Relational Operations in FOLE

Robert E. Kent
Ontologos

Abstract. This paper discusses relational operations in the first-order logical environment FOLE (Kent [7]). Here we demonstrate how FOLE expresses the relational operations of database theory in a clear and implementable representation. An analysis of the representation of database tables/relations in FOLE (Kent [10]) reveals a principled way to express the relational operations. This representation is expressed in terms of a distinction between basic components versus composite relational operations. The 9 basic components fall into three categories: reflection (2), Booleans or basic operations (3), and adjoint flow (4). Adjoint flow is given for signatures (2) and for type domains (2), which are then combined into full adjoint flow. The basic components are used to express various composite operations, where we illustrate each of these with a flowchart. Implementation of the composite operations is then expressed in an input/output table containing four parts: constraint, construction, input, and output. We explain how limits and colimits are constructed from diagrams of tables, and then classify composite relational operations into three categories: limit-like, colimit-like and unorthodox.

Keywords: tables, adjoint flow, relational operations.
Table of Contents

Relational Operations in **FOLE** ........................................ 1
Robert E. Kent

1 Introduction .................................................. 3
2 Overview .................................................. 4
3 Basic Components .......................................... 10
   3.1 Reflection ........................................ 10
   3.2 Booleans ........................................ 12
   3.3 Adjoint Flow .................................... 14
      3.3.1 Fixed Type Domain ....................... 14
      3.3.2 Fixed Signature ......................... 17
      3.3.3 All Tables .............................. 20
      3.3.4 The Square ............................ 22
4 Composite Operations for Limits ....................... 23
   4.1 Quotient ....................................... 24
   4.2 Core ........................................... 26
   4.3 Natural Join .................................. 28
      4.3.1 Semi-join ................................ 32
      4.3.2 Anti-join ................................ 34
4.4 Generic Meet ....................................... 36
5 Composite Operations for Colimits ................... 39
   5.1 Co-quotient .................................. 40
   5.2 Co-core ........................................ 42
   5.3 Data-type Join ................................ 44
      5.3.1 Data-type Semi-join .................. 48
      5.3.2 Data-type Anti-join .................. 50
   5.4 Generic Join .................................. 52
6 Unorthodox Composite Operations ..................... 55
   6.1 Selection ....................................... 56
   6.2 Select-join ..................................... 58
   6.3 Filtered Join .................................. 61
   6.4 Data-type Meet ................................ 63
   6.5 Subtraction ..................................... 65
   6.6 Division ........................................ 67
   6.7 Outer-join ..................................... 69
7 Conclusion and Future Work .............................. 71
A Appendix .................................................. 72
   A.1 Tabular Components ............................ 72
   A.2 Tables, Tabular Flow and Relations ........... 75
   A.3 Completeness and Co-completeness ............. 80
      A.3.1 Completeness ................................ 81
      A.3.2 Co-completeness .......................... 83
1 Introduction

*The Relational Model.* Many-sorted (multi-sorted) first-order predicate logic represents a community’s “universe of discourse” as a heterogeneous collection of objects by conceptually scaling the universe according to types. The relational model for database management uses a structure and language consistent with this logic. The relational model was initially discussed in two papers: “A Relational Model of Data for Large Shared Data Banks” by Codd [3] and “The Entity-Relationship Model – Toward a Unified View of Data” by Chen [2].

The relational model follows many-sorted logic by representing data in terms of many-sorted relations, subsets of the Cartesian product of multiple data-types. All data is represented horizontally in terms of tuples, which are grouped vertically into relations. A database organized in terms of the relational model is a called relational database. The relational model provides a method for modeling the data stored in a relational database and for defining queries upon it. In the relational model there are two approaches for database management: the relational algebra, which defines an imperative language, and the relational calculus, which defines a declarative language.

*FOLE.* The first order logical environment *FOLE* is a category-theoretic approach to many-sorted first order predicate logic. The *FOLE* approach to logic, and hence to databases, relies upon two mathematical concepts: (1) lists and (2) classifications. Lists represent database signatures and tuples; classifications represent data-types and logical predicates. *FOLE* represents the header of a database table as a list of sorts, and represents the body of a database table as a set of tuples classified by the header. The notion of a list is common in category theory (Mac Lane [13]). The notion of a classification is described in two books: “Information Flow: The Logic of Distributed Systems” by Barwise and Seligman [1] and “Formal Concept Analysis: Mathematical Foundations” by Ganter and Wille [5].

In this paper we explain how *FOLE* provides a categorically well-founded semantics for relational algebra. We present that semantics in a detail suitable for implementation. The *FOLE* semantics for the relational calculus will appear elsewhere. *FOLE* is described in multiple papers by the author: “Database Semantics” [6], “The First-order Logical Environment” [7], “The ERA of *FOLE*: Foundation” [8], “The ERA of *FOLE*: Superstructure” [9], and “The *FOLE* Table” [10].

*Comparisons.* To a large extent this paper follows the relational operators that Codd introduced for relational algebra. In his original paper [3] Codd introduced the eight relational operators: union, intersection, difference, Cartesian product, selection (restriction), projection, join, and division. In his book [4], the basic operators (chap.4) are: selection, projection, natural join, select-join, union, difference, intersection, and division. Various advanced operators (chap.5) include: semi-theta-join, outer-equi-join, and outer-natural-join.

This contrasts somewhat to the approach in *FOLE* (this paper), which consists of the concise collection of basic components of §3 and the large collection
of composite operators of §4, 5, 6. The basic components are of three kinds: reflection (§3.1) \(^1\) (image, include), Booleans or basic operators (§3.2) (union, intersection, difference), and adjoint flow (§3.3) (projection/inflation and expansion/restriction). The composite operators (Tbl. 1) are defined in terms of the basic components. These consist of three kinds: limit operators (§ 4), colimit operators (§ 5) and unorthodox operators (§ 6). Many more could be defined. The operations in the \texttt{FOLE} approach are based upon principles implicit in the structure of the mathematical context of tables \textsf{Tbl}, as presented in detail in the paper [10] and summarized in the appendix §A below.

## 2 Overview

Previous papers about \texttt{FOLE} have been largely theoretical. This paper is more applied. Here we discuss one representation of the relational model called relational algebra. \(^2\) Relational algebra involves operations that are used to formulate database queries. To a large extent this paper follows the eight relational operators that Codd introduced. Here we define relational operations in terms of a \texttt{FOLE} relational database. Each \texttt{FOLE} relational database is built from the basic concept of a \texttt{FOLE} table, which is discussed in detail in the paper [10]. The mathematical context of \texttt{FOLE} tables is divided into fiber contexts based at each of the tabular components: signatures, type domains, and signed domains. Tabular flow is used to move tables between fiber contexts. In the appendix § A we review some basic concepts of \texttt{FOLE}: the tabular components in § A.1 of type domains, signatures, and signed domains; and tables, tabular flow and relations in § A.2. The paper [10] has a more complete presentation of these concepts.

This paper offers a framework in which to define the operators of relational algebra. To a certain extent we follow category-theoretic guidelines and spirit. The operators of relational algebra are split into two groups: basic and composite. Basic operations (§3.2) are defined at the small scope of a fixed signed domain. Most composite operators are defined at an intermediate scope, either for a fixed type domain or for a fixed signature. \(^3\) At a fixed type domain, composite operators are defined in terms of the basic operators (Booleans) and adjoint flow along signature morphisms (§ 3.3.1). At a fixed signature, composite operators are defined in terms of the basic operators (Booleans) and adjoint flow along type domain morphisms (§ 3.3.2). Composite operators are of two kinds, either orthodox or unorthodox. Orthodox composite operators follow category-theoretic guidelines, corresponding to either the computation of limits (§ 4) or

---

\(^1\) Tables and relations are informationally equivalent. The inclusion of relations into tabular operations can be implicit. The image relation of a tabular result should be made explicit.

\(^2\) Following the original discussion of \texttt{FOLE} (Kent [7]) in the knowledge-representation community, we use “mathematical context” for the term “category”, “passage” for the term “functor”, and “bridge” for the term “natural transformation”.

\(^3\) The generic composite operators are defined at the large scope of all tables.
the computation of colimits (§5). Unorthodox operations (§6), for various reasons, do not. A flowchart is used to visualize the definition of each composite operator. The links between flowchart symbols are typed by either a signature (when operating in the context of a fixed type domain), a type domain (when operating in the context of a fixed signature), or a signed domain (when operating in the full table context).

![Flowchart](image)

**Fig. 1. FOLE Composite Relational Operations (Specification/Evaluation)**

Fig. 1 illustrates the specification and evaluation of the composite operations defined in this paper. Many of these need to use only a sufficient or adequate collection of tables (Def. 2) and (Def. 3) for their input; in particular, quotient and co-quotient, natural join and data-type join, and generic meet and generic join. The notation and description of the composite operators is given in Tbl. 1. These consist of limit, colimit and unorthodox varieties. The limit operations are applied versions of the theoretical limit operation in various scopes. The dual holds for the colimit operations. To a certain extent the limit operations follow the spirit of classical relational database operations by controlling the database through the header. The colimit operations allow you to get control of your data at the query level by manipulating the data-types. Unorthodox operations do not follow these prescriptions for various reasons; either the definition is deviant or more complex. Each limit operation has a dual, which is a colimit operation. For example, natural join is a limit operation, whose dual is the data-type join, which is a colimit operation. In addition, in the unorthodox category, the filtered join is dual to the data-type meet.

---

4 The limit operation symbols have squarish shapes, the colimit operation symbols have roundish shapes.

5 See comments in §6.3 and §6.4 on the deviancy of filtered join and data-type meet.
Limit Operations. In §4, we define (Tbl. 2) the composite operations for limits. Quotient (§4.1) represents equalizer. As the flowchart in Fig. 15 indicates, quotient is a unary operator taking one argument, one table whose signature is the target of a parallel pair of $X$-signature morphisms; it is defined by the single operation of inflation. §4.2 covers core. As the flowchart in Fig. 16 indicates, core is a binary operator taking two arguments, two tables whose type domains are linked to a third type domain through an opspan of $X$-type domain morphisms; it is the composition of restriction (twice) followed by meet. Natural join (§4.3) represents pullback. As the flowchart in Fig. 17 indicates, natural join is a binary operator taking two arguments, two tables whose signatures are linked to a third signature through a span of $X$-signature morphisms; it is the composition of inflation (twice) followed by meet. Cartesian product is a special case. §4.3.1 covers semi-join. There are two semi-joins, left or right. As the flowchart in Fig. 18 indicates, semi-join takes two arguments, two tables constrained as in the natural join; it is the composition of natural join followed by projection. §4.3.2 covers anti-join. There are two anti-joins, left or right. As the flowchart in Fig. 19 indicates, anti-join takes two arguments, two tables constrained as in the natural join; it is the composition of semi-join followed by the difference with one of the two arguments. Generic meet (§4.4) represents limit. As the flowchart in Fig. 20 indicates, generic meet takes a sufficient indexed collection of $n$ tables (Def. 2); it is the composition of restriction (inflation $n$ times) followed by meet.

Colimit Operations. In §5, we define (Tbl. 8) the composite operations for colimits. Co-quotient (§5.1) represents co-equalizer. As the flowchart in Fig. 23 indicates, co-quotient is a unary operator taking one argument, one table whose type domain is the source of a parallel pair of $X$-type domain morphisms; it is defined by the single operation of expansion. §5.2 covers co-core. As the flowchart in Fig. 24 indicates, co-core is a binary operator taking two arguments, two tables whose signatures are linked to a third signature through a span of $X$-signature morphisms; it is the composition of projection (twice) followed by join. Data-type join (§5.3) represents pushout. As the flowchart in Fig. 25 indicates, data-type join is a binary operator taking two arguments, two tables whose type domains are linked to a third type domain through an op-span of $X$-type domain morphisms; it is the composition of expansion (twice) followed by join. Disjoint sum is a special case. §5.3.1 covers data-type semi-join. There are two semi-joins, left or right. As the flowchart in Fig. 26 indicates, data-type semi-join takes two arguments, two tables constrained as in the data-type join; it is the composition of data-type join followed by restriction. §5.3.2 covers data-type anti-join. There are two anti-joins, left or right. As the flowchart in Fig. 27 indicates, anti-join takes two arguments, two tables constrained as in the data-type join; it is the composition of data-type semi-join followed by the difference with one of the two arguments. Generic join (§5.4) represents colimit. As the flowchart in Fig. 28 indicates, generic join takes a sufficient indexed collection of $n$ tables (Def. 3); it is the composition of projection (expansion $n$ times) followed by join.
Unorthodox Operations. In §6, we define (Tbl. 14) the unorthodox composite operations. §6.1 covers selection. As the flowchart in Fig. 30 indicates, selection takes two arguments, a principal table and an auxiliary relation selected against; it is the composition of inflation (twice) followed by meet. Selection is a special case of natural join. §6.2 covers select-join. As the flowchart in Fig. 31 indicates, select-join takes three arguments, two principal tables that are joined, and an auxiliary relation selected against; it is the composition of natural join followed by selection. Select-join is a natural multi-join. §6.3 covers filtered join. As the flowchart in Fig. 32 indicates, filtered join takes two arguments, two tables whose type domains are linked to a third type domain through an op-span of X-type domain morphisms; it is the composition of restriction (twice) followed by join. §6.4 covers data-type meet. As the flowchart in Fig. 33 indicates, data-type meet takes two arguments, two tables constrained as in the data-type join; it is the composition of expansion (twice) followed by meet. §6.5 covers subtraction. As the flowchart in Fig. 34 indicates, subtraction takes two arguments, two tables with one subtracted from the other. There are two type domains, with one table having the product type domain, and the other table having one of the component type domains. It is the composition of expansion (once) followed by difference. §6.6 covers division. As the flowchart in Fig. 35 indicates, division takes two arguments, two tables with one divided by the other. There are two signatures, with one table having the coproduct signature, and the other table having one of the component signatures. Division uses projection (twice), Cartesian product and difference (twice). §6.7 covers outer-join. There are two outer-joins, left or right. As the flowchart in Fig. 36 indicates, left outer-join takes two arguments, two tables with one outer-joined with the other. For outer-join, we expand the type domain by adding a null value. We then add a third table that consists of a single tuple of null values. Outer-join uses natural join, anti-join, expansion, Cartesian product and union. Since outer-join uses two signatures and two type domains with a common sort set, outer-join is an interesting and non-trivial case of adjoint flow in the square (§3.3.4).

