Relational Heterogeneity Relaxed by Subtyping

Jaap van der Woude[1] and Stef Joosten[1,2]

[1] Open Universiteit Nederland,
Postbus 2960, 6401 DL HEERLEN
[2] Ordina NV, Nieuwegein
{jaap.vanderwoude,stef.joosten}@ou.nl

Abstract. Homogeneous relation algebra is an elegant calculational framework with many applications in computing science. In one application of relation algebra, called Ampersand, heterogeneous relation algebra is used as a specification language for business processes and information systems. For this purpose a typed version of relation algebra is needed together with subtyping. This requires heterogeneous relational algebra. However, the partiality of the composition and union operators in heterogeneous relational algebra are detrimental to its manipulative power. This paper proposes a practical solution to this problem. The authors suggest to relax the partiality of the heterogeneous operators. By suitable choices this homogenisation allows for a type-based specification language, which has sufficient manipulative power.

1 Introduction

A wealth of theory has been developed for binary relations (Tarski ([9]), Schmidt and Ströhlein ([7]), Freyd and Scedrov ([4]), Backhouse et al ([1]), Brink et al ([3]), Maddux ([5]), etc), both in the homogeneous and heterogeneous versions. The theory is interesting in itself, but it has lots of nice applications too. In the current story we are concerned with the use of relations in the realm of requirements and specification. In particular we want to use the manipulative power of relation algebra to construct information systems that support business processes.

This paper is a companion to ([8]) that to a large extent introduces and motivates a method called Ampersand. Ampersand is intended for designing business processes and information systems. It features tools that generate data models and other design artifacts for constructing information systems. Ampersand derives (generates) them from a collection of business rules. The tool is used to specify real life systems and has been used successfully in industry as well as in education. Requirements engineers express business rules in a heterogeneous relation algebra. Thus, Ampersand defines business process(es) by constraints. The heterogeneous version of relation algebra provides users with a type system, which generates feedback on flawed scripts. Recently, Ampersand has been enriched with generalization, which is not discussed in ([8]). In information systems, generalization is a standard feature. It is known under various names like

H. de Swart (Ed.): RAMICS 2011, LNCS 6663, pp. 347–361, 2011.

specialization, sub- or supertyping. It manifests itself in information models via ISA relations and inheritance. We shall freely use the terms generalization and sub- or supertyping as synonyms throughout this paper. The combination of heterogeneous relation algebra with generalization has been put to good use in an improved (provisional) Ampersand compiler for ease of construction, editability and generating feedback. This paper introduces that combination and studies its consequences for Tarski's axioms. It turns out that the resulting language has the same "homogeneous" three layer structure and satisfies similar axioms.

Ampersand looks at data in an information system as a set of binary relations that "store" the data. The data is organized by a structure that resembles an ontology; it is built up of *concepts*, *relations* between concepts and *rules* governing the relations. A graph, in which concepts are represented by vertices and relations by edges, is used in Ampersand as a conceptual model. Concepts may range from concrete entities like names and numbers to complex entities like orders and abstractions like delivery-appointments. Relations vary from deterministic (functions) to partial multivalued ones. As data changes over time, the contents of these relations do too. The representation of the data in the relations in Ampersand is governed by rules that form a set of invariants. Every now and then the rules may be violated by actions that change the actual content of relations. Upon violation, the system must be brought back in a state where the invariants are satisfied.

In Ampersand every relation is given a type at define time, as in $R : A \sim B$. This declaration implies that there exist concepts A and B, and every tuple $\langle a, b\rangle \in R$ implies $a \in A$ and $b \in B$. This interpretation is specific for Ampersand and is motivated in [8]. The user can formulate rules in terms of declared relations only. Ampersand features a compiler, which deduces a computer program from these rules. The resulting program maintains a database and keeps all rules satisfied in that database. Note that the intended use is in real information systems, which implies finiteness. The possible contents of the system may be infinite, but the number of concepts and the number of basic relations in the ontology certainly will be finite.

The feedback system of the Ampersand compiler ([8]) facilitates adaptation of relations and rules. The user is signaled in cases of predictable errors, mainly based on type information. If the user provides a formula the compiler cannot assign a unique type to, there might be a mistake in the formula or a choice between overloaded term names should be made. Well-typedness arguments are straightforward in a fully typed heterogeneous relation algebra, but how should we proceed in case supertyping is allowed. E.g. let $R : A \sim B$ and $S : C \sim D$ where C is a supertype of B. The composition $R \fatsemi S$ might make sense to the user, but it is signaled incorrect by the typing mechanism. Relaxing the stringent typing convention raises a question about the composition $(\neg R) \fatsemi S$. Do we consider the complement with respect to the type B we get from R or should we use its supertype C resulting from S?

1.1 Goal

For the purpose of generalization (subtyping or isa-constructions), the author of an Ampersand script can specify an order between two different concepts. We denote that order by $C \preceq D$ and say that C is a subtype of D or D is a supertype of C. On the element level one may say a C-member *is a* D-member. Users of Ampersand may (sloppily) refer to concepts as types. The concepts are assumed to form a lattice with some appropriate special properties depending on the problem domain. In fact a join semilattice would suffice, but assuming finiteness we get the complete lattice structure for free. If a user does not specify any order, the resulting lattice of concepts is flat.

