A = \bigsqcup J(A)$ for all elements $A$. (All bounded-complete lattices with well-founded ordering are decomposable.)

Lemma 3.17. If the join-indecomposable morphisms in a decomposable upper semilattice semigroupoid with domain and range are closed under composition, and also are all domain-minimal, then:

1. $F$ is domain-minimal iff for all $p \in J(\text{dom } F)$, the composition $p \cdot F$ is join-indecomposable again.
2. Domain-minimality is closed under composition.

Proof

(1) “⇒”: Assume that $F$ is domain-minimal and $p \sqsubseteq \text{dom } F$ is a join-indecomposable subidentity. Assume further that $p \cdot F = R \sqcup S$.

From Lemma 3.5 and the fact that $p$ is a subidentity we know that $\text{dom } R \sqcup \text{dom } S = \text{dom } (R \sqcup S) = \text{dom } p \cdot F = p \cdot \text{dom } F = p$.

Since $p$ is join-indecomposable, we can assume without restriction of generality that $\text{dom } R = p$.

Since $p \cdot F \sqsubseteq F$, we also have $R \sqsubseteq F$, and with Lemma 3.15 we obtain: $R = \text{dom } R \cdot F R = p \cdot F$, so $p \cdot F$ is join-indecomposable, too.

“⇐”: Assume $R \sqsubseteq F$, then, if $p \in J(\text{dom } R)$, then $p \cdot R \sqsubseteq p \cdot F$, and $p \cdot F$ is join-indecomposable, and therefore domain-minimal, so we have according to Lemma 3.15:

$p \cdot R = \text{dom } (p \cdot R) : p \cdot F = p \cdot \text{dom } R : p \cdot F = p : p \cdot F = p \cdot F$

From this, we obtain domain-minimality of $F$ by showing the property of Lemma 3.15:

$$\text{dom } R : F = (\bigsqcup J(\text{dom } R)) : F$$
$$= \bigsqcup \{ p : J(\text{dom } R) \cdot p : F \}$$
$$= \bigsqcup \{ p : J(\text{dom } R) \cdot p : R \}$$
$$= (\bigsqcup J(\text{dom } R)) : R$$
$$= \text{dom } R : R$$
$$= R$$

(2) Assume that $F$ and $G$ are domain-minimal, and that $p \in J(\text{dom } (F \cdot G))$.

We use (1) to show that $F : G$ is domain-minimal, so we need to show that $p : F : G$ is join-indecomposable. Since $p \in J(\text{dom } F)$, we know from (1) that $p : F$ is join-indecomposable.

Then Lemma 3.16 implies that $\text{ran } (p : F)$ is join-indecomposable, too, and, since locality of domain and domain-minimality of $F$ via (1) imply

$$\text{dom } (F : G) : F = \text{dom } (F : \text{dom } G) : F = F : \text{dom } G,$$

we have $\text{ran } (p : F) \sqsubseteq \text{dom } G$, and then we obtain, again using (1), that also $\text{ran } (p : F) : G$ is join-indecomposable. Since $p : F : G = (p : F) : (\text{ran } (p : F) : G)$ is the composition of two join-indecomposable morphisms, it is, according to the assumption, join-indecomposable too. □

The proof of (2) followed the same path as the corresponding proof in [14], but the preparations differed due to the slightly different flavour of reasoning with join-indecomposability rather than with atomicity, which made us use exclusively Lemma 3.15, instead of using Definition 3.14 directly.

Although our statement of compositionality of domain-minimality is also much more involved, it has the atomic version of [14] as an immediate corollary, but encompasses for example also the relational graph homomorphisms of [25], where graph edges give rise to join-indecomposable elements that are not atoms.
The third important determinacy concept given by Desharnais and Möller is that \( F : A \rightarrow B \) is *modally deterministic* iff
\[
\langle F \rangle t \sqsubseteq [F]t \quad \text{for all subidentities} \ t \text{ on } B.
\]
This definition uses the modalities diamond, defined for relations \( R : A \rightarrow B \) and subidentities \( t \) on \( B \) as
\[
\langle R \rangle t := \text{dom} (R; t),
\]
and box \([R]t\), where box is defined in [15] using complement in the Boolean lattice of tests. Backhouse and van der Woude defined this operation as “monotype factor” using a Galois connection [3]:
\[
u \sqsubseteq [R]t \iff \text{ran} (u \circ R) \subseteq t \quad \text{for all subidentities} \ u \text{ on } B.
\]
This allows us to reformulate the condition without using modalities or complement:

**Definition 3.18.** In an ordered semigroupoid with domain and range, a morphism \( F : A \rightarrow B \) is *modally deterministic* iff for all subidentities \( t \) on \( B \) we have
\[
\text{ran} \left( \text{dom} \left( (F : t) \circ F \right) \right) \subseteq t.
\]

Using locality of domain and range, it is now easy to show that the composition of two modally deterministic morphisms is again modally deterministic:
\[
\text{ran} \left( \text{dom} \left( (F : G : t) \circ F \circ G \right) \right) = \text{ran} \left( \text{ran} \left( \text{dom} \left( (F : \text{dom} (G : t)) \circ F \right) \circ G \right) \right) \subseteq t
\]
It is also easy to show that \( F \) is modally deterministic if it is domain-minimal:
\[
\text{ran} \left( \text{dom} \left( (F : t) \circ F \right) \right) = \text{ran} \left( (F : t) \right) \quad \text{Lemma 3.15 with} \ R = F : t
\]
\[
= (\text{ran} F : t) \quad \text{export law Lemma 3.2.(4) for ran}
\]
\[
\subseteq t \quad \text{ran} F \text{ is subidentity}
\]
(For the converse implication, standard Kleene algebras of regular languages are given as a counter-example in [15]; this relies on the fact that, there, the only subidentities on the only object are \( \mathbb{I} \) and \( \perp \), which obviously makes all morphisms modally deterministic.)

4. Kleene semigroupoids

Kleene algebras are a generalisation of the algebra of regular languages; the typed version [30] is an extension of upper semilattice categories with zero morphisms. Since the reflexive aspect of the Kleene star is undesirable in semigroupoids, we adapt the axiomatisation by Kozen [28] to only the transitive aspect.

**Definition 4.1.** A *Kleene semigroupoid* is an upper semilattice semigroupoid with zero morphisms such that on homsets of endomorphisms there is an additional unary operation \( \_^+ \) which satisfies the following axioms for all \( R : A \rightarrow A, Q : B \rightarrow A, \) and \( S : A \rightarrow C \):
\[
R \sqcup R^+ \circ R^+ = R^+ \quad \text{recursive definition}
\]
\[
Q \circ R \sqsubseteq Q \quad \Rightarrow \quad Q \circ R^+ \sqsubseteq Q \quad \text{right induction}
\]
\[
R \circ S \sqsubseteq S \quad \Rightarrow \quad R^+ \circ S \sqsubseteq S \quad \text{left induction}
\]
It is interesting to compare this with Kozen’s definition, where the recursive definition axiom for \( R^* \) is
\[
\mathbb{I} \sqcup R \sqcup R^* \circ R^* = R^*
\]
so the only change to obtain our definition above is the omission of the join with the identity from the left-hand side of the recursive definition.