---

6 In §A.2 we give several examples of common auxiliary relations.
| Definition | Name | Symbol | Arity | Scope |
|------------|------|-------|-------|-------|
| § 4 Limit  |  |      |       |       |
| § 4.1     | inflate $\neq$ quotient $\triangleright$ unary Tbl(A) |
| § 4.2     | restrict $\times 2 \circ$ meet $\neq$ core $\cap$ binary Tbl(S) |
| § 4.3     | inflate $\times 2 \circ$ meet $\neq$ natural join $\boxdot$ binary Tbl(A) |
| § 4.3.1   | natural join $\circ$ project $\neq$ semi-join $\sqcup$ $\sqcap$ binary Tbl(A) |
| § 4.3.2   | (id, semi-join) $\circ$ diff $\neq$ anti-join $\ominus$ $\ominus$ binary Tbl(A) |
| § 4.4     | (restrict $\circ$ inflate) $\times n \circ$ meet $\neq$ generic meet $\| n$-ary Tbl |

| Colimit |  |      |       |       |
| § 5.1   | expand $\neq$ co-quotient $\triangleleft$ unary Tbl(S) |
| § 5.2   | project $\times 2 \circ$ join $\neq$ co-core $\cup$ binary Tbl(A) |
| § 5.3   | expand $\times 2 \circ$ join $\neq$ data-type join $\oplus$ binary Tbl(S) |
| § 5.3.1 | data-type join $\circ$ restrict $\neq$ data-type semi-join $\oplus$ $\ominus$ binary Tbl(S) |
| § 5.3.2 | (id, semi-join) $\circ$ diff $\neq$ data-type anti-join $\ominus$ $\ominus$ binary Tbl(S) |
| § 5.4   | (project $\circ$ expand) $\times n \circ$ join $\neq$ generic join $\| n$-ary Tbl |

| Unorthodox |  |      |       |       |
| § 6.1     | (inflate, id) $\circ$ meet $\neq$ selection $\sigma$ binary Tbl(A) |
| § 6.2     | (id, natural join) $\circ$ select $\neq$ select-join $\boxdot$ ternary Tbl(A) |
| § 6.3     | restrict $\times 2 \circ$ meet $\neq$ filtered join $\ominus$ binary Tbl(S) |
| § 6.4     | expand $\times 2 \circ$ meet $\neq$ data-type meet $\boxdot$ binary Tbl(S) |
| § 6.5     | (id, expand) $\circ$ diff $\neq$ subtraction $\sim$ binary Tbl(S) |
| § 6.6     | multiple $\neq$ division $\div$ binary Tbl(A) |
| § 6.7     | multiple $\neq$ outer-join $\boxdot$ $\boxdot$ binary Tbl |

Table 1. FOLE Composite Operations
Aside: For logical interpretation in FOLE, the *domain of discourse* is the context of tables, with tuples representing individuals and tables representing propositions. Interpretation is defined in terms of propositional and predicate logic. Propositional logic uses conjunction, disjunction and negation. Conjunction (\(\text{and}\)) is represented by the meet at various scopes: the small scope of a signed domain (§3.2 intersection \(\text{T}_1 \land \text{T}_2\)), the intermediate scope of a type domain (§4.3 natural join \(\text{T}_1 \varpi \text{T}_2\)), and the large scope of all tables (§4.4 generic meet \([\prod \text{T}]\)). Disjunction (\(\text{or}\)) is represented by the join at various scopes: the small scope of a signed domain (§3.2 union \(\text{T}_1 \lor \text{T}_2\)), the intermediate scope of a signature (§5.3 data-type join \(\text{T}_1 \oplus \text{T}_2\)), and the large scope of all tables (§5.4 generic join \([\biguplus \text{T}]\)). Negation (\(\text{not}\)) is represented by the difference at various scopes: the small scope of a signed domain (§3.2 difference \(\text{T}_1 - \text{T}_2\)), and the intermediate scope of a signature (§6.5 subtraction \(\text{T}_1 \sim \text{T}_2\)). Predicate logic adds the flow of tables to logical interpretation: (§3.3.1 projection/inflation \(\langle \text{tbl}_A(h) \rightarrow \text{tbl}_A(h) \rangle\)) between the intermediate scope of signatures, (§3.3.2 expansion/restriction \(\langle \text{tbl}_S(g) \rightarrow \text{tbl}_S(g) \rangle\)) between the intermediate scope of type domains, and (§3.3.3 projection \(\circ\) expansion/restriction \(\circ\) inflation \(\langle \text{tbl}(h, f, g) \rightarrow \text{tbl}(h, f, g) \rangle\)) between the small scope of signed domains.  

\(^a\) For more on this, see *Formula Interpretation* §2.2.1 of the paper “The ERA of FOLE: Superstructure” [9].  

\(^b\) To allow algebraic computations on the data domains, see the paper “The First-order Logical Environment” [7], which defines syntactic flow along term vectors.
3 Basic Components

Basic components are elements to be used in flowcharts. A case in point is the quotient composite operation of § 4.1, whose flowchart has only one component — inflation. There are three kinds of basic components: two reflectors, three Booleans (basic operations), and four components of adjoint flow (two each for type domain and signature).

3.1 Reflection.

Let $D = \langle S, A \rangle$ be a fixed signed domain. Here, we define reflection between the smallest fiber contexts $\text{Tbl}(D) \sqsupseteq \text{Rel}(D)$. The context of relations forms a sub-context of tables: there is an inclusion passage $\text{Rel}(D) \xrightarrow{\text{inc}_D} \text{Tbl}(D)$. Conversely, there is an image passage $\text{Tbl}(D) \xrightarrow{\text{im}_D} \text{Rel}(D)$ defined as follows.

A table $\langle K, t \rangle \in \text{Tbl}(D)$ with tuple function $F: K \rightarrow \text{tup}_A(S)$ is mapped to the relation $\langle \varphi t(K), i \rangle \in \text{Rel}(D)$ with inclusion tuple function $F: K \rightarrow \text{tup}_A(S)$, which is essentially its tuple subset $\varphi t(K) \subseteq \text{tup}_A(S)$. A table morphism $T' = \langle K', t' \rangle \xleftarrow{k} \langle K, t \rangle = T$ with table morphism condition $k \cdot t' = t$ is mapped to the relation morphism $R' = \langle \varphi t'(K'), i' \rangle \xrightarrow{\varphi t(K), i} R$ with relation morphism condition $\varphi t'(K') \subseteq \varphi t(K)$.

The diagram above factors the condition $k \cdot t' = t$ by diagonal fill-in. This gives the D-table morphism $\text{inc}_D(\text{im}_D(T')) \xleftarrow{\varphi t(K), i} \text{inc}_D(\text{im}_D(T))$, which is the image-inclusion composite passage applied to the $D$-table morphism $T' \xleftarrow{k} T$.

**Proposition 1.** Image and inclusion form reflections on full and fiber contexts:

$$\langle \text{im} \dashv \text{inc} \rangle : \text{Tbl} \xrightarrow{\text{inc}} \text{Rel}$$
$$\langle \text{im}_A \dashv \text{inc}_A \rangle : \text{Tbl}(A) \xrightarrow{\text{inc}_A} \text{Rel}(A)$$
$$\langle \text{im}_D \dashv \text{inc}_D \rangle : \text{Tbl}(D) \xrightarrow{\text{inc}_D} \text{Rel}(D) = \langle \varphi \text{tup}_A(S), \subseteq \rangle$$
Proof. The reflection at type domain $A$ appears in appendix § A.1 of [10].

Each reflection embodies the notion of informational equivalence. The inclusion operator $\textbf{Tbl}(D) \xrightarrow{\text{inc}} \textbf{Rel}(D)$ can be used at the input of any composite operator on tables. Dually, the image operator $\textbf{Tbl}(D) \xrightarrow{\text{im}} \textbf{Rel}(D)$ can be used at the output of any composite operator on tables.\footnote{For example, the inclusion operator is used before the inflation operator in § 4.3 to define the selection operation, and the image operator can be used after the projection operator in § 4.3.1 to define the semi-join operation.}
3.2 Booleans.

![FOLE Boolean Operators]

The Boolean operators are binary operators on tables based upon the traditional mathematical set operations. These operators require that both tables have the same set of attributes; i.e. type domain \( \mathcal{A} = \langle X, Y, \models_{\mathcal{A}} \rangle \). In addition, they require compatibility. Two tables are said to be compatible when both tables have the same number of attributes and corresponding attributes have the same data type (int, char, float, date, ...). Hence, two tables are compatible when they have the same type domain \( \mathcal{A} = \langle X, Y, \models_{\mathcal{A}} \rangle \) and the same signature \( \mathcal{S} = \langle I, x, X \rangle \); in short, when they have the same signed domain \( \mathcal{D} = \langle \mathcal{S}, \mathcal{A} \rangle \).

Here, we define the Boolean operators of meet, join and difference in the smallest table fiber context \( \text{Tbl}(\mathcal{D}) = \text{Tbl}_{\mathcal{A}}(\mathcal{S}) \). The corresponding Boolean operators in \( \text{Rel}(\mathcal{D}) \) are called set intersection, set union and set difference. \(^8\) Let \( \mathcal{T} = \langle K, t \rangle \) and \( \mathcal{T}' = \langle K', t' \rangle \) be two FOLE tables in \( \text{Tbl}(\mathcal{D}) \). These tables have tuple functions \( K \xrightarrow{t} \text{tup}(\mathcal{D}) = \text{tup}_{\mathcal{A}}(\mathcal{S}) \) and \( K' \xrightarrow{t'} \text{tup}(\mathcal{D}) = \text{tup}_{\mathcal{A}}(\mathcal{S}) \) with two image FOLE relations \( \varphi t(K), \varphi t'(K') \subseteq \text{tup}(\mathcal{D}) = \text{tup}_{\mathcal{A}}(\mathcal{S}) \) in \( \text{Rel}(\mathcal{D}) = \text{Rel}_{\mathcal{A}}(\mathcal{S}) = \langle \varphi \text{tup}_{\mathcal{A}}(\mathcal{S}), \subseteq \rangle \).

\(^8\) Codd also listed the Cartesian product as a Boolean operator. However, in this paper the Cartesian product is defined in the larger fiber \( \text{Tbl}(\mathcal{A}) \) and is closely connected to the natural join operator (see § 4.3).
Relational Operations in **FOLE**

\( \cap \): The intersection (meet, conjunction) operator produces the set of tuples that two tables share in common. The intersection operation defines the **FOLE** table \( T \land T' = \langle \hat{K}, (t, t') \rangle \) whose key set \( \hat{K} \subseteq K \times K' \) is the pullback and whose tuple map is the mediating function \( \hat{K} \xrightarrow{(t,t')} \text{tup}_A(S) \) of the opspan \( K \xrightarrow{\text{tup}} \text{tup}_A(S) \xleftarrow{t'} K' \), which maps a pair of keys \((k, k') \in \hat{K}\) to the common tuple \( t(k) = t'(k') \in \text{tup}_A(S) \). The image relation is the set-theoretic intersection \( \text{im}_D(T \land T') = \text{im}_D(T) \cap \text{im}_D(T') \). Intersection is the product in \( \text{Tbl}(D) \) with projection morphisms

\[ T \xrightarrow{\pi} T \land T' \xrightarrow{\pi'} T'. \]

\( \lor \): The union (join, disjunction) operator combines the tuples of two tables and removes all duplicate tuples from the result. The union operation defines the **FOLE** table \( T \lor T' = \langle K + K', [t, t'] \rangle \) whose key set is the disjoint union \( K + K' \) and whose tuple map is the comediating function \( K + K' \xrightarrow{[t, t']} \text{tup}_A(S) \) of the opspan \( K \xrightarrow{\text{tup}} \text{tup}_A(S) \xrightarrow{t'} K' \), which maps a key \( k \in K \) to \( t(k) \in \text{tup}_A(S) \) and maps a key \( k' \in K' \) to \( t'(k') \in \text{tup}_A(S) \). The image relation is the set-theoretic union \( \text{im}_D(T \lor T') = \text{im}_D(T) \cup \text{im}_D(T') \). Union is the coproduct in \( \text{Tbl}(D) \) with injection morphisms

\[ T \xleftarrow{\iota} T \lor T' \xleftarrow{\iota'} T'. \]

\( - \): The difference operator acts on two tables and produces the set of tuples from the first table that do not exist in the second table. The difference operation defines the **FOLE** table \( T - T' = \langle \hat{K}, \hat{t} \rangle \) whose key set \( \hat{K} \) is the tuple inverse image of the difference tuple set \( \hat{K} = t^{-1}(\varphi t(K) - \varphi t'(K')) \subseteq K \) and whose tuple map \( \hat{t} : \hat{K} \xrightarrow{t} \text{tup}_A(S) \) restricts to this subset. The image relation is the set-theoretic difference \( \text{im}_D(T - T') = \text{im}_D(T) - \text{im}_D(T') \). There is an inclusion morphism

\[ T \xrightarrow{\hat{\omega}} (T - T'). \]

**Proposition 2.** There are many algebraic laws for the Boolean operations: associativity, commutativity, idempotency for \( \land \) and \( \lor \); Distributive laws for \( \land \) over \( \lor \) and \( \lor \) over \( \land \); Distributive laws for \( \land \) w.r.t. \( \land \) and \( \lor \); Complement and double negation laws for \( - \); DeMorgans laws for \( - \) w.r.t. \( \land \) and \( \lor \).