The lattice has extreme elements representing extreme concepts: the empty ($\bot$) and the universal one ($\top$). Ampersand calls them *nothing* and *anything*. These concepts are the ones that we try to avoid in specifying the system: nothing doesn't need rules and anything doesn't allow them within reason. Ampersand's type checker rejects any relation term that has $\bot$ or $\top$ as its type.
Although concepts may be seen as sets of elementary things (also known as atoms in Ampersand), we should not necessarily interpret their meet and join setwise. In the current research version of Ampersand with editing possibilities, for example, the meet of two concepts is the more specific of the two if the concepts are order related, but nothing if not. Another example may be the join of vector spaces. In general we assume the (join semi-)lattice structure with extremities and ignore the interpretation and specialties of the concepts until the application section where choices of the concepts play a role.

Rules are constructed using the relations occurring in the ontology and they may be transformed by way of the manipulative possibilities in relation algebra. Users of Ampersand are meant to get help from relation calculus in formulating and rewriting those rules. For this reason, we need to know which of Tarski's axioms are available as transformation rules to users. Because of the typing with concepts, the homogeneous relation calculus is only locally applicable. The heterogeneous relation calculus, however, is too restrictive in its definedness of the relevant operators. We opt for an intermediate form of relation calculus that allows for subtyping, which has much of the calculational power of the homogeneous calculus and still has the typing advantages of heterogeneous relation calculus.

1.2 Result

In this paper we propose such an intermediate relational calculus. Starting of with a heterogeneous relation calculus we use generalization in order to view the lattice and composition operators on relations as being totally defined. To that end we make the subtyping explicit by introducing embeddings ϵ as relations and use them in composition with the non-compatibly typed relations to turn them into composable or addable relations. The typed lattice and composition structure is thus slightly generalized and the reverse structure is unaltered. From that point on, all we need to do is walk through Tarski's axioms using the new total operations and check their validity.

It turns out that with this construction the homogeneous relations with the ten Tarski axioms ([5], page 21) as manipulative power can be mimicked in the heterogeneous relations, preserving almost everything except for negation and the Schröder rules. But there is an acceptable way out of that omission: division and the Dedekind rule.

For the Ampersand user, this has an implication in the use of the negation. Instead of the unary complement he gets a binary subtraction operator that enables Ampersand to infer a type specialization if necessary. By way of syntactic sugar, he may omit the left hand argument, yielding the 'look and feel' of the complement operator. The Ampersand compiler facilitates this by adding a default value for the missing argument.

The Ampersand tool provides assistance in constructing relational terms by means of a type checker. In the perception of users, the type system discriminates meaningful expressions from meaningless ones. A meaningful expression in Ampersand is a relation term, which has a unique type other than $\bot$ or $\top$. Meaningless expressions are rejected by the type checker with an informative diagnostic message. This choice for meaningless terms in the augmented Ampersand tool is captured in the concept lattice and is thus separated from the relational calculus. We consider two concepts to be unrelated if their join equals $\top$, the universe of things without discriminating properties. Composition and gathering unrelated things is considered meaningless.

2 Definitions

First we introduce the usual system of heterogeneous relations and fix the notation we use. (Such may be done via order enriched categories or via allegories (e.g. [4]) but mastery of those fundamentals of mathematics is not necessary to read this paper.) After that we propose the changes to facilitate the sub- and supertyping.

2.1 Heterogeneous Relation Calculus

We consider the category $(\mathcal{C}, \mathcal{R}el)$ of relations on the concept set $\mathcal{C}$ [1]. Following the relational habit we shall denote the categorical and the relational composition by a semicolon $\fatsemi$ with high precedence. The sets of morphisms $\mathcal{R}el(A, B)$, i.e. the collections of all relations between the concepts A and B form complete, chain distributive lattices and these lattices are interrelated by way of the composition $\fatsemi$ and the converse $^\cup$, as follows:

$$\fatsemi : \mathcal{R}el(A, B) \times \mathcal{R}el(B, C) \longrightarrow \mathcal{R}el(A, C)$$

has identities as units, is associative and universally disjunctive

$$^\cup : \mathcal{R}el(A, B) \longrightarrow \mathcal{R}el(B, A)$$

is an isomorphic involution (i.e. $^\cup \circ {}^\cup = id$)

[1] If you are not familiar with the notion of category, think of a transitive directed graph whose vertices are concepts and whose edges are relations between the vertices they connect. An edge is termed morphism or relation. Pasting edges is called composition.

and they satisfy the converse-composition layer interface of contravariant distribution

$$(R \fatsemi S)^{\cup} = S^{\cup} \fatsemi R^{\cup}$$

The homsets $\mathcal{R}el(A, B)$ have a top $\top_{A,B}$, the universal relation, and a bottom $\bot_{A,B}$, the empty relation. Where the context prevents ambiguous interpretation, we shall omit subscripts.
Instead of $R \in \mathcal{R}el(A, B)$ we prefer to write $R : A \sim B$ and pronounce R is a relation between A and B (which by listening from left to right differs from being a relation between B and A).
We denote the left and right types of R by $R^{\triangleleft}$ (= A above) and $R^{\triangleright}$ (= B above) respectively.

The meet and join are written as $\sqcap$ and $\sqcup$ and they have the usual properties except for arbitrary distributivity. Considering the set-theoretic interpretation of relations in Ampersand, it is no coincidence that they look somewhat like set-theoretic intersection and union.
Often the morphisms are even assumed to form complete, complemented lattices. Moreover, the complement is used to formulate an interface between the three layers of the relation calculus: the so called Schröder rule (but it has many names and shapes)

$$\text{equivalent are: } R \fatsemi S \sqsubseteq \neg T^{\cup}, \quad T \fatsemi R \sqsubseteq \neg S^{\cup} \text{ and } S \fatsemi T \sqsubseteq \neg R^{\cup}$$

that connects the order, the composition and the converse. Although complementation might work well in the homogeneous case and even in the localized heterogeneous version, we do not consider it because we will run into trouble because of the intended additional subtyping and the consequential uncertainty of the complementation domain. So we should cope without negation or at least without negation in considering relations with different typing. Instead of negation we may use division (or factors) to be introduced shortly.