Kozen also states the induction laws with inclusions in the conclusion, although for reflexive transitive closure, equality immediately ensues. This is not the case in our definition, so this direct definition of transitive closure is in some sense more “satisfactory” than the reflexive transitive variant.

**Lemma 4.2.** Every Kleene category is a Kleene semigroupoid, with $R^+ := R : R^*$.

**Proof.** We obtain the recursive definition axiom of Kleene semigroupoids from simple Kleene category properties:

\[
\begin{align*}
R \sqcup R^+ & = R \sqcup R : R^* ; R^* \quad R^+ = R : R^* \\
& = R \sqcup R : R^* ; R^* \quad R^* : R = R : R^* \\
& = R : (\sqcup R : R^*) \quad R^* : R^* = R^*
\end{align*}
\]

Since, in Kleene categories, $R^+ \sqsubseteq R^*$, the induction axioms of Kleene semigroupoids follow immediately from those of Kleene categories. □

**Lemma 4.3.** Every Kleene semigroupoid where all objects have identities is a Kleene category, with $R^* := \sqcup R^+$. 

**Proof**

\[
\begin{align*}
\sqcup R \sqcup R^* & = \sqcup R \sqcup (\sqcup R^+) : (\sqcup R^+) \\
& = \sqcup R \sqcup (\sqcup R^+) \sqcup R^+ : (\sqcup R^+) \\
& = \sqcup R \sqcup (\sqcup R^+) \sqcup R^+ \sqcup R^+ : (\sqcup R^+) \\
& = \sqcup R \sqcup R^+ \sqcup R^+ : (R^+)^* \\
& = R^* \quad R \sqcup R^+ : (R^+)^* = R^+
\end{align*}
\]

The induction axioms follow from the identity property—we show only one:

\[
\begin{align*}
Q : R \sqsubseteq Q & \Rightarrow Q : R^+ \sqsubseteq Q \\
\iff Q : R^+ \sqsubseteq Q \quad \land \quad Q : \sqcup \sqsubseteq Q \\
\iff Q : R^+ \sqcup Q : \sqcup \sqsubseteq Q \\
\iff Q : (R^+ \sqcup \sqcup) \sqsubseteq Q \\
\iff Q : R^* \sqsubseteq Q
\end{align*}
\]

Together, these two lemmas demonstrate that the definition of Kleene semigroupoids precisely defines “Kleene categories minus identities”. Lemma 4.3 also shows that all properties of transitive closure that hold in Kleene categories and can be expressed in the language of Kleene semigroupoids are true in each Kleene semigroupoid, since they hold in the Kleene category obtained by just adjoining identities.

Semigroupoids of finite relations between infinite types provide one concrete example of a Kleene semigroupoid—thanks to its “finitary nature”, transitive closure still exists, even though the $\sqcup$-semilattice of concrete finite relations between two infinite sets is not complete.

While transitive closure of concrete relations does preserve finiteness, this is not the case for language-based models; nevertheless it should be relatively straightforward to show that the “automatic semigroupoids” of [27] are Kleene semigroupoids.

**Definition 4.4.** A Kleene semigroupoid with converse is a Kleene semigroupoid that is at the same time an OSGC, and the involution law for transitive closure holds: $(R^+)^* = (R^*)^+$.

In Kleene semigroupoids with converse, difunctional closures always exist, and can be calculated as

\[
R^\square := R \sqcup (R : R)^+ : R.
\]
5. Restricted residuals

We have seen in Section 1.3 that, in the interesting semigroupoid of finite relations between arbitrary sets, residuals do not in general exist. But we can define a set of restricted residuals that do exist for finite relations:

**Definition 5.1.** For morphisms \( S : A \rightarrow C \) and \( Q : A \rightarrow B \) and \( R : B \rightarrow C \) in an ordered semigroupoid with domain and range, we define:

- the *restricted right-residual* \( Q \cdot S \)

\[
\forall Y : B \rightarrow C \quad Y \sqsubseteq Q \cdot S \iff Q \cdot Y \sqsubseteq S \quad \text{and} \quad \text{dom } Y \sqsubseteq \text{ran } Q.
\]

- the *restricted left-residual* \( S / \cdot R \)

\[
\forall X : A \rightarrow B \quad X \sqsubseteq S / \cdot R \iff X \cdot R \sqsubseteq S \quad \text{and} \quad \text{ran } X \sqsubseteq \text{dom } R.
\]

- and (in an OSGC) the *restricted symmetric quotient* \( Q / S \)

\[
\forall Y : B \rightarrow C \quad Y \sqsubseteq Q / S \iff Q \cdot Y \sqsubseteq S \quad \text{and} \quad \text{dom } Y \sqsubseteq \text{ran } Q
\]

\[
\text{and } S \cdot Y \sqsubseteq Q \quad \text{and} \quad \text{ran } Y \sqsubseteq \text{ran } S.
\]

Where both restricted residuals exist, we obtain the restricted symmetric quotient as their meet, just as in allegories [20]:

\[
Q / S = (Q \cdot S) \sqcap (Q \cdot S)
\]

For concrete relations, we have (using infix notation for relations):

\[
y(Q \cdot S)x \quad \text{iff} \quad \forall x . x Q y \Rightarrow x Sz \quad \text{and} \quad \exists x . x Q y
\]

\[
x(S / \cdot R)y \quad \text{iff} \quad \forall z . y R z \Rightarrow x Sz \quad \text{and} \quad \exists z . y R z
\]

For finite relations between (potentially) infinite types, this definition chooses the largest domain, respectively range, on which each residual is still guaranteed to be finite if its arguments are both finite.

Where residuals exist, the restricted residuals can be defined using the unrestricted residuals:

**Lemma 5.2.** In an ordered semigroupoid with domain and range, if the residual \( Q \cdot S \) exists, then the restricted residual \( Q \cdot S \) exists, too, and we have:

\[
Q \cdot S = \text{ran } Q : (Q \cdot S).
\]

Dually, if \( S / R \) exists, then \( S / R \) exists, too, and we have:

\[
S / R = (S / R) : \text{dom } R.
\]

**Proof.** We only show the former:

\[
Y \sqsubseteq \text{ran } Q : (Q \cdot S) \iff Y \sqsubseteq \text{ran } Q : (Q \cdot S) \wedge \text{dom } Y \sqsubseteq \text{ran } Q
\]

\[
\iff Y \sqsubseteq (Q \cdot S) \wedge \text{dom } Y \sqsubseteq \text{ran } Q \quad \text{ran } Q \text{ is subidentity}
\]

\[
\iff Q : Y \sqsubseteq S \wedge \text{dom } Y \sqsubseteq \text{ran } Q \quad \text{residual def.}
\]

\[
\iff Y \sqsubseteq Q / S \quad \text{restr. res. def.} \quad \square
\]