**Proof.** Well-known. \n
\[ \text{Intersection } \bigcap_{i \in I} T_i \text{ and union } \bigvee_{i \in I} T_i \text{ can be generalized to any number of } D\text{-tables.} \]
3.3 Adjoint Flow.

\[ \text{project} \quad \Longleftrightarrow \quad \text{inflation} \]

**Fig. 4. FOLE Adjoint Flow Operators: Base \( A \)**

### 3.3.1 Fixed Type Domain.

In this section we use a fixed type domain \( A = \langle X, Y | \models A \rangle \). Here, we define adjoint flow in the mid-sized table fiber context \( \text{Tbl}(A) \). Let \( S' \xrightarrow{h} S \) be an \( X \)-sorted signature morphism with the arity (index) function \( I' \xrightarrow{h} I \), which satisfies the condition \( h \cdot s = s' \). Its tuple function \( \text{tup}_A(S') \xrightarrow{\text{tup}_A(h)} \text{tup}_A(S) \) defines by composition/pullback a fiber adjunction of tables

\[
\text{Tbl}_A(S') \xleftarrow{\langle \text{tup}_A(h) \rangle} \text{Tbl}_A(S) \quad \text{of} \quad \langle \Sigma_h \cdot \text{h*} \rangle
\]

**project:** The left adjoint (existential quantifier) \( \text{Tbl}_A(S') \xrightarrow{\text{tup}_A(h)} \text{Tbl}_A(S) \) defines projection. An \( A \)-table \( T = \langle K, t \rangle \in \text{Tbl}_A(S) \) is mapped to the \( A \)-table \( \Sigma_h(T) = T' = \langle K, t' \rangle \in \text{Tbl}_A(S') \) with its tuple function \( K \xrightarrow{t} \text{tup}_A(S') \) defined by pullback, \( t' = t \cdot \text{tup}_A(h) \). Here we have “horizontally abridged” (projected out sub-tuples from) tuples in \( \text{List}(Y) \) by tuple composition with the index function \( I' \xrightarrow{h} I \). There is an \( A \)-table morphism (LHS Fig. 5) \( T' = \langle S', K, t' \rangle \xleftarrow{\langle h,1,K,t \rangle} \langle S, K, t \rangle = T \). We say that \( A \)-table \( T' = \Sigma_h(T) \) is the projection of \( A \)-table \( T \) along \( X \)-sorted signature morphism \( S' \xrightarrow{h} S \). Being left adjoint in flow projection preserves colimits.

**inflated:** The right adjoint (inverse image) \( \text{Tbl}_A(S') \xleftarrow{\text{tup}_A(h)} \text{Tbl}_A(S) \) defines inflation. An \( A \)-table \( T' = \langle K', t' \rangle \in \text{Tbl}_A(S') \) is mapped to the \( A \)-table \( h^*(T) = T = \langle K, t \rangle \in \text{Tbl}_A(S) \), with its tuple function \( K \xleftarrow{t} \text{tup}_A(S) \) defined by pullback, \( k \cdot t' = t \cdot \text{tup}_A(h) \). Here we have “horizontally inflated” tuples in \( \text{List}(Y) \) by tuple pullback back along the index function \( I' \xrightarrow{h} I \). There is an \( A \)-table morphism (RHS Fig. 5) \( T' = \langle S', K', t' \rangle \xleftarrow{\langle h,k \rangle} \langle S, K, t \rangle = T \). We say that \( A \)-table \( T = h^*(T') \) is the inflation of \( A \)-table \( T' \) along \( X \)-sorted signature morphism \( S' \xrightarrow{h} S \). Being right/left adjoint in flow, inflation preserves limits/colimits.

---

10 There is a discussion of type domain indexing in § 3.4.1 of “The FOLE Table” [10].

11 Visually, \( \langle \cdots t_{h(i')} \cdots | i' \in I' \rangle \leftrightarrow \langle \cdots t_i \cdots | i \in I \rangle \).

12 Inverse image is left adjoint \( \text{Tbl}_A(S') \xleftarrow{\langle h^* \cdot \text{h} \rangle} \text{Tbl}_A(S) \) to universal quantification.
When the index function is an inclusion, inflation enlarges the horizontal aspect of tables.

**How does projection work?** When the index function is an inclusion \( I' \xrightarrow{i} I \), the projection \( \Sigma_h(T) \) of an \( \mathcal{A} \)-table \( T = \langle K, t \rangle \in \text{Tbl}_A(S) \) consists of the sub-tuples of \( T \) indexed by \( I' \). Hence, projection abridges the horizontal aspect of tables, ending with a subset of columns. In particular, for any \( \mathcal{A} \)-table \( T = \langle S, K, t \rangle \), an index \( i \in I \) defines an arity (index) function \( 1 \rightarrow I \), thus forming an indexing \( X \)-sorted signature morphism \( \langle 1, x \rangle \xrightarrow{i} S \) from signature (sort) \( 1 \xrightarrow{h} X \) satisfying the naturality condition \( i \cdot s = s_1 = x \). Projection along \( 1 \xrightarrow{h} I \) defines an \( \mathcal{A} \)-table morphism \( \mathcal{T}_i = \langle 1, x, K, t_1, \rangle \xleftarrow{i} \langle 1, x \rangle \xrightarrow{i} \langle S, K, t \rangle = T \) satisfying the naturality condition \( t_i = t \cdot \text{tup}_A(i) \), which states that “\( t_i \) is the \( i \)-th projection of \( t' \).”

**How does inflation work?** When the index function is an inclusion \( I' \xrightarrow{i} I \), an \( \mathcal{A} \)-relation \( R' = \langle R', t' \rangle \in \text{Rel}_A(S') \subseteq \text{Tbl}_A(S') \) with tuple subset \( R' \subseteq \text{tup}_A(S') \) is mapped (isomorphically) to the inflation relation \( h^*(R') \in \text{Rel}_A(S) \subseteq \text{Tbl}_A(S) \) with tuple subset \( R' \times \text{tup}_A(I'', s') \subseteq \text{tup}_A(S) \), where \( I'' = I - I' \) is the index set complement and \( s'' : I'' \rightarrow X \) is the restriction of signature function \( s = [s', s''] : I = I'+I'' \rightarrow X \) to this complement. The target tuple set factors as \( \text{tup}_A(S) = \text{tup}_A(S') \times \text{tup}_A(S'') \). The tuple set \( \text{tup}_A(S'') \) is the inflation referred to in the name. Inflation enlarges the horizontal aspect of tables.

---

The \( \mathcal{A} \)-table \( \mathcal{T}_i = \langle 1, x, K, t_i \rangle \), essentially the \( i \)-th-column of \( T \), consists of signature \( \langle 1, x, X \rangle \) (sort \( x = s_i \in X \)), the same set \( K \) of keys, and the tuple (data value) function \( K \xrightarrow{i} \text{tup}_A(1, s) = \text{ext}_{\text{List}(\mathcal{A})}(1, s) \cong \text{ext}_A(x) = A \subseteq Y \).

---

Fig. 5. FOLE Adjoint Flow in \( \text{Tbl}(\mathcal{A}) \)

Fig. 6. Adjoint Flow Factor
Aside: Although we usually think of projection and inflation along an injective index function, here is an example along a surjective index function. The copower $X$-signature $\mathcal{S} + \mathcal{S}$ in the opspan $\mathcal{S} \xrightarrow{\eta_1} \mathcal{S} + \mathcal{S} \xleftarrow{\eta_2} \mathcal{S}$ has inclusion index functions $I \xrightarrow{i_1} I + I \xleftarrow{i_2} I$. There is an $X$-signature morphism $\mathcal{S} + \mathcal{S} \xrightarrow{\eta} \mathcal{S}$ with (surjective) index function $I + I \xrightarrow{\eta} I$ that erases the origin: $i_1 \cdot \eta = 1_I = i_2 \cdot \eta$.

Hence, the projection $\text{Tbl}_{\mathcal{A}}(\mathcal{S} \times \mathcal{S}) \xleftarrow{\text{id}_{\mathcal{S}} \times \text{id}_{\mathcal{S}}(\eta)} \text{Tbl}_{\mathcal{A}}(\mathcal{S})$ is actually a “creation” or a “duplication”, and the inflation $\text{Tbl}_{\mathcal{A}}(\mathcal{S} \times \mathcal{S}) \xrightarrow{\text{id}_{\mathcal{S}} \times \text{id}_{\mathcal{S}}(\eta)} \text{Tbl}_{\mathcal{A}}(\mathcal{S})$ is actually an “erasure”.

**Proposition 3.** Projection preserves union $\vee$. Projection is decreasing on intersection $\wedge$. Inflation preserves union $\vee$ and intersection $\wedge$.

**Proof.** Projection is co-continuous and preserves order. Inflation is continuous, co-continuous and preserves order.

**Application.** The fiber adjunction of tables $\text{Tbl}_{\mathcal{A}}(\mathcal{S}') \xleftarrow{\text{id}_{\mathcal{S}} \times \text{id}_{\mathcal{S}}(\eta)} \text{Tbl}_{\mathcal{A}}(\mathcal{S})$ for an $X$-sorted signature morphism $\mathcal{S}' \xrightarrow{h} \mathcal{S}$ is used as follows.

- To define **inflations**
  $$\text{Tbl}_{\mathcal{A}}(\mathcal{S}_1) \xrightarrow{\text{id}_{\mathcal{S}}(h_1)} \text{Tbl}_{\mathcal{A}}(\mathcal{S}) \xleftarrow{\text{id}_{\mathcal{S}}(h_2)} \text{Tbl}_{\mathcal{A}}(\mathcal{S}_2)$$
  from two peripheral signatures $\mathcal{S}_1$ and $\mathcal{S}_2$ to a central signature $\mathcal{S}$, you need an opspan of $\mathcal{S}_1 \rightarrow \mathcal{S} \leftarrow \mathcal{S}_2$ of $X$-sorted signature morphisms. One way to get this is to assume an $X$-sorted signature span $\mathcal{S}_1 \xleftarrow{h_1} \mathcal{S} \xrightarrow{h_2} \mathcal{S}_2$ to define a coproduct $X$-signature opspan $\mathcal{S}_1 \xrightarrow{i_1} \mathcal{S}_1 + \mathcal{S}_2 \xleftarrow{i_2} \mathcal{S}_2$. This is used by natural join in §4.3; hence, it is also used by Cartesian product, selection and select join there. Furthermore, it is used by semi-join in §4.3.1, by anti-join in §4.3.2, by outer-join in §6.7, and by division in §6.6.

- To define **projections**
  $$\text{Tbl}_{\mathcal{A}}(\mathcal{S}_1) \xleftarrow{\text{id}_{\mathcal{S}}(h_1)} \text{Tbl}_{\mathcal{A}}(\mathcal{S}) \xrightarrow{\text{id}_{\mathcal{S}}(h_2)} \text{Tbl}_{\mathcal{A}}(\mathcal{S}_2)$$
  from two peripheral signatures $\mathcal{S}_1$ and $\mathcal{S}_2$ to a central signature $\mathcal{S}$, you need a span of $\mathcal{S}_1 \xleftarrow{h_1} \mathcal{S} \xrightarrow{h_2} \mathcal{S}_2$ of $X$-sorted signature morphisms. This is used by project-join in §5.2.
3.3.2 Fixed Signature. Part of this paper, the adjoint flow in §3.3.1 and the classic relational operations of §4, deals with traditional relation algebra. This assumes that we can manipulate the header part of table, but that the data part is fixed. In the FOLE representation the header part is represented by a signature, and the data part is represented by a type domain. Hence, using FOLE to represent traditional relation algebra, we fix the type domain, and allow the signature to vary. However, in the FOLE representation of relation databases, we can manipulate both the header part and the data part (the dual approach to relational algebra). This section and §5 explain this dual approach.

![Fig. 7. FOLE Adjoint Flow Operators: Base S](image)

In this section we use a fixed signature $S = \langle I, x, X \rangle$. Here, we define adjoint flow in the mid-sized table fiber context $\text{Tbl}(S)$.\(^{13}\) Let $A' \xrightarrow{g} A$ be an $X$-sorted type domain morphism\(^ {14}\) with data value function $Y' \leftrightarrow Y$ satisfying the condition $\text{ext}_{A'} \cdot g^{-1} = \text{ext}_A$; or that $g^{-1}(A'_x) = A_x$ for all $x \in X$. This implies that $A'_x \supseteq \text{var}(A_x)$ for all $x \in X$.\(^ {15}\)\(^ {16}\) Its tuple function

$$tup_S(A') \xrightarrow{tup_S(g)(-)} tup_S(A)$$

defines composition/pullback a fiber adjunction of tables

$$\text{Tbl}_S(A') \xleftarrow{\left(\text{tbl}_S(g) - \text{tbl}_S(g)\right)_{-}^{-1} g} \text{Tbl}_S(A). \tag{2}$$

**restrict:** The right adjoint $\text{Tbl}_S(A') \xrightarrow{\text{tbl}_S(g)*} \text{Tbl}_S(A)$ defines restriction. An $S$-table $T' = \langle K', t' \rangle \in \text{Tbl}_S(A')$ is mapped to the $S$-table $g^*(T') = T = \langle K, t \rangle \in \text{Tbl}_S(A)$, with its tuple function $K \rightarrow \text{tup}_S(A)$ defined by pullback, $k \cdot t' = t \cdot \text{tup}_S(g)$. Here we have “vertically abridged” tuples in $\text{List}(Y')$ by tuple pullback with the data value function $Y' \leftrightarrow Y$. There is an $S$-table morphism (RHS Fig. 8) $T' = \langle A', K', t' \rangle \xrightarrow{(g,k)} \langle A, K, t \rangle = T$. We say that $S$-table $T = g^*(T')$ is the restriction of $S$-table $T'$ along $X$-sorted type domain morphism $A' \xrightarrow{g} A$.