We replace the Schröder rule by the three-layer interface known as the modular identity ([4]) or the Dedekind rule ([6]) that partly makes up for the lack of conjunctivity of $\fatsemi$ as follows:

$$R \fatsemi S \sqcap T \sqsubseteq (R \sqcap T \fatsemi S^{\cup}) \fatsemi (S \sqcap R^{\cup} \fatsemi T)$$

In the presence of complementation the Dedekind rule is equivalent to the Schröder rule.

An interesting property of many relational systems, not captured by the ten Tarski axioms, is the universal disjunctivity of the composition. I.e.

$$(\sqcup U : U \in \mathcal{U} : U) \fatsemi R = (\sqcup U : U \in \mathcal{U} : U \fatsemi R) \quad \text{and}$$
$$R \fatsemi (\sqcup U : U \in \mathcal{U} : U) = (\sqcup U : U \in \mathcal{U} : R \fatsemi U)$$

Note that we use ternary notation for quantifications and repeated operator applications, similar to the Z-notation (with colons as separators in stead of a bar

and a bullet). The first part of the quantification designates the repeated operator and the dummies to be used in the terms (e.g. in the second quantification these are $\sqcup U$), the second part indicates the domain for the dummies ($U \in \mathcal{U}$) and the third part is for the terms to be quantified ($U \fatsemi R$). In section **3.3** more examples of this notation occur.

The universal disjunctivity property of relational systems is equivalent to ([2]) the existence of two Galois connections between the left and right compositions and operations that might therefore be called factorisations or divisions (the term that we shall use here):

$$R \sqsupseteq S \fatsemi T \;\equiv\; R /\!\!/ T \sqsupseteq S \qquad \text{and} \qquad S \fatsemi T \sqsubseteq R \;\equiv\; T \sqsubseteq S \backslash\!\backslash R$$

Adjointness of $\fatsemi T$ and $/\!\!/ T$ and of $S\fatsemi$ and $S\backslash\!\backslash$ is (in a certain sense) equivalent to universal distributivity of $\fatsemi T$ and $S\fatsemi$ respectively. The name division is illustrated by the next cancellation property:

$$R \sqsupseteq (R /\!\!/ T) \fatsemi T \quad \text{and} \quad S \fatsemi (S \backslash\!\backslash R) \sqsubseteq R$$

These cancellation properties show some typing restrictions in the heterogeneous setting. The composition and the inclusion should make sense, so the typing of the lefthand division is $R /\!\!/ T : R^{\triangleleft} \sim T^{\triangleleft}$ and it only exists if the right types of R and T are the same ($R^{\triangleright} = T^{\triangleright}$).

Division is strongly related to negation. Indeed, from the Schröder rule it follows for instance that $S \backslash\!\backslash R \;=\; \neg\,(S^{\cup} \fatsemi \neg R)$, and thus that $\neg\; S = S^{\cup} \backslash\!\backslash \neg I$. So, considering the reluctant use of the negation in our lattice layer, division might be a useful operation to replace some of the applications of negation. For the application in constructing constraints and business rules the division has the following set-theoretic interpretation:

$$a \langle R /\!\!/ T \rangle b \;\equiv\; (\forall\, c : b\; T\; c : a\; R\; c) \quad \text{and} \quad a \langle S \backslash\!\backslash R \rangle b \;\equiv\; (\forall\, c : c\; S\; a : c\; R\; b)$$

2.2 Adding Supertyping

In information modeling the notion of *ISA* relation captures inheritance of entities. It stands for embedding specialized things as things. Here we coin the term generalization or supertyping for this. Generalization or supertyping may be thought of as losing some but possibly not all information (e.g. a manager is an employee), while subtyping stands for refining or specializing the information (e.g. adding a locked drawer makes a desk more determined). In the standard heterogeneous relation calculus we cannot add the fact that manager *Baas* uses locked-drawer-desk *ldd*481 to the relation of employees using desks because the types do not match. But we do want to do so because *Baas* is an employee and *ldd*481 is a desk. What we can do is embedding subconcepts in concepts (or concepts in superconcepts) and extend that embedding to relations, so employee is a supertype of manager and locked-drawer-desk is a subtype of desk.

Having a relation between subconcepts, the question arises how to extend it to their generalizations? And the other way around: can we restrict relations

between types to subtypes? Instead of the given examples we may also think of clients as special persons, appointments as generalized meetings, invitations of clients to meetings or appointments for persons and the like.

In Ampersand the concepts occurring in the ontology may be sub- or supertyped and we assume that the generalization is structured as a lattice with extremal elements ($\bot$ and $\top$). We consider the generalization as an order on the concepts, given by way of embeddings of subtypes in types. The combination of heterogeneous relation algebra and generalization is substantiated by introducing the embedding functions (ϵ) as heterogeneous relations, thus incorporating the subtyping in the relation algebra. This enables us to extend the composition and the definedness of the lattice operators. As follows:

Consider the lattice $(\mathcal{C}, \preceq)$ of concepts as a subcategory of $(\mathcal{C}, \mathcal{R}el)$ by adding an embedding relation $\epsilon_{A,B}$ for every $A \preceq B$. Clearly, we want

$$\epsilon_{A,A} = id_A \qquad \text{and} \qquad \epsilon_{A,B} \fatsemi \epsilon_{B,C} = \epsilon_{A,C}$$

Moreover, the embeddings should be injective and functional (total and univalent) relations, hence

$$id_A = \epsilon_{A,B} \fatsemi \epsilon^{\cup}_{A,B} \quad \text{and} \quad \epsilon^{\cup}_{A,B} \fatsemi \epsilon_{A,B} \sqsubseteq id_B$$

We assume that the concepts, their members and their relations are such that this is possible.