This “definedness restriction” essentially “takes away” from the standard residuals only the “uninteresting part”, where the corresponding universally quantified formula is trivially true, and therefore is still useful in relational formalisations in essentially the same way as the “full” residuals. We can formalise this “uninteresting part”: 
Lemma 5.3. In an ordered semigroupoid with domain and range, if for $Q : A \to B$ and $S : A \to C$, 
- the restricted residual $Q \setminus S$,
- the range $\text{ran } Q$,
- the identity $\mathbb{1}_B$ and $\mathbb{1}_C$,
- the difference $\mathbb{1}_B - \text{ran } Q$,
- and the restricted top morphism (see Definition 3.4) $\top_{(\mathbb{1}_B - \text{ran } Q),\mathbb{1}_C}$
all exist, then the residual $Q \setminus S$ exists, too, and we have:

$$Q \setminus S = (Q \setminus S) \sqcup \top_{(\mathbb{1}_B - \text{ran } Q),\mathbb{1}_C}.$$ 

Proof

$$Y \subseteq (Q \setminus S) \sqcup \top_{(\mathbb{1}_B - \text{ran } Q),\mathbb{1}_C}$$
$$\iff (\text{ran } Q) : Y \subseteq Q \setminus S$$
$$\iff Q : (\text{ran } Q) : Y \subseteq S$$ restricted residual
$$\land \text{ dom } ((\text{ran } Q) : Y) \subseteq \text{ran } Q$$
$$\iff Q : Y \subseteq S$$
$$\iff Y \subseteq (Q \setminus S)$$ residual \hfill \Box$$

We now can complete the semi-allegory hierarchy along the lines of the allegory hierarchy of Freyd and Scedrov:

Definition 5.4. A distributive semi-allegory is a semi-allegory that is also an upper semilattice semigroupoid with zero morphisms.

The class of semigroupoids corresponding to division allegories is naturally defined using restricted residuals:

Definition 5.5. A division semi-allegory is a distributive semi-allegory in which all restricted residuals exist.

We then complete the hierarchy of semigroupoids by joining the semi-allegory branch with the Kleene branch, adding pseudo-complements, and our “finite replacement” for largest elements in homsets from Definition 3.4:

Definition 5.6. A Dedekind semigroupoid is a division semi-allegory that is also a Kleene semigroupoid and has pseudo-complements, and where for any two subidentities $p : A \to A$ and $q : B \to B$ the restricted top morphism $\top_{p,q}$ exists.

6. Basic datatype definitions

We now turn to characterising datatype “constructions” in the relational semigroupoid setting.

So far, speaking in terms of our motivating example of finite relations between infinite sets, we only needed to eliminate infinite constants, like $\mathbb{1}$ and $\top$, and replace them by infinite collections of their finite counterparts, or replace operations yielding infinite results for finite arguments, like complement and residuals, with operations that restrict the view on certain finite “essential parts” of those results, namely difference and restricted residuals.

For datatype constructions, we have to deal with infinite relations, like product projections, that are not so easily boiled down to finite effects.

We use the relatively simple setting of subobject construction to emphasise the issues involved through extensive discussion of an example, and illustrate our approach to solving these issues.

In the following subsections, we then also apply this approach to the other datatype constructions.

6.1. Subobjects

Let us consider again the motivating example of finite relations between infinite sets. Finite subsets of infinite sets do not involve any infinite relations, and are therefore no problem—we can use the same characterisation as in ordered
categories with converse and domain, characterising the subobject using a finite subidentity, the (finite) identity on the subobject, and a finite injection mapping.

Characterising infinite subsets of infinite sets using only finite relations is more interesting. As an example, consider the set \( \mathbb{Z} \) of integers and its subset \( \mathbb{N} \) of natural numbers, with the infinite injection mapping \( \mathbb{N}to\mathbb{Z} : \mathbb{N} \to \mathbb{Z} \) with range \( \text{ran}\ Nto\mathbb{Z} = \text{nonNeg} \), where \( \text{nonNeg} \) is the infinite subidentity on \( \mathbb{Z} \) containing all non-negative integers.

In the setting of ordered categories with converse and domain, \( \mathbb{N} \) together with \( \text{Nto}\mathbb{Z} \) is a subobject of \( \mathbb{Z} \) for \( \text{nonNeg} \) since \( \text{Nto}\mathbb{Z} \) is an injective mapping with range \( \text{nonNeg} \).

For any finite relation \( R : A \leftrightarrow \mathbb{Z} \), the composition \( R : \text{Nto}\mathbb{Z} \) is of course finite again. We can therefore define an operation \( (\_)^{\mathbb{Z}\to\mathbb{N}} \) mapping finite relations \( R : A \leftrightarrow \mathbb{Z} \) to finite relations \( R^{\mathbb{Z}\to\mathbb{N}} : A \leftrightarrow \mathbb{N} \), with

\[ R^{\mathbb{Z}\to\mathbb{N}} := R \cdot \text{Nto}\mathbb{Z} \]

Now the question is whether we can axiomatise this operation using only finite relations in a way that precisely enforces the subobject situation.

Note that for each subidentity \( k \) on \( \mathbb{Z} \) we obtain with

\[ (k^{\mathbb{Z}\to\mathbb{N}})^- = (k : \text{Nto}\mathbb{Z})^- = Nto\mathbb{Z} : k^- \subseteq \text{Nto}\mathbb{Z} \]

a finite “approximation” of the infinite injection mapping. Also, we need to expect that \( (\_)^{\mathbb{Z}\to\mathbb{N}} \) is determined by its action on subidentities, because

\[ R^{\mathbb{Z}\to\mathbb{N}} = (R : \text{ran}\ R)^{\mathbb{Z}\to\mathbb{N}} = (R : \text{ran}\ R) : \text{Nto}\mathbb{Z} = R : (\text{ran}\ R : \text{Nto}\mathbb{Z}) = R : (\text{ran}\ R)^{\mathbb{Z}\to\mathbb{N}} \]

and each subidentity is in the image of \( \text{ran} \), since \( k = \text{ran} k \).

While working from subidentities on \( \mathbb{Z} \), their restriction to non-negative integers is another important operation in this context:

\[ k^{\text{nonNeg}} := k : \text{nonNeg} \]

This operation is obviously a kernel operation, since it satisfies the following three laws:

- decreasing: \( k^{\text{nonNeg}} \subseteq k \),
- monotone: \( k_1 \subseteq k_2 \) implies \( k_1^{\text{nonNeg}} \subseteq k_2^{\text{nonNeg}} \),
- idempotent: \( (k^{\text{nonNeg}})^{\text{nonNeg}} = k \).