**expand:** The left adjoint $\text{Tbl}_S(A') \xrightarrow{\text{tbl}_S(g)} \text{Tbl}_S(A)$ defines expansion. An $S$-table $T = \langle K, t \rangle \in \text{Tbl}_S(A)$ is mapped to the $S$-table $g^*(T) = T' = \langle K, t' \rangle \in \text{Tbl}_S(A')$, with its tuple function $K \rightarrow \text{tup}_S(A')$ defined by composition,

---

\(^{13}\) There is a discussion of signature indexing in § 3.3.1 of “The FOLE Table” [10].

\(^{14}\) This is described as an $X$-sorted type domain morphism $A' \xrightarrow{(f_X,g)} A$ in [10].

\(^{15}\) Hence, for an injective data value function $Y' \leftrightarrow Y$, we have the inclusion $A'_x \supseteq A_x$ for all $x \in X$.

\(^{16}\) Hence, $A = g^{-1}(A')$, as discussed by the Yin definition in §A.1.
\[ t' = t \cdot \text{tup}_S(g) \]. Here we have “vertically merged” tuples in \textbf{List}(Y) with tuples in \textbf{List}(Y') by tuple composition with the data value function \( Y' \leftarrow Y \).

There is an \( S \)-table morphism (LHS Fig. 8) \( T' = \langle A', K, t' \rangle \xleftarrow{(g \cdot k)} \langle A, K, t \rangle = T \). We say that \( S \)-table \( T' = \sum_g(T) \) is the expansion of \( S \)-table \( T \) along \( X' \)-sorted type domain morphism \( A' \xrightarrow{g} A \).

\begin{center}
\begin{tikzpicture}
  \node (A) at (0,0) {\textbf{A}'};  \node (B) at (2,0) {\textbf{A}};  \node (C) at (0,-2) {\textbf{A}'};  \node (D) at (2,-2) {\textbf{A}};
  \node (E) at (4,0) {\textbf{Tbl}_S(A')};  \node (F) at (6,0) {\textbf{Tbl}_S(A)};
  \node (G) at (4,-2) {\textbf{Tbl}_S(A')};  \node (H) at (6,-2) {\textbf{Tbl}_S(A)};

  \draw[->] (A) -- (B) node[midway,above] {\( g \)} node[midway,below] {\( k \)};
  \draw[->] (C) -- (D) node[midway,above] {\( g \)} node[midway,below] {\( k \)};
  \draw[->] (E) -- (F) node[midway,above] {\( \sum_g(T) \)} node[midway,below] {\( T \)};
  \draw[->] (G) -- (H) node[midway,above] {\( \sum_g(T) \)} node[midway,below] {\( T \)};

  \draw[->, bend left=20] (E) to node[above] {\textit{expand}} (G);
  \draw[->, bend left=20] (F) to node[above] {\textit{restrict}} (H);

  \draw[->, bend left=20] (C) to node[below] {\textit{expand}} (E);
  \draw[->, bend left=20] (D) to node[below] {\textit{restrict}} (F);

  \draw[->, bend right=20] (A) to node[below] {\textit{expand}} (C);
  \draw[->, bend right=20] (B) to node[below] {\textit{restrict}} (D);

  \draw[->, bend right=20] (E) to node[above] {\textit{expand}} (G);
  \draw[->, bend right=20] (F) to node[above] {\textit{restrict}} (H);

  \draw[->, bend right=20] (C) to node[above] {\textit{expand}} (E);
  \draw[->, bend right=20] (D) to node[above] {\textit{restrict}} (F);

\end{tikzpicture}
\end{center}

\textbf{Fig. 8. FOLE Adjoint Flow in Tbl(S)}

\begin{center}
\begin{tikzpicture}
  \node (A) at (0,0) {\textbf{Tbl}_S(A')};  \node (B) at (2,0) {\textbf{Tbl}_S(A)};
  \node (C) at (0,-2) {\textbf{Tbl}_S(A')};  \node (D) at (2,-2) {\textbf{Tbl}_S(A)};

  \draw[->] (A) -- (B) node[midway,above] {\textit{expand}} node[midway,below] {\textit{tbl}_S(g)};
  \draw[->] (C) -- (D) node[midway,above] {\textit{restrict}} node[midway,below] {\textit{tbl}_S(g)};

\end{tikzpicture}
\end{center}

\textbf{Fig. 9. Adjoint Flow Factor}
How does restriction work? We can think of restriction as a validation process. When the data value function is an inclusion $Y' \hookrightarrow Y$, we can regard the values in $Y' \setminus Y$ as being inauthentic and non-usable. For each sort $x \in X$, the data value function maps the source data type $A'_x$ to the restricted target data type $A_x = A'_x \cap Y$, and the restriction $g''(T')$ consists of only those tuples of $T'$ with data values in $Y$; all other tuples are omitted. Hence, restriction abridges the vertical aspect of tables, ending with a subset of rows.

Aside: Although we usually think of expansion and restriction along an injective data value function, here is an example along a surjective data value function. The power $X$-type domain $\mathcal{A} \times \mathcal{A}$ in the span $\mathcal{A} \leftarrow i_1 \rightarrow \mathcal{A} \times \mathcal{A} \leftarrow i_2 \rightarrow \mathcal{A}$ has inclusion data value functions $\mathcal{A} \leftarrow \iota \rightarrow \mathcal{A} \times \mathcal{A} \leftarrow \iota \rightarrow \mathcal{A}$. There is an $X$-type domain morphism $\mathcal{A} \rightarrow \mathcal{A} \times \mathcal{A}$ with (surjective) data value function $\mathcal{A} \leftarrow \iota \rightarrow \mathcal{A} \times \mathcal{A}$ that erases the origin: $i_1 \cdot \iota = 1_Y = i_2 \cdot \iota$. Hence, the expansion $\mathcal{Tbl}_S(\mathcal{A}) \xrightarrow{\tilde{db}_S(\iota)} \mathcal{Tbl}_S(\mathcal{A} \times \mathcal{A})$ is actually an “erasure”, and the restriction $\mathcal{Tbl}_S(\mathcal{A}) \xrightarrow{\tilde{db}_S(\iota)} \mathcal{Tbl}_S(\mathcal{A} \times \mathcal{A})$ is actually a “creation” or a “duplicaton”.

Application. The fiber adjunction of tables $\mathcal{Tbl}_S(\mathcal{A}') \xrightarrow{\tilde{db}_S(\iota)} \mathcal{Tbl}_S(\mathcal{A})$ for an $X$-sorted type domain morphism $\mathcal{A}' \rightarrow \mathcal{A}$ is used as follows.

- To define expansions
  $\mathcal{Tbl}_S(\mathcal{A}_1) \xrightarrow{\tilde{db}_S(\iota_1)} \mathcal{Tbl}_S(\mathcal{A}) \xrightarrow{\tilde{db}_S(\iota_2)} \mathcal{Tbl}_S(\mathcal{A}_2)$
  from two peripheral type domains $\mathcal{A}_1$ and $\mathcal{A}_2$ to a central type domain $\mathcal{A}$, you need a span of $\mathcal{A}_1 \leftarrow \mathcal{A} \rightarrow \mathcal{A}_2$ of $X$-sorted type domain morphisms. One way to get this is to assume an $X$-sorted type domain opspan $\mathcal{A}_1 \xleftarrow{g_1} \mathcal{A} \xrightarrow{g_2} \mathcal{A}_2$ to define a product $X$-type domain span $\mathcal{A}_1 \xleftarrow{s_1} \mathcal{A}_1 \times \mathcal{A}_2 \xrightarrow{s_2} \mathcal{A}_2$. This is used by data-type join in §5.3; hence, it is also used by disjoint sum and data-type meet there. This is used by data-type semi-join in §5.3.1; hence, it is also used by data-type semi-meet there. Finally, by subtraction in §6.5.

- To define restrictions
  $\mathcal{Tbl}_S(\mathcal{A}_1) \xrightarrow{\tilde{db}_S(\iota_1)} \mathcal{Tbl}_S(\mathcal{A}) \xrightarrow{\tilde{db}_S(\iota_2)} \mathcal{Tbl}_S(\mathcal{A}_2)$
  from two peripheral type domains $\mathcal{A}_1$ and $\mathcal{A}_2$ to a central type domain $\mathcal{A}$, you need an opspan of $\mathcal{A}_1 \xleftarrow{g_1} \mathcal{A} \xrightarrow{g_2} \mathcal{A}_2$ of $X$-sorted type domain morphisms. This is used by filter-join in §6.3.
### 3.3.3 All Tables

In this section we define adjoint flow in the full table context \( \text{Tbl} \). Let \( \langle S', A' \rangle \xrightarrow{(h,f,g)} \langle S, A \rangle \) be a signed domain morphism. By Prop. 11 in § A.2, its tuple function factors into two parts (Fig. 42).}

\[
\begin{align*}
\text{project/expand:} & \quad \text{Tbl}_{A'}(S') \xrightarrow{\text{expand}} \text{Tbl}_A(S) \quad \text{and} \quad \text{Tbl}_{A'}(S') \xrightarrow{\text{restrict}} \text{Tbl}_A(S) \\
\text{restrict/inflate:} & \quad \text{Tbl}_{A'}(A') \xrightarrow{\text{restrict}} \text{Tbl}_A(S) \quad \text{and} \quad \text{Tbl}_{A'}(A') \xrightarrow{\text{inflate}} \text{Tbl}_A(S).
\end{align*}
\]

Hence, the fiber adjunction of tables also factors into two parts: either forward (right-to-left) by composition or backward (left-to-right) by pullback. \(^{17}\)

\[
\begin{align*}
\text{Tbl}_{A'}(S') & \xrightarrow{\text{expand}} \text{Tbl}_A(S) & \text{Tbl}_A(S) & \xrightarrow{\text{project}} \text{Tbl}_{A'}(S') \quad & \text{Tbl}_A(S) & \xrightarrow{\text{inflate}} \text{Tbl}_{A'}(S') \\
\text{Tbl}_{A'}(A') & \xrightarrow{\text{restrict}} \text{Tbl}_A(S) & \text{Tbl}_A(S) & \xrightarrow{\text{restrict}} \text{Tbl}_{A'}(S') \quad & \text{Tbl}_A(S) & \xrightarrow{\text{inflate}} \text{Tbl}_{A'}(S').
\end{align*}
\]

\(\text{project/expand:} \quad \text{The left adjoint } \text{Tbl}_{A'}(S') \xrightarrow{\text{h-projector}} \text{Tbl}_A(S) \text{ defines projection-expansion.} \)

An \( A \)-table \( T = \langle K, t \rangle \in \text{Tbl}_A(S) \) is mapped to the \( S' \)-table \( \Sigma_{h,f,g}(T) = T' = \langle K, t' \rangle \in \text{Tbl}_{A'}(A') \), with its tuple function \( \Sigma_{h,f,g} : T' \xrightarrow{h \cdot \text{tup}} \text{Tbl}_{A'}(A') \) defined by composition, \( t' = t \cdot \text{tup}(h, f, g) \). Here we have “horizontally abridged” and then “vertically extended” tuples in \( \text{List}(Y') \) by tuple composition with the signed domain morphism \( \langle S', A' \rangle \xrightarrow{(h,f,g)} \langle S, A \rangle \). There is a table morphism (LHS Fig. 11) \( T' = \langle S', A', K, t' \rangle \xrightarrow{(h,f,g), \Sigma_{h,f,g}} \langle S, A, K, t \rangle = T \). We say that table \( T' = \Sigma_{h,f,g}(T) \) is the projection-expansion of table \( T \) along signed domain morphism \( \langle S', A' \rangle \xrightarrow{(h,f,g)} \langle S, A \rangle \).

\(\text{restrict/inflate:} \quad \text{The right adjoint } \text{Tbl}_{A'}(A') \xrightarrow{\text{h-pullback}} \text{Tbl}_A(S) \text{ defines restriction-inflation.} \)

A table \( T' = \langle K', t' \rangle \in \text{Tbl}_{A'}(A') \) is mapped to the table \( \langle h, f, g \rangle^*(T') = T = \langle K, t \rangle \in \text{Tbl}_A(S) \), with its tuple function \( \langle h, f, g \rangle^* : T' \xrightarrow{\text{h-pullback}} \text{Tbl}_A(S) \) defined by pullback, \( k \cdot t' = t \cdot \text{tup}(h, f, g) \). Here we have “vertically abridged” and then “horizontally inflated” tuples in \( \text{List}(Y') \) by tuple pullback along the signed domain morphism \( \langle S', A' \rangle \xrightarrow{(h,f,g)} \langle S, A \rangle \). There is a table morphism (RHS Fig. 11) \( T' = \langle S', A', K, t' \rangle \xrightarrow{(h,f,g), \Sigma_{h,f,g}} \langle S, A, K, t \rangle = T \). We say that table \( T = \langle h, f, g \rangle^*(T') \) is the restriction-inflation of table \( T' \) along signed domain morphism \( \langle S', A' \rangle \xrightarrow{(h,f,g)} \langle S, A \rangle \).