Following the embedding of concepts, we can now embed the relations (the homsets) too. Assume $A \preceq C$ and $B \preceq D$, then $R : A \sim B$ is embedded as relation between the supertypes C and D by composing it with the appropriate embeddings:

$$\text{if } R : A \sim B \text{ then } \epsilon^{\cup}_{A,C} \fatsemi R \fatsemi \epsilon_{B,D} : C \sim D$$

Indeed, in the left concept only A-members of C are considered via $\epsilon^{\cup}_{A,C}$, while the resulting B-members are properly embedded in D.

Not only can we embed relations in supertypes, we may also restrict relations to subtypes:

$$\text{if } S : C \sim D \text{ then } \epsilon_{A,C} \fatsemi S \fatsemi \epsilon^{\cup}_{B,D} : A \sim B$$

Because of the embedding $\epsilon_{A,C}$ only A-members of C will occur on the left type of S, while the resulting D-members are filtered by $\epsilon^{\cup}_{B,D}$ so that only B-members are accepted.

As expected, restriction after embedding is just the identity, but embedding after restriction stays restrictive. Indeed:

$$\epsilon_{A,C} \fatsemi \epsilon^{\cup}_{A,C} \fatsemi R \fatsemi \epsilon_{B,D} \fatsemi \epsilon^{\cup}_{B,D} \;=\; id_A \fatsemi R \fatsemi id_B \;=\; R$$

while

$$\epsilon^{\cup}_{A,C} \fatsemi \epsilon_{A,C} \fatsemi S \fatsemi \epsilon^{\cup}_{B,D} \fatsemi \epsilon_{B,D} \sqsubseteq id_C \fatsemi S \fatsemi id_D \;=\; S$$

Finally, note that if two concepts, say A and B, have a common supertype, say C, we can consider the commonality within them: $\epsilon_{A,C} \fatsemi \epsilon^{\cup}_{B,C} : A \sim B$. In particular this holds for the join $A \vee B$ instead of C as well as for $\top$ instead of C.

3 Adapting the Operations

In our heterogeneous relation system we incorporated the sub- and supertyping by embeddings between refining concepts, leading to embedding and restriction of relations. The next step is to extend the inter-homset relation operators $\fatsemi$, $^{\cup}$ and the homset lattice operations $\sqcup$ and $\sqcap$, preferably retaining the interfaces between the layers.

3.1 New Composition

In the heterogeneous system, the composition is only defined if the right concept of the first and the left concept of the second argument are equal:

$$\fatsemi : (A \sim B) \times (B \sim C) \longrightarrow (A \sim C)$$

We want to relax the equality "in the middle" so, that composition is also defined in case the middle components are refining. To that end we define the extended composition by

$$\mathbin{⨾} : (A \sim B) \times (C \sim D) \longrightarrow (A \sim D) \quad \text{with}$$

$$R \mathbin{⨾} S \;=\; R \fatsemi \epsilon_{B,B \vee C} \fatsemi \epsilon^{\cup}_{C,B \vee C} \fatsemi S$$

There is nothing new with the new composition (i.e. $R \mathbin{⨾} S \;=\; R \fatsemi S$) in case the right concept of R coincides with the left concept of S (i.e. $B = C$ in the above), so $\mathbin{⨾}$ is a total extension of $\fatsemi$.

The identity is also the unit of the new composition, but even more is true: embedding and restriction are just new compositions with the appropriate identities. For the embedding (assuming $R \in A \sim B$, $A \preceq C$, $B \preceq D$):

$$\epsilon^{\cup}_{A,C} \fatsemi R \fatsemi \epsilon_{B,D} \;=\; id_C \fatsemi \epsilon_{C,C} \fatsemi \epsilon^{\cup}_{A,C} \fatsemi R \fatsemi \epsilon_{B,D} \fatsemi \epsilon^{\cup}_{D,D} \fatsemi id_D \;=\; id_C \mathbin{⨾} R \mathbin{⨾} id_D$$

and, similarly for the restriction (assuming $S \in C \sim D$, $A \preceq C$, $B \preceq D$):

$$\epsilon_{A,C} \fatsemi S \fatsemi \epsilon^{\cup}_{B,D} \;=\; id_A \fatsemi \epsilon_{A,C} \fatsemi \epsilon^{\cup}_{C,C} \fatsemi S \fatsemi \epsilon_{D,D} \fatsemi \epsilon^{\cup}_{B,D} \fatsemi id_B \;=\; id_A \mathbin{⨾} S \mathbin{⨾} id_B$$