This operation can easily be derived from \( (\_)^{\mathbb{Z}\to\mathbb{N}} \), via \( k^{\text{nonNeg}} = \text{dom}\ ((k^{\mathbb{Z}\to\mathbb{N}})^-), \) but since \( (\_)^{\mathbb{Z}\to\mathbb{N}} \) does not involve the object \( \mathbb{N} \) in any way, it provides us with an appropriate starting point for our abstract definition.

Whether we use the full version of \( (\_)^{\mathbb{Z}\to\mathbb{N}} \) or only its restriction to subidentities is essentially a matter of taste. We choose the former since for many library designs it introduces fewer preconditions that cannot be enforced by the type system.

**Definition 6.1.** In a OSGC with domain, let an object \( A \) and a kernel operation \( K \) on the subidentities on \( A \) be given.

A subobject restriction (from \( A \) to \( S \)) for \( K \) is a triple \((S, (A\to\mathbb{A})\to\mathbb{S}, (S\to\mathbb{A})\to\mathbb{A})\) consisting of an object \( S \) and two operations

- restriction \( (A\to\mathbb{A})\to\mathbb{S} \), mapping \( R : B \to A \) to \( R(A\to\mathbb{A}) : B \to S \), and
- extension \( (S\to\mathbb{A})\to\mathbb{A} \), mapping \( Q : C \to S \) to \( Q(S\to\mathbb{A}) : C \to A \)

where the following laws hold for arbitrary objects \( B, C \), morphisms \( R : B \to A \) and \( Q : C \to S \), subidentities \( a, a_1, a_2 \) on \( A \), and subidentities \( s \) on \( S \):

1. \( R(A\to\mathbb{A}) = R : (\text{ran}\ R)(A\to\mathbb{A}) \).
2. \( Q(S\to\mathbb{A}) = Q : (\text{ran}\ Q)(S\to\mathbb{A}) \).
3. \( (\text{ran}\ (a(A\to\mathbb{A})))S(A) = (a(A\to\mathbb{A}))^- \).
4. \( (\text{ran}\ (s(S\to\mathbb{A})))A(S) = (s(S\to\mathbb{A}))^- \).
5. \( K a = (a(A\to\mathbb{A}))S(A) \).
6. \( s = (s(S\to\mathbb{A}))A(S) \).
Lemma 6.2. If \((S, \rightarrow, \leftarrow)\) is a subobject restriction for \(K\), then
\(\text{ran} (R^{A \rightarrow S}) = \text{ran} ((\text{ran} R)^{A \rightarrow S}) \text{ and } \text{ran} (Q^{S \rightarrow A}) = \text{ran} ((\text{ran} Q)^{S \rightarrow A}).\)
(1) \(\text{ran} (R^{A \rightarrow S}) = \text{ran} (R \cdot (\text{ran} R)^{A \rightarrow S})\)
(2) \((R^{A \rightarrow S})^{S \rightarrow A} = R \cdot K \cdot (\text{ran} R)\)
(3) \((Q^{S \rightarrow A})^{A \rightarrow S} = Q\)
(4) For each subidentity \(a\) on \(A\) and \(s\) on \(S\), \(a^{A \rightarrow S}\) and \(s^{S \rightarrow A}\) are univalent and injective.
(5) \(\text{dom} (s^{S \rightarrow A}) = s \text{ and } \text{dom} (a^{A \rightarrow S}) = K a\).

Proof
(1) \(\text{ran} (R^{A \rightarrow S}) = \text{ran} (R \cdot (\text{ran} R)^{A \rightarrow S}) = \text{ran} ((\text{ran} R) \cdot (\text{ran} R)^{A \rightarrow S})\)\)
locality of \(\text{ran} \)
(2) \(\text{idempot. ran}\), Definition 6.1.(1)
(3) \(\text{ran} (R \cdot (\text{ran} R)^{A \rightarrow S})^{S \rightarrow A} \cdot (\text{ran} (\text{ran} R)^{A \rightarrow S})^{S \rightarrow A} = (R^{A \rightarrow S})^{S \rightarrow A} \cdot (\text{ran} (\text{ran} R)^{A \rightarrow S})^{S \rightarrow A}\)
(4) \(\text{Definition 6.1}(3)\)
(5) \(\text{Definition 6.1}(4)\)

Lemma 6.3. If \(\lambda : S \rightarrow A\) is a subobject injection in an ordered category with converse and domain, i.e., an injective mapping, then defining \(R^{A \rightarrow S} := R \cdot \lambda\) and \(Q^{S \rightarrow A} := Q \cdot \lambda\) produces a subobject restriction from \(A\) to \(S\) for \(K\) defined by
\(K a := a \cdot \text{ran} \lambda\).

Proof. By straightforward calculation. \(\Box\)

Lemma 6.4. If a subobject restriction \((S, \rightarrow, \leftarrow)\) is given in an ordered category with converse and domain, then \(\lambda := \text{inj}_{S \rightarrow A}\) is an injective mapping from \(S\) to \(A\).

Proof. This follows directly from Lemma 6.2.(4) and (5). \(\Box\)
6.2. Object equivalences

We have not yet shown that a kernel operation $K$ monomorphically determines a subobject restriction—the finite-relation example shows that isomorphisms may not be available.

But since isomorphisms are just special cases of monomorphisms, we can specialise our subobject restriction analogously:

Definition 6.5. An object equivalence in a semi-allegory is a subobject restriction for the identity operation.

It is easy to check that this does in fact give rise to an equivalence relation on objects, which we can use to appropriately show monomorphism of subobject restrictions:

Theorem 6.6. If $(S_1, \_A \mapsto S_1, S_1 \mapsto A)$ and $(S_2, \_A \mapsto S_2, S_2 \mapsto A)$ are two subobject restrictions of $A$ for $K$, then an object equivalence from $S_1$ to $S_2$ is defined by

$$R^{S_1 \mapsto S_2} := (R^{S_1 \mapsto A})_{A \mapsto S_2}$$

$$Q^{S_2 \mapsto S_1} := (Q^{S_2 \mapsto A})_{A \mapsto S_1}$$

Proof

(1) $R^{S_1 \mapsto S_2}$

$$= (R^{S_1 \mapsto A})_{A \mapsto S_2}$$

$$= (R^{S_1 \mapsto A} : (\text{ran}(R^{S_1 \mapsto A}))_{A \mapsto S_2}$$

$$= R : (\text{ran}(R)_{S_1 \mapsto A})_{A \mapsto S_2}$$

$$= R : (((\text{ran}(R))_{S_1 \mapsto A})_{A \mapsto S_2}$$

$$= R : (\text{ran}(R)_{S_1 \mapsto A})_{A \mapsto S_2}$$

(2) $Q^{S_2 \mapsto S_1} = Q : (\text{ran}(Q))_{S_2 \mapsto S_1}$ is shown in the same way.

(3) $(\text{ran}(S_1^{S_1 \mapsto S_2}))_{S_2 \mapsto S_1}$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

$$= (((\text{ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_2}))_{A \mapsto S_1}$$

(4) $(\text{ran}(S_1^{S_1 \mapsto S_2}))_{S_1 \mapsto S_2} = (S_2^{S_2 \mapsto S_1})$ is shown analogously.