\(^{17}\) There is an \( h \)-closure for \( h \)-projection and an \( h \)-interior for inflation. There is a \( g \)-closure for \( g \)-expansion and a \( g \)-interior for restriction.
How does the left-adjoint of flow work? The left-adjoint of flow is projection followed
by expansion. When the index function is an inclusion \( I' \xrightarrow{h} I \), the flow \( \Sigma_h(T) \)
consists of the sub-tuples of \( T \) indexed by \( I' \). Hence, the first part of flow (projection)
restricts the horizontal aspect of tables, ending with a subset of columns. When the
data value function is an inclusion \( Y \xrightarrow{g} Y' \), the flow \( \Sigma_g(\Sigma_h(T)) \) does not alter tuples,
but places them in the larger context \( \text{List}(Y') \) of data tuples. Hence, the second part
of flow (expansion) does not alter the table \( \Sigma_h(T) \), but places it in a larger context of
data tuples. For pure projection, use an identity data value function \( Y \xrightarrow{1_Y} Y \). For
pure expansion, use an identity index function \( I \xrightarrow{1_I} I \).

How does right-adjoint of flow work? The right-adjoint of flow is restriction fol-
lowed by inflation. When the data value function is an inclusion \( Y' \xrightarrow{g} Y \), the flow \( \Sigma_g(T') \)
consists of only those tuples of \( T' \) with data values in \( Y' \); all other tuples are omitted.
Hence, the first part of flow (restriction) restricts the vertical aspect of tables, ending with a subset of rows. When the index function is an inclusion \( I' \xrightarrow{h} I \),
the flow \( h^*(g^*(T')) \) factors on the complement subset \( \text{tup}_A(T') \) for \( I'' = I - I' \).
Hence, the second part of flow (inflation) enlarges the horizontal aspect of tables. For
pure restriction, use an identity index function \( I \xrightarrow{1_I} I \). For pure inflation, use an
identity data value function \( Y \xrightarrow{1_Y} Y \).
3.3.4 The Square. Adjoint flow in the square works in the full context $\text{Tbl}$. By fixing a set of sorts $X$, we allow the header indexing to vary and we allow the data-types to vary.\footnote{In concert with §3.3.1 and §3.3.2, this might be entitled “Fixed Sort Set”.} This allows both processing in $\text{Tbl}(A)$ and processing in $\text{Tbl}(S)$. Let $D' = \langle S', A' \rangle \xrightarrow{(h, 1, g)} \langle S, A \rangle = D$ be a signed domain morphism with an identity sort function $X \xrightarrow{1} X$. This consists of an $X$-sorted signature $S' \xrightarrow{h} S$ which satisfies the condition $h \cdot s = s'$, and an $X$-sorted type domain morphism $A' \xrightarrow{g} A$ which satisfies the condition $\text{ext}_{A'} \cdot g^{-1} = \text{ext}_A$: or that $g^{-1}(A'_x) = A_x$ for all $x \in X$, so that $g^{-1}(A') = A$. Here $S' \xrightarrow{h} S$ could be a morphism in $\text{Tbl}(A')$ or $\text{Tbl}(A)$ (top/bottom of Fig. 12), and $A' \xrightarrow{g} A$ could be a morphism in $\text{Tbl}(S')$ or $\text{Tbl}(S)$ (left/right of Fig. 12). Hence, the signed domain morphism factors in two ways, as visualized in the square (Fig. 13). Because of this flexibility, by interspersing Booleans at the signed domains (corners), there are a variety of flowcharts definable here. The outer-join in §6.7 demonstrates one possible use for the square.
4 Composite Operations for Limits.

The basic components of § 3 are the components to be used in flowcharts. Composite operations are operations whose flowcharts are composed of one or more basic components. In addition to its basic components, a composite operation also has a constraint, which is used to construct its output. In this section we define the composite relational operations (Tbl. 2) related to limits. Each composite operation defined here has a dual relational operation (Tbl. 8) related to colimits. \(^{19}\) For limit operations we need only a sufficient collection of tables linked by the given collection of signatures (see Def. 2). Fig. 14 gives the possible routes of flow for limits.

\[
\begin{array}{c}
\text{quotient: } \forall \ W \wedge - \rightleftharpoons \text{project} \\
\text{core: } T_1 \cap S T_2 = \text{tbl}_{S}(g_1)(T_1) \times \text{tbl}_{S}(g_2)(T_2) \rightleftharpoons \text{restrict} \\
\text{natural join: } T_1 \sqcap A T_2 = \text{tbl}_{A}(h_1)(T_1) \times \text{tbl}_{A}(h_2)(T_2) \rightleftharpoons \text{inflating} \\
\text{semi-join: } T_1 \sqcap A T_2 = \text{tbl}_{A}(h_1)(T_1) \sqcap A T_2 \rightleftharpoons \text{restrict} \\
\text{anti-join: } T_1 \sqcap A T_2 = T_1 - (T_1 \sqcap A T_2) \rightleftharpoons \text{inflating} \\
\text{generic meet: } \forall \ W \wedge T = \bigwedge \{ \text{tbl}(h_i, \hat{f}_i, \hat{g}_i)(T_i) \mid i \in I \} \rightleftharpoons \text{restrict}
\end{array}
\]

Table 2. FOLE Composite Relational Operations for Limits

\(^{19}\) The quotient operation is dual to the co-quotient operation. The core operation is dual to the co-core operation. The natural join operation is dual to the data-type join operation. The generic meet operation is dual to the generic join operation.
4.1 Quotient.

In this section, we focus on tables in the context $\text{Tbl}(A)$ for fixed type domain $A$. In this context, generic meets — for the special case of the equalizer of a parallel pair — are called quotients. Here, we define an equivalence on attributes, specifically on indexes. In §5.1 we discuss a dual notion; there we define an equivalence on data values. The quotient operation defined here is dual to the co-quotient operation defined in §5.1.

**Constraint:** Consider a parallel pair of $X$-signature morphisms $h_1, h_2 : S \rightarrow S'$ in $\text{List}(X)$ consisting of a parallel pair of index functions $h_1, h_2 : I \rightarrow I'$ satisfying $h_1 \cdot s = s'$ and $h_2 \cdot s = s'$. This is the constraint for quotient (Tbl. 3).

**Construction:** We can construct the coequalizer of this constraint in $\text{List}(X)$ with quotient $X$-signature $\hat{S} = \langle \hat{I}, \hat{s} \rangle$ and projection $X$-signature morphism $\hat{S} \xrightarrow{\hat{h}} S$. Each index $i \in \hat{I}$ is an equivalence class of pairs generated by the relation $\{(h_1(i'), h_2(i')) : i' \in I'\}$. The index function $\hat{I} \xrightarrow{\hat{h}} I$ maps an index $i \in I$ to the equivalence class generated by these pairs. The sort map $\hat{I} \xrightarrow{\hat{h}} X$ is well-defined $\hat{s}(\hat{i}) = s(h_1(i')) = s(h_2(i'))$ for any equivalent pair $i' \in I'$.

---

**Fig. 14. Routes of Flow: Limits**

**Fig. 15. FOLE Quotient Flow Chart**
(\textit{h}_1(i'), \textit{h}_2(i')) for \textit{i}' \in \textit{I}'. This is the construction for quotient (Tbl. 3).

**Input/Output:** Consider a table \( \mathcal{T} = \langle \textit{K}, \textit{t} \rangle \in \textsf{Tbl}_A(\mathcal{S}) \). This table forms an adequate collection (Def. 2) to compute the equalizer. This is the input for quotient (Tbl. 3). The output is computed with one inflation.

- Inflation \( \textsf{Tbl}_A(\hat{\mathcal{S}}) \leftarrow \textsf{Tbl}_A(\mathcal{S}) \) (§3.3.1) along the tuple function of the \textit{X}-signature morphism \( \hat{\mathcal{S}} \leftarrow \mathcal{S} \) maps the table \( \mathcal{T} \) to the \textit{A}-table \( \hat{\mathcal{T}} = \textsf{Tbl}_A(\hat{\textit{h}})(\mathcal{T}) \) = \( \langle \hat{\textit{K}}, \hat{\textit{t}} \rangle \in \textsf{Tbl}_A(\hat{\mathcal{S}}) \), with its tuple function \( \hat{\textit{K}} \leftarrow \texttup A(\hat{\mathcal{S}}) \) defined by pullback, \( \hat{\textit{k}} \cdot \textit{t} = \textit{t} \cdot \texttup A(\hat{\textit{h}}) \). \(^{22}\) This defines the \textit{A}-table morphism \( \hat{\mathcal{T}} \xrightarrow{(\hat{\textit{h}}, \hat{\textit{k}})} \mathcal{T} \), which is the output for quotient (Tbl. 3).

Quotient is the one-step process

\[
\text{quotient}(\mathcal{T}) \equiv \textsf{Tbl}_A(\hat{\textit{h}})(\mathcal{T}).
\]

**Aside:** Theoretically, this would represent the equalizer, the limit (see the application discussion for completeness in §A.3) of a parallel pair \( \langle \textit{h}_1, \textit{k}_1 \rangle, \langle \textit{h}_2, \textit{k}_2 \rangle : \mathcal{T} \rightrightarrows \mathcal{T}' \) of \textit{A}-table morphisms. But practically, we are only given the constraint (parallel pair) \( \textit{h}_1, \textit{h}_2 : \mathcal{S} \sqsubseteq \mathcal{S}' \) of \textit{X}-signature morphisms and the input \( \mathcal{T} \) in Tbl. 3. Similar comments, which distinguish the practical from the theoretical, hold for the natural join operation in §4.3.

| \textit{h}_1, \textit{h}_2 : \mathcal{S} \sqsubseteq \mathcal{S}' | \text{constraint} | \hat{\mathcal{S}} \xleftarrow{\textit{h}} \mathcal{S} | \text{construction} | \mathcal{T} \in \textsf{Tbl}_A(\mathcal{S}) | \text{input} | \text{quotient}(\mathcal{T}) \xrightarrow{(\textit{h}, \textit{k})} \mathcal{T} | \text{output} |
|---|---|---|---|---|---|---|
| \textit{h}_1, \textit{h}_2 : \mathcal{S} \sqsubseteq \mathcal{S}' | \text{constraint} | \hat{\mathcal{S}} \xleftarrow{\textit{h}} \mathcal{S} | \text{construction} | \mathcal{T} \in \textsf{Tbl}_A(\mathcal{S}) | \text{input} | \text{quotient}(\mathcal{T}) \xrightarrow{(\textit{h}, \textit{k})} \mathcal{T} | \text{output} |

**Table 3. FOLE Quotient I/O**

---

\(^{22}\) The quotient table \( \hat{\mathcal{T}} \) contains the set of all mergers of tuples in \( \mathcal{T} \) of equivalence classes of sorts.
4.2 Core.

Let \( S \) be a fixed signature. This section discusses the core operation. The core operation is somewhat unorthodox, since it does not have a construction process. However, although it is defined for tables with a fixed signature, it computes a limit-like result: it uses right adjoint flow (restriction) followed by intersection. In fact, it corresponds to the first half of flow along a signed domain morphism followed by intersection. This binary operation gives the core tuples\(^{23}\) in a pair of \( S \)-tables. We use the following routes of flow from Fig. 14.

\[
\text{restrict} \quad \sqcap \quad \text{restrict} \quad \text{meet} \quad \wedge
\]

We first restrict the data types of the tables to the common data values, and then we intersect.\(^{24}\)

**Constraint/Construction:** Consider an \( X \)-sorted type domain opspan \( A_1 \xrightarrow{g_1} A \xleftarrow{g_2} A_2 \) consisting of a span of data value functions \( Y_1 \xrightarrow{g_1} Y \xleftarrow{g_2} Y_2 \).

This is both the constraint and the construction for core (Tbl. 4). We deviate from orthodoxy (limit construction) at this step.

**Input:** Consider a pair of tables \( T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1) \) and \( T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2) \). This is the input for core (Tbl. 4).

**Output:** The output is restriction (twice) followed by meet.

- Restriction \( \text{Tbl}_S(A_1) \xrightarrow{\text{tbll}_S(\delta_1)} \text{Tbl}_S(A) \) (§3.3.2) along the tuple function

\[
\text{of the } X\text{-sorted type domain morphism } A_1 \xrightarrow{g_1} A \text{ maps the } S\text{-table } T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1) \text{ to the } S\text{-table } \text{tbll}_S(\delta)(T_1) = T_1 = \langle K_1, t_1 \rangle \in
\]

\((X\text{-sorted type domain morphism})\)

\[
\begin{align*}
\text{core} \\
\text{restrict} & \quad \sqcap & \quad \text{restrict} \\
\text{meet} & \quad \wedge
\end{align*}
\]

\textbf{Fig. 16. FOLE Core Flow Chart}

\( A \) tuple is core when it appears in both tables and its data values are taken from both data sets.

\( \) The core operation is the dual of the co-core operation.
The core in two-step process \( \text{Tbl}_S(A) \), with its tuple function \( \hat{K}_1 \xrightarrow{\hat{t}_1} \text{tup}_S(A) \) defined by pullback, \( \hat{k}_1 \cdot \hat{t}_1 = \hat{t}_1 \cdot \text{tup}_S(g_1) \). This is linked to the table \( T_1 \) by the \( S \)-table morphism \( T' = \langle A', K', \tau' \rangle \xrightarrow{\langle g_1, k_1 \rangle} \langle \hat{K}_1, \hat{t}_1 \rangle = \hat{T}_1 \). Similarly for \( S \)-table \( T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2) \).