3.2 New Order

The lattice operations are only defined per homset. If we want to join two relations with different typings, we should look for a common supertype embed the relations accordingly and join them there. Similarly we find the common workspace for the meet of two relations of different types. We have to pay the

price that we do lose some information (and we'll never get that back) but the relations can be joined where originally heterogeneity wouldn't allow it. So define $\textcircled{\sqcup}$ and $\textcircled{\sqcap}$ by

$$\textcircled{\sqcup}, \textcircled{\sqcap} \ : \ (A \sim B) \times (C \sim D) \longrightarrow (A \vee C \ \sim \ B \vee D) \quad \text{with}$$

$$\begin{aligned} R \textcircled{\sqcup} S &= \epsilon^{\cup}_{A,A\vee C} \fatsemi R \fatsemi \epsilon_{B,B\vee D} \ \sqcup \ \epsilon^{\cup}_{C,A\vee C} \fatsemi S \fatsemi \epsilon_{D,B\vee D} \\ &= id_{A\vee C} \textcircled{\fatsemi} R \textcircled{\fatsemi} id_{B\vee D} \ \sqcup \ id_{A\vee C} \textcircled{\fatsemi} S \textcircled{\fatsemi} id_{B\vee D} \end{aligned}$$

$$\begin{aligned} R \textcircled{\sqcap} S &= \epsilon^{\cup}_{A,A\vee C} \fatsemi R \fatsemi \epsilon_{B,B\vee D} \ \sqcap \ \epsilon^{\cup}_{C,A\vee C} \fatsemi S \fatsemi \epsilon_{D,B\vee D} \\ &= id_{A\vee C} \textcircled{\fatsemi} R \textcircled{\fatsemi} id_{B\vee D} \ \sqcap \ id_{A\vee C} \textcircled{\fatsemi} S \textcircled{\fatsemi} id_{B\vee D} \end{aligned}$$

The new lattice operations $\textcircled{\sqcup}$ and $\textcircled{\sqcap}$ are total extensions of the old ones $\sqcup$ and $\sqcap$. It is readily seen that $\textcircled{\sqcup}$ and $\textcircled{\sqcap}$ are symmetric and associative and that they distribute. Zeros do exist for both $\textcircled{\sqcup}$ and $\textcircled{\sqcap}$, viz. $\top\!\!\top : \top \sim \top$ and $\bot\!\!\bot : \top \sim \top$, but only $\textcircled{\sqcup}$ allows for a unit: $\bot\!\!\bot : \bot \sim \bot$. Therefore we don't have to hope too much for a suitable definition of a (new) negation.

Some doubt arises because of this anomaly, and quite rightly so. We can define order extensions of $\sqsubseteq$ such that $\textcircled{\sqcup}$ or $\textcircled{\sqcap}$ are the join or meet, but not both. Let's define for $R : A \sim B$ and $S : C \sim D$:

$$R \textcircled{\sqsubseteq} S \ \equiv \ A \preceq C \wedge B \preceq D \wedge \ \epsilon^{\cup}_{A,C} \fatsemi R \fatsemi \epsilon_{B,D} \sqsubseteq S$$

Then $\textcircled{\sqcup}$ is the disjunction (join) for $\textcircled{\sqsubseteq}$ and thus we have

$$R \textcircled{\sqsubseteq} S \ \equiv \ R \textcircled{\sqcup} S = S$$

But be aware that with this choice $\textcircled{\sqcap}$ is only locally the meet for $\textcircled{\sqsubseteq}$. Because all unions exist, the global meet for $\textcircled{\sqsubseteq}$ does exist, but it is not an operation that is useful for our purposes. The meet would require restriction of the left and right types, but domain restriction is not a reasonable operation on relations as it suggests gain of knowledge or structure where we can only lose it by generalization. We had to forget about the specialties before we were able to intersect, so the price of this totalization is that those specialties are lost.
In an Ampersand script, the user need not worry about the supertype level. He uses $\textcircled{\sqcap}$ as his meet operator, which is analyzed by the type checker. After checking and inferring the correct types, Ampersand works exclusively locally, so the almost-meet indeed is equivalent to $\sqcap$ where it occurs.

Note that the local complements cannot be extended to a global complement in general. Even globally complementing a local top for a nontrivial concept will not succeed. In general the notion of complement doesn't make sense in an environment with generalization, because different supertypings destroy the base of complementation. Instead we still may consider the typed complements like $\top\!\!\top_{A,B} \setminus R$ for relation $R : A \sim B$. This insight has influenced the implementation of Ampersand. As of the next version, the complement operator is substituted by a binary minus sign. If the left hand argument is missing, the compiler assumes $\top\!\!\top_{A,B}$ in which $A \sim B$ is the type of the right hand argument. The type checker deduces the type and generates an error message if it is not uniquely defined.

3.3 Interfaces

The converse doesn't need adaptation since the definedness doesn't change.

The reader may want to verify that the interface between the converse and the composition layers smoothly carries over to the converse and the new composition:

$$(R \mathbin{⨾} S)^{\cup} \;=\; S^{\cup} \mathbin{⨾} R^{\cup}$$

The interface between the composition and the lattice layer, i.e. the universal disjunctivity of the composition, is also valid for the new versions of composition and disjunction. To show that, let $\mathcal{U}$ be a collection of relations, say $U : U^{\triangleleft} \sim U^{\triangleright}$ for $U \in \mathcal{U}$. Let A and B be the joins of their left and right domain concepts respectively and let $R : C \sim D$ then