(5) $s_1 = (((s_1^{S_1 \mapsto A})_{A \mapsto S_1}S_1 \mapsto A)_{A \mapsto S_1}S_1 \mapsto A)_{A \mapsto S_1}$

$= (s_1^{S_1 \mapsto A} : (K \text{ ran}(S_1^{S_1 \mapsto A}))_{A \mapsto S_1}$$

$= (((s_1^{S_1 \mapsto A})_{A \mapsto S_2}))_{S_2 \mapsto A} : A \mapsto S_1$$

$= (s_1^{S_1 \mapsto A})_{A \mapsto S_2}S_2 \mapsto S_1$$

(6) $s_2 = (S_2^{S_2 \mapsto S_1})_{S_1 \mapsto S_2}$ is shown in the same way. □
6.3. Quotients

As we have seen above, the setting of ordered categories with converse is sufficient for defining equivalence relations, and a quotient for an equivalence relation \( \Xi : \mathcal{A} \to \mathcal{A} \) is defined by an object \( Q \) and a surjective projection mapping \( \chi : \mathcal{A} \to Q \).

On infinite sets, each equivalence relation is infinite. However, there are several different cases:

- If \( \Xi = \mathbb{1} \) is finite, then the quotient is infinite.
- If \( \Xi = \mathbb{1} \) is infinite, but each equivalence class is finite, then the quotient is still infinite.
- If there are finitely many equivalence classes, at least some of them have to be infinite, but the quotient would be finite.
- Of course there could also be infinitely many infinite equivalence classes (together with any number of finite equivalence classes) again producing an infinite quotient.

If \( \chi \) is the projection into the quotient of an equivalence relation, then \( R : \chi \) is finite for finite \( R \), but \( Q : \chi \) does not necessarily preserve the finiteness of \( Q \).

The main reason why we needed the extension operation \( \chi : \mathcal{S} \mapsto \mathcal{A} \) as part of the subobject restriction interface was for being able to obtain totality of the injection mapping.

This corresponds to surjectivity of the projection here, and since \( \chi : \mathcal{S} \mapsto \mathcal{A} \) does not preserve finiteness, we cannot use it directly for this purpose. Instead, we resort to admitting “locally total” subrelations \( q : \mathcal{Q} \mapsto \mathcal{Q} \) of \( \chi : \mathcal{Q} \). These (possibly non-deterministic) “choices of representatives” are aesthetically not satisfying, but pragmatically acceptable, in particular from the point of view of our motivating example of programming with finite relations between infinite types, where it is natural to expect some representative to be accessible for any equivalence class for which a handle can be obtained through a library interface.

**Definition 6.7.** In a OSGC with domain, let an object \( \mathcal{A} \) be given, and a family \( \Xi \) of partial equivalence relations (transitive and symmetric morphisms) containing \( \Xi_\alpha \) for each subidentity \( \alpha \) on \( \mathcal{A} \), and satisfying the following condition:

- if \( \alpha \subseteq \beta \), then \( \Xi_\alpha = \alpha : \Xi_\beta \colon \alpha \).

A quotient contraction (from \( \mathcal{A} \) to \( \mathcal{Q} \)) for \( \Xi \) is a tuple \( \langle \mathcal{Q}, \_\mathcal{A} \mapsto \mathcal{Q}, \_\mathcal{Q} \mapsto \mathcal{A} \rangle \) consisting of an object \( \mathcal{S} \) and two operations

- contraction \( \_\mathcal{A} \mapsto \mathcal{Q} \), mapping \( R : \mathcal{B} \to \mathcal{A} \) to \( R \circ \_\mathcal{A} \mapsto \mathcal{Q} : \mathcal{B} \to \mathcal{Q} \), and
- expansion \( \_\mathcal{Q} \mapsto \mathcal{A} \), mapping \( S : \mathcal{C} \to \mathcal{Q} \) to \( S \circ \_\mathcal{Q} \mapsto \mathcal{A} : \mathcal{C} \to \mathcal{A} \).

such that the following laws hold:

1. \( R \circ \_\mathcal{A} \mapsto \mathcal{Q} = R \circ (\text{ran } R) \circ \_\mathcal{A} \mapsto \mathcal{Q} \).
2. \( \text{ran } \_\mathcal{Q} \mapsto \mathcal{A} = \text{ran } (\text{ran } S) \circ \_\mathcal{Q} \mapsto \mathcal{A} \).
3. \( \text{ran } (q \circ \_\mathcal{Q} \mapsto \mathcal{A}) \circ \_\mathcal{A} \mapsto \mathcal{Q} = (q \circ \_\mathcal{Q} \mapsto \mathcal{A}) \).
4. \( \alpha \circ \_\mathcal{Q} \mapsto \mathcal{A} : (\_\mathcal{A} \mapsto \mathcal{Q}) \circ \_\mathcal{A} \mapsto \mathcal{Q} = \Xi_\alpha \).
5. \( (R \circ \_\mathcal{Q} \mapsto \mathcal{A}) \circ \_\mathcal{Q} \mapsto \mathcal{A} \subseteq R : \Xi_\alpha \).
6. \( (S \circ \_\mathcal{Q} \mapsto \mathcal{A}) \circ \_\mathcal{Q} \mapsto \mathcal{Q} = S \).

Again, relatively straightforward calculations show that quotient contractions are unique up to object equivalence, and that in an ordered category with converse and domain, every quotient gives rise to a quotient contraction, and vice versa.

7. Direct products

Since product projection mappings can again be infinite, we axiomatise finiteness-preserving “usage patterns” of the potentially infinite projection functions \( \pi : \mathcal{A} \times \mathcal{B} \to \mathcal{A} \) and \( \rho : \mathcal{A} \times \mathcal{B} \to \mathcal{B} \).

- The fork operation as introduced in the context of relation algebras by Haeberer et al. [22] is, in allegories with direct products, a derived operation, defined by \( R \cup S := R \cdot \pi \cap S \cdot \rho \), which preserves finiteness.
The “target projection” operations which compose their argument with the product projections also preserve finiteness; in an allegory with direct products we would define:

\[ P^\pi := P : \pi \quad \text{and} \quad P^\rho := P : \rho. \]

These can be axiomatised without projections:

**Definition 7.1.** In a semi-allegory, given two objects \( A \) and \( B \) such that for all subidentities \( a \) on \( A \) and \( b \) on \( B \) the restricted top morphism \( \top_{a,b} \) exists, a direct product \( (P, \nabla, \nabla^\pi, \nabla^\rho) \) for \( A \) and \( B \) consists of:

- a product object \( P \),
- a fork operation \( \nabla \) taking, for any object \( C \), two morphisms \( R : C \to A \) and \( S : C \to B \) to a morphism \( (R \nabla S) : C \to \mathcal{P} \) (to make the typing explicit, we will occasionally write \( R \nabla : S \)),
- two target projection operations \( \nabla^\pi \) and \( \nabla^\rho \) taking any morphism \( Q : C \to \mathcal{P} \) to \( Q^\pi : C \to A \), respectively to \( Q^\rho : C \to B \).