- Intersection (§ 3.2) of the two restriction tables \( \hat{T}_1 \cap \hat{T}_2 \) in the context \( \text{Tbl}_S(A) \) defines the core table \( T_1 \cap_S T_2 = \hat{T}_2 \cap \hat{T}_2 = \langle \hat{K}_1 \times \hat{K}_2, \langle \hat{t}_1, \hat{t}_2 \rangle \rangle \), whose key set \( \hat{K}_1 \times \hat{K}_2 \) is the product and whose tuple map \( \hat{K}_1 \times \hat{K}_2 \to \langle \hat{t}_1, \hat{t}_2 \rangle \), \( \text{tup}_S(A) \) maps a pair of keys \( \langle \hat{k}_1, \hat{k}_2 \rangle \in \hat{K}_1 \times \hat{K}_2 \) to the common tuple \( \hat{t}_1(\hat{k}_1) = \hat{t}_2(\hat{k}_2) \in \text{tup}_A(S) \). Intersection is the product in \( \text{Tbl}_S(A) \) with span \( \hat{T}_1 \xrightarrow{\pi_1} \hat{T}_1 \cap \hat{T}_2 \xrightarrow{\pi_2} \hat{T}_2 \).

Restriction composed with meet defines the span of \( A \)-table morphisms

\[
T_1 \xrightarrow{\langle g_1, k_1 \rangle} \langle g_1, k_1 \rangle \cap_S T_2 \xrightarrow{\langle g_2, k_2 \rangle} T_2,
\]

which is the output for core (Tbl. 4).

Core is restriction followed by meet (conjunction or intersection). This is the two-step process

\[
T_1 \cap_S T_2 \doteq \text{tbl}_S(g_1)(T_1) \land \text{tbl}_S(g_2)(T_2).
\]

| \( A_1 \xrightarrow{g_1} A \xrightarrow{g_2} A_2 \) constraint /construction | \( T_1 \in \text{Tbl}_S(A_1) \) and \( T_1 \in \text{Tbl}_S(A_1) \) input | \( T_2 \xrightarrow{\langle g_1, k_1 \rangle} T_1 \cap_S T_2 \xrightarrow{\langle g_2, k_2 \rangle} T_2 \xrightarrow{\pi_2} T_2 \) output |
| --- | --- | --- |
| \( T_1 \) \( \text{con} \) \( T_2 \) | \( \text{input} \) | \( \text{output} \) |

**Table 4. FOLE Core I/O**

---

25 The core in § 4.2, \( T_1 \xrightarrow{\langle g_1, k_1 \rangle} T_1 \cap_S T_2 \xrightarrow{\langle g_2, k_1 \rangle} T_2 \), is homogeneous with and has a direct connection to both tables \( T_1 \) and \( T_2 \). This is comparable with the co-core (§ 5.2) \( T_1 \xrightarrow{\langle h_1, i_1 \rangle} T_1 \cup_S T_2 \xrightarrow{\langle h_2, i_1 \rangle} T_2 \), which is homogeneous with and has a direct connection to both tables \( T_1 \) and \( T_2 \).
4.3 Natural Join.

Natural Join.  

\[
\begin{array}{c}
\text{inflate} \quad \otimes \quad \text{inflate} \\
\text{meet} \quad \wedge \\
\end{array}
\]

Fig. 17. FOLE Natural Join Flow Chart

The natural join for tables is the relational counterpart of the logical conjunction for predicates. Where the meet operation (§3.2) is the analogue for logical conjunction at the small scope Tbl(D) of a signed domain table fiber, the natural join is defined at the intermediate scope Tbl(A) of a type domain table fiber, and the generic meet (§4.4) is defined at the large scope Tbl of all tables. We identify these three concepts as limits at different scopes.

In this section, we focus on tables in the context Tbl(A) for fixed type domain A. In this context, generic meets — for the special case of pullback — are called natural joins, the join of two A-tables. As we observed in [10], these limits are resolvable into inflations (called substitutions there) followed by meet. We use the following routes of flow from Fig. 14.

\[
\begin{array}{c}
\text{inflate} \quad \otimes \quad \text{inflate} \\
\text{meet} \quad \wedge \\
\end{array}
\]

The natural join operation is dual to the data-type join operation of §5.3. Similar to data-type join, we can define natural join for any number of tables \( \{T_1, T_2, T_3, \ldots, T_n\} \) with a comparable constraint.

**Constraint:** Consider an X-sorted signature span \( S_1 \xrightarrow{h_1} S \xrightarrow{h_2} S_2 \) in List(X) consisting of a span of index functions \( I_1 \xrightarrow{h_1} I \xrightarrow{h_2} I_2 \). This is the constraint for natural join (Tbl. 5).

**Construction:** The pushout of this constraint in List(X) is the opspan \( S_1 \xleftarrow{\iota_1} S_1 + S_2 \xrightarrow{\iota_2} S_2 \) of injection X-signature morphisms with pushout signature.

\[
\text{For a brief discussion of natural join, see §4.4 of “The FOLE Table” [10].}
\]

**Natural join is a limit (meet-like) operation. To fit better with the limit-colimit duality in this paper (see Tbl. 1), we modify the traditional symbol ‘×’ for natural join, using the symbol ‘ tỏ’ instead.**
Natural join is inflation followed by meet. This is the construction for natural join (Tbl. 5).

**Input:** Consider a pair of tables \( T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_A(S_1) \) and \( T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_A(S_2) \). These two tables form an adequate collection (Def. 2) to compute the pullback. This is the input for natural join (Tbl. 5).

**Output:** The output is inflation (twice) followed by meet.

- Inflation \( \text{Tbl}_A(S_1) \xrightarrow{\text{tbl}_A(t_1)} \text{Tbl}_A(S + S S_2) \) (§ 3.3.1) along the tuple function of the \( X \)-signature morphism \( S_1 \xrightarrow{\pi_1} S_1 S S_2 \) maps the table \( T_1 \) to the table \( \text{tbl}_A(t_1)(T_1) = \langle K_1, t_1 \rangle \in \text{Tbl}_A(S_1 S S_2) \), with its tuple function \( K \xrightarrow{t} \text{tup}_A(S_1 + S S_2) \) defined by pullback, \( K \cdot t_1 = t_1 \cdot \text{tup}_A(t_1) \).

  This is linked to the table \( T_1 \) by the \( A \)-table morphism \( T_1 = \langle S_1, K_1, t_1 \rangle \xrightarrow{(s_1, k_1)} \langle S_1 S S_2, K_1, t_1 \rangle = T_1 \). Similarly for \( A \)-table \( T_2 = \langle S_2, K_2, t_2 \rangle \in \text{Tbl}_A(S_2) \).

- Intersection (§ 3.2) of the two inflation tables \( \hat{T}_1 \) and \( \hat{T}_2 \) in the context \( \text{Tbl}_A(S + S S_2) \) defines the natural join table \( T_1 \boxtimes_A T_2 = \hat{T}_1 \wedge \hat{T}_2 = \langle \hat{K}_{12}, (\hat{t}_1, \hat{t}_2) \rangle \), whose key set \( \hat{K}_{12} \) is the pullback and whose tuple map is the mediating function \( \hat{K}_{12} \xrightarrow{(\hat{t}_1, \hat{t}_2)} \text{tup}_A(S_1 + S S_2) \) of the opspan \( \hat{K}_1 \xrightarrow{\hat{t}_1} \text{tup}_A(S_1 + S S_2) \xrightarrow{\hat{t}_2} \hat{K}_2 \), resulting in the span \( \hat{T}_1 \xrightarrow{\hat{s}_1} T_1 \boxtimes_A T_2 \xrightarrow{\hat{s}_2} \hat{T}_2 \).

Inflation composed with meet defines the span of \( A \)-table morphisms

\[
\begin{array}{c}
T_1 \xrightarrow{(s_1, k_1)} T_1 \boxtimes_A T_2 \xrightarrow{(s_2, k_2)} T_2,
\end{array}
\]

which is the output for natural join (Tbl. 5).

Natural join is inflation followed by meet. This is the two-step process

\[
T_1 \boxtimes_A T_2 = \text{tbl}_A(t_1)(T_1) \wedge \text{tbl}_A(t_2)(T_2). \tag{28}
\]

**Aside:** Theoretically this would represent pullback, the limit (see the application discussion for completeness in § A.3) of an opspan \( T_1 \xrightarrow{\langle h_1, k_1 \rangle} T \xrightarrow{\langle h_2, k_2 \rangle} T_2 \) of \( A \)-tables. But practically, we are only given \(^a\) the constraint (span) \( S_1 \xrightarrow{h_1} S \xleftarrow{h_2} S_2 \) of \( X \)-sorted signatures and the input tables \( T_1 \in \text{Tbl}_A(S_1) \) and \( T_1 \in \text{Tbl}_A(S_1) \) in Tbl. 5. Similar comments, which distinguish the practical from the theoretical, hold for the quotient operation in § 4.1.

\(^a\) In practice, the natural join is commonly understood to be the set of all combinations of tuples in \( T_1 \) and \( T_2 \) that are equal on their common attribute names.

\(\text{The natural join is empty, if one of the arguments is empty.}\)
Cartesian Product. The Cartesian product is a special case of the natural join. Let $S_1$ and $S_2$ be two $X$-signatures. These are linked by the span of $X$-signatures $S_1 \overset{0_{t_1}}{\to} S_2 \overset{0_{t_2}}{\to} S_3$ with initial $X$-signature $S_0 = \langle \emptyset, 0_X, X \rangle$ and injection index functions $I_1 \overset{0_{i_1}}{\to} \emptyset \overset{0_{i_2}}{\to} I_2$. This is the constraint for Cartesian product (Tbl. 6). It is a special case of the constraint for Cartesian product. The pushout (colimiting cocone) of this $X$-signature span is the coproduct $X$-signature $S_1 + S_2 = \langle I + I_2, \{s, s_2\} \rangle$ with disjoint union index set $I + I_2$ and injection $X$-signature morphisms (opspan) $S_1 \overset{\cup_{t_1}}{\to} S_1 + S_2 \overset{\cup_{t_2}}{\to} S_2$ with inclusion index functions $I_1 \cup_{i_1} I_1 + I_2 \cup_{i_2} I_2$. The tuple set factors as $\text{tup}_A(S_1 + S_2) \cong \text{tup}_A(S_1) \times \text{tup}_A(S_2)$. This is the construction for Cartesian product (Tbl. 6). Let $T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_A(S_1)$ and $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_A(S_2)$ be two $A$-tables with key sets and a tuple functions $K_1 \overset{t_1}{\to} \text{tup}_A(S_1)$ and $K_2 \overset{t_2}{\to} \text{tup}_A(S_2)$. This is the input for Cartesian product (Tbl. 6). The Cartesian product $T_1 \times T_2$ of the two $A$-tables $T_1$ and $T_2$ is a special case of natural join — just link the tables through the span of tuple functions $\text{tup}_A(S_1) \overset{\text{tbl}_A(t_1)}{\leftarrow} \text{tup}_A(S_1 + S_2) \overset{\text{tbl}_A(t_2)}{\rightarrow} \text{tup}_A(S_2)$, and then use inflation (twice) and intersection. The Cartesian product table $T_1 \times T_2$, which has the binary product key set $K_1 \times K_2$ with product tuple function $\overset{t_1 \times t_2}{\rightarrow} \text{tup}_A(S_1) \times \text{tup}_A(S_2)$, is linked to the component tables with the span of projection table morphisms

$$T \overset{\langle t_1, t_2 \rangle}{\leftarrow} T_1 \times T_2 \overset{\langle t_1, t_2 \rangle}{\rightarrow} T_2.$$ 

This is the output for Cartesian product (Tbl. 6).

**Proposition 4.** Natural join $\times$ distributes over union $\lor$ and intersection $\land$

$$T_1 \times_A (T_2 \lor T_2) \cong (T_1 \times_A T_2) \lor (T_1 \times_A T_2).$$

**Proof.** Inflation is continuous and co-continuous. Intersection $\land$ is distributive over itself and union $\lor$. $\blacksquare$

**Proposition 5.** Natural join is associative

$$(T_1 \times_A T_2) \times_A T_3 \cong T_1 \times_A (T_2 \times_A T_3).$$

The tuple subset of the Cartesian product table $T_1 \times T_2$ is the Cartesian product of the tuple sets $\varphi t_1(K_1) \subseteq \text{tup}_A(S_1) \subseteq \text{List}(Y_1)$ and $\varphi t_2(K_2) \subseteq \text{tup}_A(S_2) \subseteq \text{List}(Y_2)$.

---

**Table 5.** $\text{FOLE}$ Natural Join I/O
Proof. Basic category theory; see Saunders Mac Lane [13] □

Aside: At the intermediate scope, in the context $\text{Tbl}(\mathcal{A})$ of a type domain table fiber, generic meets — for the special case of equalizer — are called quotients, and generic meets — for the special case of pullback — are called natural joins.

Fact 1. (Mac Lane [13]) Limits can be constructed from equalizers and multi-pullbacks.

Proof. Equalizers can be constructed from products and pullbacks. Limits can be constructed from products and equalizers. Binary products are pullbacks from the terminal object. Arbitrary products are iterated binary products. □
4.3.1 Semi-join. Let $\mathcal{A}$ be a fixed type domain. For any two $\mathcal{A}$-tables $T_1 \in \text{Tbl}_\mathcal{A}(S_1)$ and $T_2 \in \text{Tbl}_\mathcal{A}(S_2)$ that are linked through an $X$-sorted signature span $S_1 \xrightarrow{h_1} S \xrightarrow{h_2} S_2$ in $\text{List}(X)$, the left semi-join $T_1 \mid_1 T_2$ is the set of all tuples in $T_1$ for which there is a tuple in $T_2$ that is equal on their common attribute names; the other columns of $T_2$ do not appear. Hence, the left semi-join is defined to be the projection from the natural join. The right semi-join is similar. We use the following routes of flow from Fig. 14.

The constraint, construction and input for semi-join are identical to that for natural join. Only the output is different.