$$\begin{array}{ll}
 & (\bigsqcap \mathcal{U}) \mathbin{⨾} R \\
= & \quad \{ \text{ definition } \bigsqcap \ \} \\
 & (\sqcup U : U \in \mathcal{U} : \epsilon^{\cup}_{U^{\triangleleft},A} \mathbin{⨾} U \mathbin{⨾} \epsilon_{U^{\triangleright},B}) \mathbin{⨾} R \\
= & \quad \{ \text{ definition } ⨾ \ \} \\
 & (\sqcup U : U \in \mathcal{U} : \epsilon^{\cup}_{U^{\triangleleft},A} \mathbin{⨾} U \mathbin{⨾} \epsilon_{U^{\triangleright},B}) \mathbin{⨾} \epsilon_{B,B \vee C} \mathbin{⨾} \epsilon^{\cup}_{C,B \vee C} \mathbin{⨾} R \\
= & \quad \{ \text{ universal distributivity of } ⨾ \ \} \\
 & (\sqcup U : U \in \mathcal{U} : \epsilon^{\cup}_{U^{\triangleleft},A} \mathbin{⨾} U \mathbin{⨾} \epsilon_{U^{\triangleright},B} \mathbin{⨾} \epsilon_{B,B \vee C} \mathbin{⨾} \epsilon^{\cup}_{C,B \vee C} \mathbin{⨾} R) \\
= & \quad \{ \text{ transitivity of embeddings } \ \} \\
 & (\sqcup U : U \in \mathcal{U} : \epsilon^{\cup}_{U^{\triangleleft},A} \mathbin{⨾} U \mathbin{⨾} \epsilon_{U^{\triangleright},B \vee C} \mathbin{⨾} \epsilon^{\cup}_{C,B \vee C} \mathbin{⨾} R) \\
= & \quad \{ \text{ exercise: } X \vee Z \sqsubseteq Y \;\Rightarrow\; \epsilon_{X,X \vee Z} \mathbin{⨾} \epsilon^{\cup}_{Z,X \vee Z} = \epsilon_{X,Y} \mathbin{⨾} \epsilon^{\cup}_{Z,Y} \ \} \\
 & (\sqcup U : U \in \mathcal{U} : \epsilon^{\cup}_{U^{\triangleleft},A} \mathbin{⨾} U \mathbin{⨾} \epsilon_{U^{\triangleright}, U^{\triangleright} \vee C} \mathbin{⨾} \epsilon^{\cup}_{C, U^{\triangleright} \vee C} \mathbin{⨾} R) \\
= & \quad \{ \text{ definition } ⨾ \ \} \\
 & (\sqcup U : U \in \mathcal{U} : \epsilon^{\cup}_{U^{\triangleleft},A} \mathbin{⨾} (U \mathbin{⨾} R)) \\
= & \quad \{ \text{ definition } \bigsqcap \ \} \\
 & (\bigsqcap U : U \in \mathcal{U} : U \mathbin{⨾} R)
\end{array}$$

Universal $\bigsqcap$-junctivity of $⨾$ guarantees the existence of adjoint division operators, say $/\!\!\oslash$ and $\backslash\!\!\oslash$, given by

$$R \sqsupseteq S \mathbin{⨾} T \;\equiv\; R /\!\!\oslash T \sqsupseteq S \quad \text{and} \quad S \mathbin{⨾} T \sqsubseteq R \;\equiv\; T \sqsubseteq S \backslash\!\!\oslash R$$

The typing of the new factors should follow from their construction, e.g.

$$R /\!\!\oslash T \;=\; (\bigsqcap U : R \sqsupseteq U \mathbin{⨾} T : U)$$

leading to the typing $R /\!\!\oslash T : R^{\triangleleft} \sim \top$, provided $T^{\triangleright} \preceq R^{\triangleright}$, otherwise $R /\!\!\oslash T = \bot_{\bot,\bot}$. Remember that $R /\!\!/ T$ needed $T^{\triangleright} = R^{\triangleright}$, so the new factor is slightly more

defined, with a (too) big right domain concept, but otherwise it is the same. Indeed, here we also have

$$a\langle R \oslash T\rangle b \equiv (\forall\, c : b\ T\ c : a\ R\ c) \quad \text{and} \quad a\langle S \oslash R\rangle b \equiv (\forall\, c : c\ S\ a : c\ R\ b)$$

The interface between the converse and the lattice (order) layers carries over smoothly to the converse and the new lattice(like) operations:

$$(R \underline{\cup} S)^{\cup} = R^{\cup} \underline{\cup} S^{\cup} \quad \text{and} \quad (R \underline{\cap} S)^{\cup} = R^{\cup} \underline{\cap} S^{\cup}$$

After introducing the total versions of the composition and the disjunction (order) we harvested the majority of the Tarski rules:

- We do have a suitably rich order structure, be it that we have an operation that in several respects looks like the meet but it isn't. It does have a reasonable interpretation and sufficient manipulative possibilities to keep it (more evidence below).
- The extended composition structure still forms a monoid
- The converse didn't even change.
- All interfaces between the layers are preserved except for Schröders rule that doesn't make sense without complementation.

The interface between all three layers is still lacking. Our caution with respect to the negation directed our interest towards the modular law ([4]). An inconvenience for the asymmetric modular law is the unfortunate typing, but this may be corrected by choosing the slightly more complicated but symmetric version called the Dedekind rule ([6]). (We gratefully thank the referee for suggesting the symmetric version and pointing at the typing advantage.) A severe drawback is the fact that the meet-like operator occurring in this new Dedekind rule is only locally the meet. So, globally, the Dedekind rule does not capture the boundary for conjunctivity as it did for the original relational calculus. Nevertheless we are happy with it because it still gives the manipulative power comparable to the original rule and it serves a purpose in mimicking the Tarski-rules for our new operations in the heterogeneous setting with generalization. Indeed, the Ampersand user, though working in a heterogeneous setting, is allowed to transform his expressions with this rule as if his environment is homogeneous, leaving the reponsibility for the syntactic correctness to the type-checker.