In addition, we introduce the following derived operations for two sub-identities \( a \) on \( A \) and \( b \) on \( B \):

- **local projections:**
  \[ \pi_{P,a,b} := ((\text{dom } \top_{a,b}) \nabla A \top_{a,b})^{-} \quad \text{and} \quad \rho_{P,a,b} := (\top_{b,a} \nabla B \text{ran } \top_{a,b})^{-}. \]

- **local product:**
  \[ a \times b := \pi_{P,a,b} \nabla_P \rho_{P,a,b}. \]

The following axioms are required to hold, with objects \( C \) and \( D \), morphisms \( R, R_1 : C \to A \); \( R_2 : D \to A \); \( S, S_1 : C \to B \); \( S_2 : D \to B \); and \( P : C \to \mathcal{P} \) and \( Q : D \to \mathcal{C} \), and sub-identities \( a \) on \( A \) and \( b \) on \( B \):

1. \( (R \nabla S) = R : \pi_{P,\text{ran } R, \text{ran } S} \nabla S : \rho_{P,\text{ran } R, \text{ran } S} \)
2. \( (R_1 \nabla S_1) : (R_2 \nabla S_2)^{-} = R_1 : R_2^{-} \nabla S_1 : S_2^{-} \)
3. The local product \( a \times b \) is a subidentity again, \( \nabla_P \)
4. \( \text{ran } P \subseteq a \times b \Leftrightarrow \text{ran } (P^\pi) \subseteq a \land \text{ran } (P^\rho) \subseteq b \)
5. \( Q : (R \nabla S) \subseteq (Q : R) \nabla (Q : S) \), with equality if \( Q \) is univalent
6. \( \text{dom } (R \nabla S) = (\text{dom } R) : (\text{dom } S) \)
7. \( (R \nabla R_1) \nabla (S \nabla S_1) = (R \nabla S) \nabla (R_1 \nabla S_1) \) (implying monotonicity of \( \nabla )
8. \( P^\pi = P : \pi_{P,\text{ran } (P^\pi), \text{ran } (P^\rho)} \quad \text{and} \quad P^\rho = P : \rho_{P,\text{ran } (P^\pi), \text{ran } (P^\rho)} \)
9. \( (R \nabla S)^\pi = (\text{dom } S) : R \quad \text{and} \quad (R \nabla S)^\rho = (\text{dom } R) : S \)
10. \( (Q : P)^\pi = Q : (P^\pi) \quad \text{and} \quad (Q : P)^\rho = Q : (P^\rho) \)
11. \( \text{dom } (P^\pi) = \text{dom } P = \text{dom } (P^\rho) \)

(We do not claim that these axioms are independent of each other.)

It is straightforward to check that a direct product in an allegory, i.e., a tabulation of a greatest morphism, is also a direct product according to Definition 7.1. The converse implication is more interesting; in preparation for its proof, we first list some properties of the operations defined above:

**Lemma 7.2.** In a direct product \( (P, \nabla, \nabla^\pi, \nabla^\rho) \) according to Definition 7.1:

- the local projections \( \pi_{P,a,b} \) and \( \rho_{P,a,b} \) and the local product \( a \times b \) preserve meets in the subidentities \( a \) and \( b \) (and are therefore monotonic),
- \( \text{dom } \pi_{P,a,b} = a \times b = \text{dom } \rho_{P,a,b} \)
This is a subidentity by (3), so we have
\[ \pi : \pi \cap \rho = \pi \vee \rho. \]

The converse inclusion follows from (4):
\[ \pi \cap \rho = \pi \vee \rho \subseteq \square_p. \]
Direct sums with injections and direct powers with element relations can be dealt with similarly; for the latter, the use of restricted residuals implies that set comprehension is typically restricted to non-empty sets.

8. Implementation

For designing a library providing a datatype of concrete relations that implements the various semigroupoid interfaces from the previous sections, the following issues are of crucial importance:

- Which semigroupoid(s) will be supported? How will the objects of the semigroupoid be related to host language data types? That is, how much of the semigroupoid type checking will be done by the host language type system?
- How will the different interfaces to the same concrete semigroupoid be managed in the programming language?
- How will concrete relations be represented? That is, what kind of low-level data structure will be used to represent relations?

We now discuss each of these questions in the context of the purely functional programming language Haskell [36], which, due to its mathematical nature and due to the algebraic culture established in its community by pioneers like Bird and deMoor [7], Bird and Wadler [8], and Hughes [23] should be a “friendly environment” for semigroupoid-based library interfaces. As far as we describe implementation decisions, they refer to the Haskell library Data.Rel that is becoming available at URL http://relmics.mcmaster.ca/~kahl/RATH/.

8.1. “Natural” Haskell relations

The library Data.Rel in particular provides a binary relation type constructor Rel that is intended to be a natural pendant to the finite partial function type constructor Map provided by Data.Map. For two Haskell types a and b, an element of type Map a b can be understood as a partial function from the set of (sufficiently defined) elements of a to the set of elements of b. (Elements of a need to be at least sufficiently defined so that lookup can determine equality; normally this implies that elements of a used as keys in Maps need to be fully defined, but for some (user-defined) types, the definition of the equality and ordering predicates can also handle elements that are only partially defined.)

Similarly, the type Rel a b implements finite relations between the types a and b, where a relation can be understood as a set of pairs, and pair components are restricted to those elements on which the equality and ordering functions provided by the Ord interface terminate. This means roughly that Rel a b is intended to be used to establish relationships only between finite, fully defined elements of a and b, but no relationships with the (partially) undefined (and infinite) elements that also inhabit most Haskell datatypes.

To ease adoption by programmers used to the interfaces of Data.Set and Data.Map, and also to ease interfacing with non-relation-algebraic aspects of applications, Data.Rel includes a sub-interface that strictly follows the naming and argument order conventions established by Data.Set and Data.Map, and allows to interface between relations of type Rel a b and elements of the types a and b, and also with lists of pars, of type [(a, b)], which are the de-facto standard internal interchange format in Haskell (a constraint "Ord a" expresses that a linear ordering needs to be available on type a): Besides these, we also provide some point-level functions following the naming and argument order conventions of Data.Set and Data.Map:

- empty :: Rel a b
- member :: (Ord a, Ord b) ⇒ a → b → Rel a b → Bool
- insert :: (Ord a, Ord b) ⇒ a → b → Rel a b → Rel a b
- fromList :: (Ord a, Ord b) ⇒ [(a, b)] → Rel a b
- toList :: (Ord a, Ord b) ⇒ Rel a b → [(a, b)]

Besides these, the data type constructor Rel exposes an interface with relation-algebraic flavour, but providing only operations from appropriate semigroupoids—the ones listed here are all available in Dedekind semigroupoids:
There are three ways to situate the objects of the relation semigroupoid underlying a relation datatype with respect to the host language (Haskell) type system:

"Types as objects" guarantees full type safety.