Constraint: The constraint for semi-join is the same as the constraint for natural join (Tbl. 5): an $X$-sorted signature span $S_1 \xrightarrow{h_1} S \xrightarrow{h_2} S_2$ in $\text{List}(X)$.

Construction: The construction for semi-join is the same as the construction for natural join (Tbl. 5): the opspan $S_1 \iota_1 S S_2 \iota_2 S_2$ of injection $X$-signature morphisms with pushout signature $S_1 + S S_2$.

Input: The input for semi-join is the same as the input for natural join (Tbl. 5): a pair of tables $\mathcal{T}_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_\mathcal{A}(S_1)$ and $\mathcal{T}_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_\mathcal{A}(S_2)$.

Output: The output is natural join followed by projection.

- Natural join results in the table $\mathcal{T}_1 \boxtimes_\mathcal{A} \mathcal{T}_2 = tbl_\mathcal{A}(\iota_1)(\mathcal{T}_1) \cap tbl_\mathcal{A}(\iota_2)(\mathcal{T}_2)$ with key set $\hat{K}_{12}$ and tuple function $\hat{t}_{(i_1, i_2)} : \text{tup}_\mathcal{A}(S_1 + S S_2)$.

- Projection $\text{Tbl}_\mathcal{A}(S_1) \xrightarrow{\text{tup}_\mathcal{A}(\iota_1)} \text{Tbl}_\mathcal{A}(S + S S_2)$ (§ 3.3.1) along the tuple function of the $X$-signature morphism $S_1 \xrightarrow{\iota_1} S_1 + S S_2$ maps the natural join table $\mathcal{T}_1 \boxtimes_\mathcal{A} \mathcal{T}_2$ to the left semi-join table $\mathcal{T}_1 \mid_1 \mathcal{T}_2 = \text{tup}_\mathcal{A}(\iota_1)(\mathcal{T}_1) \cap \text{tup}_\mathcal{A}(\iota_2)(\mathcal{T}_2) = \langle \hat{K}_{12}, \hat{t}_{(i_1, i_2)} \rangle$ with key set $\hat{K}_{12}$ and tuple function $\text{tup}_\mathcal{A}(S_1) \xrightarrow{\iota_1} \hat{K}_{12}$ defined by composition $\text{tup}_\mathcal{A}(S_1) \xrightarrow{\text{tup}_\mathcal{A}(\iota_1)} \text{tup}_\mathcal{A}(S_1 + S S_2) \xrightarrow{\iota_2} \hat{K}_{12}$. 

Fig. 18. FOLE Semi-Join Flow Chart
Semi-join is natural join followed by projection. For left semi-join this is the two-step process

$$T_1 \boxdot_4 T_2 = tbl_A(\iota_1)(T_1 \boxdot_2 T_2).$$

This defines the table morphism $T_1 \boxdot_4 T_2 \hookrightarrow\kappa \leftarrow T_1 \boxdot_4 T_2$ in $Tbl(A)$. There is a sub-table relationship $T_1 \hookrightarrow\kappa \leftarrow T_1 \boxdot_4 T_2$ in the small fiber table context $Tbl_A(S_1)$, which is the output of left semi-join. The right semi-join has a similar definition (Tbl. 5).

These factor $T_1 \stackrel{k_1}{\rightarrow} T_1 \boxdot_A T_2 \xrightarrow{(\iota_1,1)} T_1 \boxdot_2 T_2 \xrightarrow{(\iota_2,1)} T_1 \boxdot_4 T_2 \xrightarrow{k_2} T_2$, the span of $A$-table morphisms $T_1 \xrightarrow{k_1 \cdot (\iota_1,1)} T_1 \boxdot_A T_2 \xrightarrow{(\iota_2,k_2)} T_2$, which is the output for natural join. 30

---

30 The semi-join of a Cartesian product of non-empty tables gives either of the tables: the left semi-join gives the left table, and the right gives the right.
4.3.2 Anti-join. The anti-join operations are related to the semi-join operations. The left (right) anti-join of two tables is the complement of the left (right) semi-join. Let $A$ be a fixed type domain. For any two $A$-tables $T_1$ and $T_2$ that are linked through an $X$-sorted signature span $S_1 \xleftarrow{h_1} S \xrightarrow{h_2} S_2$, the left anti-join $T_1 \odot_A T_2$ is the set of all tuples in $T_1$ for which there is no tuple in $T_2$ that is equal on their common attribute names. We use the following routes of flow.

Left anti-join within the context $\text{Tbl}(A)$ is left semi-join, followed by difference. This is the two-step process illustrated in Fig. 19. The constraint, construction and input for anti-join are identical to that for natural join. Only the output is different.

**Constraint:** The constraint for anti-join is the same as the constraint for natural join (Tbl. 5): an $X$-sorted signature span $S_1 \xleftarrow{h_1} S \xrightarrow{h_2} S_2$ in $\text{List}(X)$

**Construction:** The construction for anti-join is the same as the construction for natural join (Tbl. 5): the opspan $S_1 \xleftarrow{h_1} S \xrightarrow{h_2} S_2 \xleftarrow{\iota_1} S_1 + S_2$ of injection $X$-signature morphisms with pushout signature $S_1 + S_2$.

**Input:** The input for anti-join is the same as the input for natural join (Tbl. 5): a pair of tables $T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_A(S_1)$ and $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_A(S_2)$.

**Output:** The output is semi-join followed by difference.

- Left semi-join results in the table $T_1 \boxplus_A T_2 = \text{tbl}_A(t_1)(T_1 \boxplus_A T_2) = \langle \hat{K}_{12}, \hat{t}_1 \rangle$ with key set $\hat{K}_{12}$ and tuple function $\text{tup}_A(S_1) \xleftarrow{t_1} \hat{K}_{12}$.

- Difference in the small table fiber context $\text{Tbl}_A(S_1)$ gives the left anti-join table $T_1 \odot_A T_2 = T_1 - (T_1 \boxplus_A T_2)$.
Anti-join is semi-join, followed by difference. For left anti-join, this is the two-step process

$$T_1 \sqcup_A T_2 \equiv T_1 - (T_1 \sqcup_A T_2).$$

There is an inclusion morphism $$T_1 \xrightarrow{\bar{\omega}_1} T_1 \sqcup_A T_2$$ in the small fiber table context $$\text{Tbl}_A(S_1),$$ which is the output for left anti-join. The right anti-join has a similar definition (Tbl. 5). \(^{31}\)

Aside: There is an alternate method for computing the anti-join. For the left anti-join, project $$T_1$$ and $$T_2$$ to the $$X$$-sorted signature $$S$$ using the $$X$$-sorted signature span $$S_1 \xleftarrow{h_1} S \xrightarrow{h_2} S_2$$ getting $$A$$-tables $$\text{tbl}_A(h_1)(T_1) = \hat{T}_1$$ and $$\text{tbl}_A(h_2)(T_2) = \hat{T}_2.$$ Form the difference $$\hat{T}_{12} = \hat{T}_1 - \hat{T}_2.$$ Then, form the natural join $$T_1 \boxtimes_A \hat{T}_{12}.$$ \(^{a}\) We assume the index functions are injective $$I_1 \xleftarrow{h_1} I \xrightarrow{h_2} I_2.$$

**Proposition 6.** The left anti-join is $$T_1 \sqcup_A T_2 \equiv T_1 \boxtimes_A \hat{T}_{12}.$$ \(^{32}\)

**Proof.** Consider a tuple $$t = (\hat{\alpha}_1, \hat{\alpha}_{12}) \in \text{tup}_A(S_1),$$ with projection sub-tuple $$\hat{\alpha}_{12} = \text{tup}_A(h_1)(t) \in \text{tup}_A(S).$$ Then, $$t \in T_1 \sqcup_A T_2$$ iff $$t \in T_1,$$ $$t \notin T_1 \sqcup_A T_2$$ iff $$t \in T_1,$$ $$\hat{\alpha}_{12} \notin \hat{T}_{12}$$ iff $$t \in T_1 \sqcup_A \hat{T}_{12}.$$ \(\blacksquare\)

\(^{31}\) The anti-join of a Cartesian product of non-empty tables is empty.

\(^{32}\) This argument is in terms of the underlying relations in the reflection of Prop. 1.
4.4 Generic Meet.

The generic meet for tables is the relational counterpart of the logical conjunction for predicates. Where the meet operation (§ 3.2) is the analogue for logical conjunction at the small scope $\text{Tbl}(\mathcal{D})$ of a signed domain table fiber, and the quotient operation (§ 4.1) and the natural join operation (§ 4.3) are special cases of the analogue at the intermediate scope $\text{Tbl}(\mathcal{A})$ of a type domain table fiber, the generic meet operation is defined at the large scope $\text{Tbl}$ of all tables. We identify all of these concepts as limits at different scopes.

In this section, we focus on tables in the full context $\text{Tbl}$ of all tables. These limits are resolvable into restriction-inflations followed by meet. The generic meet operation is dual to the generic join operation (§ 5.4). The generic meet operation only needs a sufficient collection of tables (Def. 2). To reiterate, we identify FOLE generic meets with all limits in the context $\text{Tbl}$.

**Constraint:** Consider a diagram $D : \mathcal{I}^{\text{op}} \to \text{Dom}$ consisting of a linked collection of signed domains $\{\mathcal{D}_i : \langle h, f, g \rangle \to D_j \text{ for } i \in \mathcal{I}\}$. This is the constraint for generic meet (Tbl. 7).

**Construction:** Let $\hat{\mathcal{D}} = \bigsqcup D$ be the colimit in $\text{Dom}$ with injection signed domain morphisms $\{\mathcal{D}_i : \langle \hat{h}_i, \hat{f}_i, \hat{g}_i \rangle \to \hat{\mathcal{D}} \text{ for } i \in \mathcal{I}\}$ that commute with the links in the constraint: $\langle \hat{h}_i, \hat{f}_i, \hat{g}_i \rangle = \langle h, f, g \rangle \circ \langle \hat{h}_j, \hat{f}_j, \hat{g}_j \rangle$. This is the construction for generic meet (Tbl. 7).

**Input:** Let $I \xrightarrow{T} \text{Tbl}$ be a sufficient indexed collection of tables (Def. 2) $\{\mathcal{T}_i = T(i) \in \text{Tbl}(D_i) \text{ for } i \in I\}$ for some indexing set $I \subseteq \text{obj}(\mathcal{I})$. This is the input for generic meet (Tbl. 7).

---

33 Generic joins and colimits in the context of $\text{Tbl}$ (§ 5.4) can be constructed out of limits in the context of signed domains $\text{Dom}$, the table projection-expansion operation along signed domain morphisms, and joins (colimits) in small table fibers.
Output: Generic meet is restriction/inflation \((i \in I\) times) followed by meet.

- For each index \(i \in I\), restriction/inflation \(\text{Tbl}(D_i) \xrightarrow{\text{tbl}(h_i, f_i, g_i)} \text{Tbl}(\hat{D})\) (§3.3.3) along the tuple function of the signed domain morphism \(D_i \xrightarrow{\langle h_i, f_i, g_i \rangle} \hat{D}\), \(\hat{D} = \bigsqcup D\) maps the table \(T_i \in \text{Tbl}(D_i)\) to the table \(\hat{T}_i = \langle \hat{K}_i, \hat{t}_i \rangle \in \text{Tbl}(\hat{D})\) with its tuple function \(\hat{K}_i \xrightarrow{\hat{t}_i} \text{tup}(\hat{D})\) defined by pullback, \(\hat{k}_i \cdot \hat{t}_i = \hat{t}_i \cdot \text{tup}(h_i, f_i, g_i)\). Here we have “vertically restricted” and then “horizontally inflated” tuples in \(\text{tup}(D_i) \subseteq \text{List}(\hat{D})\) by pullback along the tuple function \(\text{tup}(D_i) \xleftarrow{\text{tup}(h_i, f_i, g_i)} \text{tup}(\hat{D})\) (see RHS Fig. 11). This is linked to the table \(\hat{T}_i\) by the table morphism \(\hat{T}_i = \langle \hat{K}_i, \hat{t}_i \rangle \xrightarrow{\langle h_i, f_i, g_i \rangle, \hat{k}_i} \langle \hat{K}_i, \hat{t}_i \rangle = \hat{T}_i\).

- Intersection (§3.2) of the tables \(\{\hat{T}_i \mid i \in I\}\) in the fiber context \(\text{Tbl}(\hat{D})\) defines the generic meet \(\bigsqcup T = T = \bigwedge\{\hat{T}_i \mid i \in I\} = \langle \hat{K}, \hat{t} \rangle\), whose key set \(\hat{K}\) is the pullback and whose tuple map is the mediating function \(\hat{K} \xrightarrow{(\hat{t})} \text{tup}(\hat{D})\) of the multi-opspan \(\{\hat{K}_i \xrightarrow{\hat{t}_i} \text{tup}(\hat{D}) \mid i \in I\}\), resulting in the discrete multi-span (cone) \(\{\hat{T}_i \xleftarrow{} \hat{T} \mid i \in I\}\).

Restriction-inflation composed with meet defines the multi-span of table morphisms

\[
\{T_i \xleftarrow{\langle h_i, f_i, g_i \rangle, \hat{k}_i} \bigsqcup T = \hat{T} \mid i \in I\},
\]

illustrated in Fig. 21, which is the output for generic meet (Tbl. 7).