So we will show that

$$R \mathbin{⨟} S \mathbin{\underline{\cap}} T \ \underline{\subseteq}\ (R \mathbin{\underline{\cap}} T \mathbin{⨟} S^{\cup}) \mathbin{⨟} (S \mathbin{\underline{\cap}} R^{\cup} \mathbin{⨟} T)$$

Let $R : A \sim B$, $S : P \sim Q$ and $T : X \sim Y$, then the two composed relations on the righthand side are

$$\epsilon^{\cup}_{A,A\vee X} \mathbin{⨟} R \mathbin{⨟} \epsilon_{B,B\vee P} \sqcap \epsilon^{\cup}_{X,A\vee X} \mathbin{⨟} T \mathbin{⨟} \epsilon_{Y,Q\vee Y} \mathbin{⨟} \epsilon^{\cup}_{Q,Q\vee Y} \mathbin{⨟} S^{\cup} \mathbin{⨟} \epsilon_{P,B\vee P}$$

and

$$\epsilon^{\cup}_{P,B\vee P} \mathbin{⨟} S \mathbin{⨟} \epsilon_{Q,Q\vee Y} \sqcap \epsilon^{\cup}_{B,B\vee P} \mathbin{⨟} R^{\cup} \mathbin{⨟} \epsilon_{A,A\vee X} \mathbin{⨟} \epsilon^{\cup}_{X,A\vee X} \mathbin{⨟} T \mathbin{⨟} \epsilon_{Y,Q\vee Y}$$

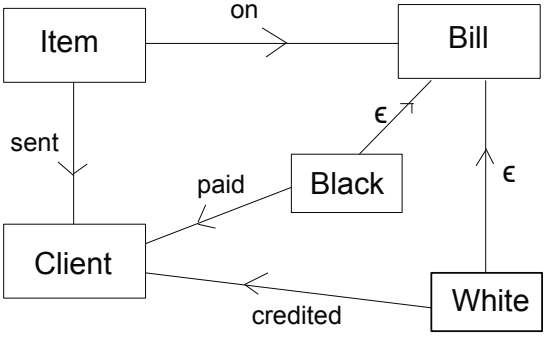
The supertyped relations $\epsilon^{\cup} \fatsemi U \fatsemi \epsilon$ occur in these expressions also in conversed version with the same embeddings, so we may abbreviate them as R', S', T' and their converses in the calculation below

$$
\begin{array}{ll}
 & (R \sqcap T \fatsemi S^{\cup}) \fatsemi (S \sqcap R^{\cup} \fatsemi T) \\
= & \quad \{ \text{ above with abbreviation } \} \\
 & (R' \sqcap T' \fatsemi S'^{\cup}) \fatsemi (S' \sqcap R'^{\cup} \fatsemi T') \\
= & \quad \{ \text{ intermediate types are the same } (B \vee P) \text{, so the composition is local } \} \\
 & (R' \sqcap T' \fatsemi S'^{\cup}) \fatsemi (S' \sqcap R'^{\cup} \fatsemi T') \\
\sqsupseteq & \quad \{ \text{ Dedekind in the original relation calculus } \} \\
 & R' \fatsemi S' \sqcap T \\
= & \quad \{ \text{ unfold abbreviation } \} \\
 & \epsilon^{\cup}_{A,A\vee X} \fatsemi R \fatsemi \epsilon_{B,B\vee P} \fatsemi \epsilon^{\cup}_{P,B\vee P} \fatsemi S \fatsemi \epsilon_{Q,Q\vee Y} \sqcap \epsilon^{\cup}_{X,A\vee X} \fatsemi T \fatsemi \epsilon_{Y,Q\vee Y} \\
= & \quad \{ \text{ new operators } \} \\
 & R \fatsemi S \sqcap T
\end{array}
$$

This concludes the rendering of almost all Tarski axioms from the homogeneous relations in our adaptation of the heterogeneous relations.

3.4 Example

A very small example to illustrate some of the above notions is about business rules for a company that sells ordered items to clients and bills them either by crediting them or, to evade taxes, by direct payment. So he needs two kinds of bills modeled by subtypes, the black bills and the white ones. The mini-ontology is:



The following two business rules show the generalized composition and union and show the use of division that cannot completely do without type restriction.

– An item on a bill that is paid for or credited is sent to the client.

$$\mathsf{on} \fatsemi (\mathsf{paid} \sqcup \mathsf{credited}) \sqsubseteq \mathsf{sent}$$

– A black bill must be paid by the client when all items on it are sent to him. For a translation we reformulate it into predicate calculus and transform it to a rule in terms of relations: for arbitrary bill β and client γ

$$
\begin{array}{ll}
 & (\forall\, i : i\langle \mathsf{on}\rangle\beta : i\langle \mathsf{sent}\rangle\gamma) \;\Rightarrow\; \beta\langle \mathsf{paid}\rangle\gamma \\
\equiv & \quad \{\ \text{division}\ \} \\
 & \beta\langle \mathsf{on}\backslash\!\backslash\mathsf{sent}\rangle\gamma \;\Rightarrow\; \beta\langle \mathsf{paid}\rangle\gamma \\
\approx & \quad \{\ \text{forall bills and clients}\ \} \\
 & \mathsf{on}\backslash\!\backslash\mathsf{sent} \sqsubseteq \mathsf{paid} \\
\approx & \quad \{\ \text{don't mind the subtypes, but for a domain of black bills}\ \} \\
 & id_{\mathsf{Black}} \,⨾\, (\mathsf{on} \,\overline{⨾}\, \mathsf{sent}) \sqsubseteq \mathsf{paid}
\end{array}
$$