"Sets as objects" offers finer granularity at the expense of dynamic compatibility checks for relations on possibly different subsets of the same type.

"Elements as objects" uses elements of a single datatype as objects, with no support from the type system for relation compatibility.

The last approach has been taken by the relation-algebraic experimentation toolkit RATH [24], and is motivated there by a point of view that considers whole relation algebras as data items, and, for example, testing of all objects for some property is a natural operation.

Unlike that situation in RATH, we are here concerned with concrete relation-algebraic operations on finite relations as a programming tool in a polymorphically typed programming language. In this context, both of the first two views have natural applications, so we support both, and we support a uniform programming style across the two views by organising all relational operations, including those listed in Section 8.1, into a wide range of Haskell type classes exported by \texttt{Data.Rel}, with even finer granularity than the hierarchy of definitions in Section 2, and supplemented by classes for the corresponding structures with identities, including categories, allegories, and relation algebras.

8.2. Programming in different semigroupoids

There are three ways to situate the objects of the relation semigroupoid underlying a relation datatype with respect to the host language (Haskell) type system:

"Types as objects" guarantees full type safety.

"Sets as objects" offers finer granularity at the expense of dynamic compatibility checks for relations on possibly different subsets of the same type.

"Elements as objects" uses elements of a single datatype as objects, with no support from the type system for relation compatibility.

The last approach has been taken by the relation-algebraic experimentation toolkit RATH [24], and is motivated there by a point of view that considers whole relation algebras as data items, and, for example, testing of all objects for some property is a natural operation.

Unlike that situation in RATH, we are here concerned with concrete relation-algebraic operations on finite relations as a programming tool in a polymorphically typed programming language. In this context, both of the first two views have natural applications, so we support both, and we support a uniform programming style across the two views by organising all relational operations, including those listed in Section 8.1, into a wide range of Haskell type classes exported by \texttt{Data.Rel}, with even finer granularity than the hierarchy of definitions in Section 2, and supplemented by classes for the corresponding structures with identities, including categories, allegories, and relation algebras.
Exporting all relational operations as class members first of all makes relational programming implementation independent: Applications written only against these class interfaces can be used, without change, on any new implementation.

Providing a class hierarchy with very fine granularity in addition extends the scope of possible models that can be implemented; currently we only have implementations of concrete relations, but the machinery can easily be extended to, for example, relational graph homomorphisms [25], or fuzzy relations.

8.2.1. Types as objects

It is quite obvious from the presentation of the Data.Rel interface that the choice of relation semigroupoid here is essentially the same as in the specification notations Z [41] and B [2], where only certain sets are types: If different subsets $A_1, A_2 : \mathbb{P} A$ and $B_1, B_2 : \mathbb{P} B$ of two types $A$ and $B$ are given, the relations in $A_1 \leftrightarrow B_1$ are still considered as having the same type as the relations in $A_2 \leftrightarrow B_2$, namely the type $A \leftrightarrow B$. Therefore, if $R : A_1 \leftrightarrow B_1$ and $S : A_2 \leftrightarrow B_2$, writing for example $R \cap S$ is perfectly legal and well-defined.

This means that in this view we are operating in a relation semigroupoid that has only types as objects—we realise this in the Rel relation type constructor. This has the advantage that Haskell type checking implements semigroupoid morphism compatibility checking, so relation-algebraic Rel expressions are completely semigroupoid-type-safe. Since some Haskell types are infinite, Rel can implement only semigroupoid interfaces, up to Dedekind semigroupoids, but no category interfaces. Also, Rel can only provide pseudo-complements (difference), not complements, just like the de-facto-standard library module Data.Set.

8.2.2. Sets as objects

The situation described above is different from the point of view taken by the category Rel which has all sets as objects. If an implementation is to realise this point of view, then the empty relation $\emptyset : \{0, 1\} \leftrightarrow \{0, 1, 2\}$, for example, must be different from the empty relation $\emptyset : \{0, 1, 2, 3\} \leftrightarrow \{0\}$, since in a category or semigroupoid, source and target objects need to be accessible for every morphism, and operations on incompatible morphisms should not be defined.

Realising static morphism compatibility checking for this view would normally involve dependent types, at least in the presence of operations like sub-object construction from either a subidentity, or from a subidentity kernel operation as in Definition 6.1. One could also use Haskell type system extensions as implemented in GHC, the currently most widely used compiler, to achieve most of this type safety, but the interface would definitely become less intuitive.

Realising this “arbitrary sets as objects” view in Haskell naturally uses finite subsets of types as objects; we provide this in the SetRel type constructor. This still has to resort to dynamic relation compatibility checking, since now two relations of the same type, for example $r, s :: \text{SetRel} a b$, can now have, for example, different source objects, which are accessible as src $r$ and src $s$, both of type Carrier.Set a, representing finite sets in a way that is compatible with the internal representation of SetRel relations.

This forces programmers either to move all relational computations into an appropriate error monad to deal with possible dynamic “relation type” errors, or to employ the common semigroupoid interface, where the operations provided by the SetRel implementation become partial, with possible run-time failures in the case of morphism incompatibility errors.

The “sets as objects” view has the advantage that the full relation algebra interface becomes available, and, in the BDD-based implementations, an implementation with partial operations can be realised with much lower overhead than the total operations of the “types as objects” view.

8.3. Implementation of concrete relations

The main reason why previously no significant relation library existed for Haskell is, quite likely, that all “obvious” implementation choices inside the language are unsatisfactory.

More space- and time-efficient representations that also can make use of certain regularities in the structure to achieve more compact representations are based on binary decision diagrams (BDDs) [11,40]. Several BDD packages are freely available, but the only known Haskell implementations [9,12] are still rather inefficient and incomplete.

A Haskell binding to the CMU BDD library [32] is used in [21] and is available, while the binding described in [13] does not currently seem to be available.
Even with a Haskell BDD library, or with a complete Haskell binding to an external BDD library, there still would be considerable way to go to implement relation-algebraic operations; we are aware of two BDD-based implementations of relational operations: \texttt{gbdd} \cite{35} is a C++ library providing relational operations using a choice of underlying BDD C libraries, and \texttt{KURE} \cite{33} is the BDD-based kernel library of the \texttt{RelView} system \cite{4,5}; \texttt{KURE} is written in C, and provides many special-purpose functions such as producing element relations between sets and their powersets \cite{31}. and has already been used for a Java binding \cite{19}.

Since C++ is notoriously hard to interface with Haskell, \texttt{KURE} remains as the natural choice for implementing \texttt{Data.Rel} with reasonable effort.

However, it turned out that producing a Haskell binding \texttt{KureRel} to \texttt{KURE} still was a non-trivial task, mainly because of heavily imperative APIs motivated by the graphical user interaction with \texttt{RelView}. In addition, \texttt{RelView} and \texttt{KURE} do not support relations where at least one dimension is zero; we take care of this entirely on the Haskell side.

### 8.4. Semigroupoid classes and instances in \texttt{Data.Rel}

On top of \texttt{KureRel}, we have implemented efficient instances for (the appropriate parts of) the semigroupoid class hierarchy for a number of datatype constructors, including the following:

- **\texttt{NRel}**—Concrete relations between finite ordinals:

  \texttt{NRel} is used for finite relations between the sets \( n \) for \( n \in \mathbb{N} \), where \( 0 = \varnothing \) and \( n + 1 = \{0, \ldots, n\} \). This gives rise to a relation algebra, but since no choice of products or sums is injective, most product- and sum-related classes cannot be implemented. \texttt{NRel} is a simple wrapper around \texttt{KureRel} that is necessary only for typing reasons; conceptually this provides exactly the relation algebra that is used by \texttt{RelView} (which uses a different interface to sums and products).

- **\texttt{CRel}**—Concrete relations between finite sets:

  \texttt{CRel} is used for the relation algebra of finite relations between finite sets. For this, all provided interfaces have been implemented. A \texttt{CRel} is implemented as a triple consisting of two \texttt{Carriers} representing the source and target sets (together with eventual sum, product, or powerset structure), and one \texttt{KureRel} with the dimensions of the two carriers.

  \texttt{SetRel} is a special case of this, where both source and target are plain set carriers.

  The \texttt{CRel} interface implements the “sets as objects” view, and therefore does not generate adaptation functions; this essentially allows one to sacrifice type safety for speed.

- **\texttt{TypeRel}**—Finite relations between types:

  \texttt{TypeRel} is used for the Dedekind semigroupoid of finite relations between Haskell types. \texttt{Carriers} provide support for choices of sum and product, and \texttt{Rel} is the special case of \texttt{TypeRel} for unstructured carriers.

  The implementation of \texttt{TypeRel} is just a wrapper around \texttt{CRel}, and the implementations of the relational operations automatically generate adaptation injections as necessary.

  Besides these very efficient relation datatypes, we also provide some instances implemented purely in Haskell, for very different reasons and also intended for very different purposes:

- **\texttt{FMS}**—Finite set-valued functions between types:

  This is a naïve implementation of relations as set-valued functions; it is provided mainly as a reference implementation, and for demonstration on platforms where the BDD library used by \texttt{KURE} is not available.

- **\texttt{FinMap}**—Finite partial functions between types:

  \texttt{FinMap} provides a semigroupoid view on finite partial functions between Haskell types, which form a lower semilattice semigroupoid with domain, range, zero-morphisms, pseudo-complements, and a large part of the product and sum interface. \texttt{FinMap} uses \texttt{Data.Map.Map} for its implementation, but provides a differently flavoured interface: \texttt{Data.Map} does not even provide function composition.

- **\texttt{FinInjMap}**—Finite partial injections between types:

  A \texttt{FinInjMap} essentially combines two \texttt{FinMaps}, where one is the converse (and therefore inverse) of the other. This implements the groupoid of finite and injective partial functions, and therefore has instances for the semi-allegory and division semi-allegory classes, too.

  Its inclusion was motivated by actual practical applications where the algebraic view allowed a much higher-level programming style than the direct use of \texttt{Data.Map}.
8.5. Visualisation

Our library can be used interactively from the Haskell interpreter GHCi, which provides a very flexible environment for experimentation.

For example, using a small utility function `classGraph` written using the Haskell syntax datatypes and parsing functions included with the GHC distribution, we can extract the subclass relation for the semigroupoid classes of our library by passing its relevant source file location, and then find out about the type of the produced relation, its numbers of nodes and edges (showing that the whole library currently exports 85 classes), and, just as an example, display those edges that are the only incoming edges at their target and the only outgoing edges at their source, once producing a RelView-style bit matrix drawing, and once using dot to layout the produced subgraph; finally use a 3D graph layout algorithm to present relation `g1` in an OpenGL viewing window:

```haskell
> cg <- classGraph "Data/Rel/Classes.lhs"
> :t cg
cg :: SetRel HsId HsId
> Carrier.size $ source cg
85
> length $ Data.Rel.toList cg
152
> gv $ tighten $ injectivePart cg &&& univalentPart cg
> gl' pinkStyle g1
```

As a more complex example, we calculate the subgraph reachable from the class `DedekindSemigroupoid`, which is just small enough to be shown here—since the class graph is produced as a `SetRel`, we convert it to a `Rel` to make subsequent calculations easier to formulate:

```haskell
> let start = hsId "DedekindSemigroupoid"
> let cg' = TypeRel cg
> let d = ran $ endoFromList [(start, start)] *** (transClos cg' ||| dom cg')
> dot $ tighten $ d *** cg'
```

![Diagram of class relationships](image-url)
9. Conclusion

Starting from the insight that, with relations as data, the usual model is one of finite relations between both finite and infinite types, we showed that a hierarchy of relational theories based on semigroupoids instead of categories still captures essentially all expressivity of relation-algebraic formalisations at only minimal cost of working around the absence of identities. We believe that, in this context, our axiomatisations of subidentities, restricted residuals, restricted top elements, transitive closure, and direct products are interesting contributions.

We used this hierarchy of theories to guide the design of a collection of Haskell type class interfaces, and provided implementations both for the rather intuitive “types as objects” view, where we have to live with the absence of identities, and for a “finite sets as objects” view, where we have the full theory and interface of relation algebras at our disposal. These implementations of concrete relations are based on the efficient BDD routines of the RelView kernel library KURE.

To Haskell programmers, this offers a standard data type for finite relations that had been sorely missing, with an implementation that is so efficient that for many uses it will now be perfectly feasible to just write down a point-free relation-algebraic formulation, without spending any effort on selecting or developing a non-point-free algorithm which usually would be much less perspicuous. Even for hard problems this can be a viable method; Berghammer and Milanese describe how to implement a direct SAT solver in RelView, and report that this performs quite competitively for satisfiable problems [6]. It is straightforward to translate such RelView algorithms into Haskell using our library; this essentially preserves performance, and in many cases also adds type safety.

To those interested in programming with relations, we offer an interface to the state-of-the-art BDD-based relation-algebraic toolkit KURE in the state-of-the-art pure functional programming language Haskell. In comparison with for example the imperative special-purpose programming language of RelView, this has obvious advantages in flexibility, interoperability, and accessibility. Especially those who are mathematically inclined will feel more at home in Haskell than in the RelView programming language or in C or Java, which are the other alternatives for access to KURE.

Acknowledgments

I am grateful to Aws Albarghouti, Shiqi Cao, and Scott West for their work on different aspects of the implementation of Data.Rel, supported by NSERC Undergraduate Summer Research Assistantships McMaster Workstudy positions, and McMaster Engineering UROP positions, and to NSERC for additional funding.

References