4 Application in Ampersand

Ampersand is a tool for constructing an information system from formally given business rules. These rules are defined as relation terms over a finite set of relations. Rules and relations are introduced (declared) by the user in a suitable syntax. The system assists its users (business rule engineers as well as students) in constructing relevant relational terms and rules. To that end a syntactic calculus of relations based on the ten Tarski axioms is enriched with typing arguments to filter out the evidently meaningless terms. The totalization of the heterogeneous relations above represents the first stage in constructing Ampersand relation terms. Relation terms may be composed, joined, intersected and reversed without bothering too much about the definedness of the operators. The necessity of that carelessness stems from the polymorphic naming as well as from the use of sub- and supertypes (overloading of relations and atoms, see [8] section 3.3). E.g. if some director is combined with a specific desk it should not primarily concern us whether he uses his desk as a manager, an employee or as a person, we may not want to make that typing distinction when constructing our desk-usage rule now. (Although it may be relevant later if the quality of the drawer locks is at stake.) So we may discuss many relations all called "uses" between every subtype of "person" and every subtype of "desk". The totality of the new operators allows that carelessness.

A user of Ampersand experiences that he needs not worry about types until the type checker detects an ambiguity (or omission). The language contains a mechanism by which the user can add type information to relation terms, in order to disambiguate his expressions and satisfy the type checker. Since Ampersand will not do anything until the script of the user is correct, (s)he experiences the semantics of the operations in the second stage only.

The second stage starts with the introduction of a suitable concept lattice, the generalizations in which are declared initially. We assume that that two concepts are completely independent if they have $\top$ as join. We don't have a relevant supertype of numbers and employees, of currencies and blood pressures, of amino acids and scientific journals. This is used in judging parts of relational terms:

- A composition with coarsest intermediate concept equal to $\top$ is certainly empty, since the glued left and right types are completely independent.
- A disjunction of relations with one of the domain concepts $\top$ combines independent things and is considered to be undesirable, we don't mix unmixable atoms.

The types of all domain concepts are calculated by the type checker and the instances of the above anomalies are detected and used to inform the user of probable mistakes. The precision of catching mistakes depends heavily on the strength of the concept lattice as a discriminator of (ir)relevancies. But it may give hints even if independence is not equivalent to joining up to $\top$.
The Ampersand user, or the information architect in the preceding modeling process, may define the dependencies and independencies in the subtyping order corresponding to her own needs. E.g. in recent provisional Ampersand applications the choice for trivial concept conjunctions was made, in older ones the flat lattice resulted. It is up to the user.

In the current Ampersand version negation poses a problem. If we have a relation in ordinary and supertype version, what should we do with the negation? Are we dealing with the complement in the ordinary type or does the complement in the supertype play a role. This is especially relevant in case the relation is composed with terms related to the supertype. In fact, that problem was the incentive for the investigations in this paper. The situation can now be significantly improved by adopting division while replacing the Schröder rules by the Dedekind rule and suitably restricting the $\top$-side of the division.
Negation is undoubtedly a natural element in specification of real world phenomena. In the industrial practice, negation is typically used in conjunction with other operators that allow the compiler to replace it. In case it is used explicitly by the user, the type checker helps. If the use of negation is such that an ambiguity arises, the user can fix this error by adding the intended type information. Theoretically we are not too much astray, since division and negation are related by a combination of the Schröder rule (though replaced by the Dedekind rule) and the complement of the identity, which exists in Ampersand. With this combination of theoretical and practical typing assistance Ampersand offers a version of relation calculus to its users that has appeared to be more than acceptable in practice.

5 Conclusion

We have enriched the heterogeneous relations with the notion of supertyping. With a rather natural construction the partial operations (composition and disjunction) are extended to total operations, so that much of the manipulative power of the homogeneous relations is regained for the heterogeneous relations. The result is a relational framework that explains why the current typing arguments in the business rule tool Ampersand do stand to reason. It also raises the question what we can do about the role of the negation, a partial answer to which may be found in the use of the division.The design advantage and the educational value of the Dedekind rule and the divisions could has not been judged, since these novelties are not yet taught in relation to Ampersand. The use of negation in constructing business rules needs to be investigated further.

References

1. Aarts, C., Backhouse, R., Hoogendijk, P., Voermans, T., van der Woude, J.: A relational theory of datatypes (1992), STOP summer school notes, http://www.cs.nott.ac.uk/~rcb/papers/abstract.html#book
2. Backhouse, R.: Pair algebras and Galois connections. Information Processing Letters 67(4), 169–176 (1978)
3. Brink, C., Kahl, W., Schmidt, G. (eds.): Relational methods in computer science. Advances in computing. Springer, Heidelberg (1997)
4. Freyd, P., Scedrov, A.: Categories, allegories. North-Holland Publishing Co., Amsterdam (1990)
5. Maddux, R.: Relation algebras. Elsevier, Amsterdam (2006)
6. Riguet, J.: Relations binaires, fermetures, correspondances de Galois. Bull. Soc. Math. France 76, 114–155 (1948)
7. Schmidt, G., Ströhlein, T.: Relationen und Grafen. Springer, Heidelberg (1988)
8. Michels, G., Joosten, S., van der Woude, J., Joosten, S.: Ampersand: Applying Relation Algebra in Practice. In: de Swart, H. (ed.) RAMICS 2011. LNCS, vol. 6663, pp. 280–293. Springer, Heidelberg (2011)
9. Tarski, A.: On the calculus of relations. Journal of Symbolic Logic 6(3), 73–89 (1941)