Generic meet is restriction/inflation \((i \in I\) times) followed by meet. This is the two-step process

\[\bigsqcup T = \bigwedge \{\text{tbl}(h_i, f_i, g_i)(T_i) \mid i \in I\}.\]

Aside: Theoretically this would represent the limit of a diagram \(I \xrightarrow{T} \text{Tbl}\) consisting of a linked collection of tables. But practically, we are only given the constraint (a diagram) \(I^\text{op} \xrightarrow{D} \text{Dom}\) consisting of a linked collection of signed domains \(\{D_i = D(i) \mid i \in I\}\) and the input \(I \xrightarrow{T} \text{Tbl}\) consisting of a sufficient indexed collection of tables (Def. 2) \(\{T_i = T(i) \in \text{Tbl}(D_i) \mid i \in I \subseteq \text{obj}(I)\}\).
Table 7. FOLE Generic Meet I/O

Fig. 21. FOLE Generic Meet
5 Composite Operations for Colimits.

To repeat, the basic components of §3 are the components to be used in flowcharts. Composite operations are operations whose flowcharts are composed of one or more basic components. In addition to its basic components, a composite operation also has a constraint, which is used to construct its output. In this section we define the the composite relational operations (Tbl. 8) related to colimits. Each composite operation defined here has a dual relational operation (Tbl. 2) related to limits. For colimit operations we need only a sufficient collection of tables linked by a complete collection of signatures (see Def. 3). Fig. 22 gives the possible routes of flow for colimits.

| Operation          | Expression                                                                 | In ( ) | Table 8. FOLE Composite Relational Operations for Colimits |
|--------------------|-----------------------------------------------------------------------------|--------|----------------------------------------------------------|
| co-quotient        | $\phi_T = tbl_A(\hat{g})(T)$                                               | $\in$  | $\text{Tbl}_A(\hat{A})$                                  |
| co-core            | $T_1 \cap A T_2 = tbl_A(h_1)(T_1) \cap tbl_A(h_2)(T_2) \in$ $\text{Tbl}_A(S)$ |
| data-type join     | $T_1 \oplus S T_2 = tbl_S(g_1)(T_1) \cup tbl_S(g_2)(T_2) \in$ $\text{Tbl}_S(A_1 \times A_2)$ |
| data-type semi-join| $T_1 \oplus S T_2 = tbl_S(g_1)(T_1) \cup tbl_S(g_2)(T_2) \in$ $\text{Tbl}_S(A_1)$ |
| data-type anti-join| $T_1 \ominus S T_2 = T_1 - (T_1 \oplus S T_2) \in$ $\text{Tbl}_S(A_1)$ |
| generic join       | $\bigvee \{ tbl(h_i, f_i, g_i)(T_i) | i \in I \} \in$ $\text{Tbl}(D)$ |

Fig. 22. Routes of Flow: Colimits

---

34 The co-quotient operation is dual to the quotient operation. The co-core operation is dual to the core operation. The data-type join operation is dual to the natural join operation. The generic join operation is dual to the generic meet operation.
5.1 Co-quotient.

In this section, we focus on tables in the context $\text{Tbl}(S)$ for fixed signature $\mathcal{S}$. In this context, generic joins — for the special case of the co-equalizer of a parallel pair — are called co-quotients. Here, we define an equivalence on data values. In §4.1 we discuss a dual notion; there we define an equivalence on attributes, specifically on indexes. The co-quotient operation defined here is dual to the quotient operation defined in §4.1.

**Constraint:** Consider a parallel pair $g_1, g_2 : A \Rightarrow A'$ of $X$-type domain morphisms in $\text{Cls}(X)$. This is the constraint for co-quotient (Tbl. 9).

**Construction:** We can construct the equalizer of this constraint in $\text{Cls}(X)$ with co-quotient $X$-type domain $A$ along $\text{tup}_S(\mathcal{I})$ defined by composition, $\tilde{t} = t \cdot \text{tup}_S(\mathcal{I})$. This defines the $S$-table morphism $\tilde{T} : (\mathcal{I}) \rightarrow T$, which is the output for co-quotient (Tbl. 9).

**Input/Output:** Consider a table $T = \langle K, t \rangle \in \text{Tbl}_S(A)$. This table forms an adequate collection (Def. 3) to compute the coequalizer. This is the input for co-quotient (Tbl. 9). The output is computed with one expansion.

- Expansion $\text{Tbl}_S(\mathcal{I}) \leftrightarrow sbs([R]) \text{Tbl}_S(A)$ along the tuple function of the $X$-type domain morphism $\mathcal{I} \mapsto \mathcal{I} \rightarrow A$ maps the table $T$ to the table $\tilde{T} = \text{Tbl}_S([R])(T) = \langle K, \tilde{t} \rangle \in \text{Tbl}_S(A)$, with its tuple function $K \mapsto \text{tup}_S(\mathcal{I})$ defined by composition, $\tilde{t} = t \cdot \text{tup}_S([R])$. This defines the $S$-table morphism $\tilde{T} : (\mathcal{I}) \rightarrow T$, which is the output for co-quotient (Tbl. 9).

---

Basic components (§3) are components to be used in flowcharts. In particular, the co-quotient composite operation of this section has a flowchart with only one component — expansion. In addition to its one component, it also has a constraint, which is used to construct its output.

The data value functions in the constraint satisfy the invariant $\mathcal{I} = \langle X, R \rangle$ consisting of the full subset of sorts $X$ and the equivalence relation of data values $R = \{(g_1(y'), g_2(y')) | y' \in Y' \} \subseteq Y \times Y$ defined by the expression $g_1(y') \models_A x \iff y' \models_A x \iff g_2(y') \models_A x$ for the parallel pair in the constraint. The co-quotient $X$-type domain $\mathcal{I} = \langle X, R, \models_A(\mathcal{I}) \rangle$ has as data values the $R$-equivalence classes of data values in $Y'$. (invariants/quotients are discussed in [1])
Co-quotient is the one-step process

$$\varsigma_S(T) \equiv tbl_S([[R]])(T).$$

**Aside:** Theoretically this would represent the co-equalizer, the colimit (see the application discussion for co-completeness in §A.3) of a parallel pair $\langle g_1, k_1 \rangle, \langle g_2, k_2 \rangle : T \rightrightarrows T'$ of $S$-table morphisms. But practically, we are only given the constraint (parallel pair) $g_1, g_2 : A \rightrightarrows A'$ of $X$-type domain morphisms and the input $T$ in Tbl. 9. Similar comments, which distinguish the practical from the theoretical, hold for the data-type join operation in §5.3.

| $g_1, g_2 : A \rightrightarrows A'$ | constraint |
| $\mathcal{A}/\mathcal{I}_{rs}$, $\mathcal{A}$ | construction |
| $\mathcal{T} \in \text{Tbl}_S(\mathcal{A})$ | input |
| $\varsigma_S(T)$, $([[R]])(T)$ | output |

**Table 9. FOLE Co-quotient I/O**
5.2 Co-core.

Let $\mathcal{A}$ be a fixed type domain. This section discusses the co-core operation. The co-core operation is somewhat unorthodox, since it does not have a construction process. However, although it is defined for tables with a fixed type domain, it computes a colimit-like result: it uses left adjoint flow (projection) followed by union. In fact, it corresponds to the first half of flow along a signed domain morphism followed by union. We use the following routes of flow from Fig. 22.

The idea here is to get rid of some of the unused indexes, thereby getting rid of some of the unused data types. This is comparable to the filtered join for signatures in §6.3, where we want to get rid of some of the unreliable or inauthentic data values. However, co-core is an orthodox operation since it does follow the construction of a limit, whereas filtered join is an unorthodox operation since it does not follow the construction of a colimit.

**Constraint/Construction:** Consider an $X$-sorted signature span $S_1 \xrightarrow{h_1} S \xrightarrow{h_2} S_2$ consisting of a span of index functions $I_1 \xrightarrow{h_1} I \xrightarrow{h_2} I_2$. This is both the constraint and the construction for co-core (Tbl. 10). We deviate from orthodoxy (colimit construction) at this step.

**Input:** Consider a pair of tables $T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_A(S_1)$ and $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_A(S_2)$. This is the input for co-core (Tbl. 10).

**Output:** The output is projection (twice) followed by join.

---

37 A header is co-core when it appears in both tables and its index values are taken from either index set.

38 The co-core operation is the dual of the core operation.
• Projection $T_{bl_A(S_1)} \xrightarrow{\text{tbl}_A(h_1)} T_{bl_A(S)}$ (§ 3.3.1) along the tuple function of the X-signature morphism $S_1 \xrightarrow{h_1} S$ maps the $A$-table $T_1$ to the $A$-table $\tilde{T}_1 = \text{tbl}_A(h_1)(T_1) = \langle K_1, t_1 \rangle \in T_{bl_A(S)}$, with its tuple function $K_1 \xrightarrow{t_1} \text{tup}_A(S)$ defined by composition, $\tilde{t}_1 = t_1 \cdot \text{tup}_A(h_1)$. This is linked to the table $T_1$ by the $A$-table morphism $\tilde{T}_1 = \langle S, K_1, \tilde{t}_1 \rangle$. Similarly for $A$-table $T_2 = \langle K_2, t_2 \rangle \in T_{bl_A(S_2)}$.

• Union (§ 3.2) of the two projection tables $\tilde{T}_1$ and $\tilde{T}_2$ in the context $T_{bl_A(S)}$ defines the co-core table $T_1 \cup_A T_2 = \tilde{T}_1 \cup \tilde{T}_2 = \langle K_1 + K_2, [\tilde{t}_1, \tilde{t}_2] \rangle$, whose key set $K_1 + K_2$ is the disjoint union and whose tuple map $\tilde{T}_1 + \tilde{T}_2 \xrightarrow{[\tilde{t}_1, \tilde{t}_2]} \text{tup}_A(S)$ maps $k_1 \in K_1$ to $\tilde{t}_1(k_1) \in \text{tup}_A(S)$ and maps $k_2 \in K_2$ to $\tilde{t}_2(k_2) \in \text{tup}_A(S)$. Union is the coproduct in $T_{bl_A(S)}$ with opspan $\tilde{T}_1 \xrightarrow{\tilde{t}_1} \tilde{T}_1 \cup \tilde{T}_2 \xrightarrow{\tilde{t}_2} \tilde{T}_2$.

Projection composed with join defines the opspan of $A$-table morphisms

$$
\begin{align*}
T_1 \xrightarrow{(h_1, \tilde{t}_1)} T_1 \cup_A T_2 \xrightarrow{(h_2, \tilde{t}_2)} T_2,
\end{align*}
$$

which is the output for co-core (Tbl. 10).

Co-core is projection followed by join (disjunction or union). This is the two-step process

$$
T_1 \cup_A T_2 \doteq \text{tbl}_A(h_1)(T_1) \cup \text{tbl}_A(h_2)(T_2). \quad 39
$$

| $S_1 \xrightarrow{h_1} S \xrightarrow{h_2} S_2$ | constraint /construction |
|---------------------------------|-------------------------|
| $T_1 \in T_{bl_A(S_1)}$ and $T_2 \in T_{bl_A(S_2)}$ | input |
| $T_1 \xrightarrow{(h_1, \tilde{t}_1)} T_1 \cup_A T_2 \xrightarrow{(h_2, \tilde{t}_2)} T_2$ | output |

**Table 10. FOLE Co-core I/O**

39 The co-core $T_1 \xrightarrow{(g_1, \tilde{t}_1)} T_1 \cup_A T_2 \xleftarrow{(g_2, \tilde{t}_2)} T_2$, is homogeneous with and has a direct connection to both tables $T_1$ and $T_2$. This is comparable with the core (§ 4.2) $T_1 \xleftarrow{(g_1, \tilde{t}_1)} T_1 \cap_S T_2 \xrightarrow{(g_2, \tilde{t}_2)} T_2$, which is homogeneous with and has a direct connection to both tables $T_1$ and $T_2$. 
5.3 Data-type Join.

The data-type join for tables is the relational counterpart of the logical disjunction for predicates. Where the join operation is the analogue for logical disjunction at the small scope $\text{Tbl}(D)$ of a signed domain table fiber, the data-type join is defined at the intermediate scope $\text{Tbl}(S)$ of a signature table fiber, and the generic join (§5.4) is defined at the large scope $\text{Tbl}$ of all tables. We identify these three concepts as colimits at different scopes.

In this section, we focus on tables in the context $\text{Tbl}(S)$ for fixed signature (header) $S$. In this context, generic joins — for the special case of pushout — are called data-type joins, the join of two $S$-tables. These colimits are resolvable into expansions followed by join. We use the following routes of flow from Fig. 22.

The data-type join operation is dual to the natural join operation of §4.3. Similar to natural join, we can define data-type join for any number of tables $\{T_1, T_2, T_3, \ldots, T_n\}$ with a comparable constraint.

**Constraint:** Consider an $X$-sorted type domain opspan $A_1 \xrightarrow{g_1} A \xleftarrow{g_2} A_2$ in $\text{Cls}(X)$ consisting of a span of data value functions $Y_1 \xrightarrow{g_1} Y \xleftarrow{g_2} Y_2$. This is the constraint for data-type join (Tbl. 11).

**Construction:** The pullback of this constraint in $\text{Cls}(X)$ is the span $A_1 \xleftrightarrow{g_1} A_2 \xleftrightarrow{g_2} A_2$ of projection $X$-type domain morphisms with pullback type domain $A_1 \times_A A_2$ and data value function opspan $Y_1 \xrightarrow{g_1} Y_1 +_Y Y_2, [g_1, g_2] \xleftarrow{g_2} Y_2$. This is the construction for data-type join (Tbl. 11).

**Input:** Consider a pair of tables $T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1)$ and $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2)$. These two tables form an adequate collection (Def. 3) to compute the pushout. This is the input for data-type join (Tbl. 11).
Output: The output is expansion (twice) followed by join.

- Expansion $\text{Tbl}_S(A_1) \xrightarrow{tbl_S(\tilde{g}_1)} \text{Tbl}_S(A_1 \times_A A_2)$ along the tuple function of the $X$-type domain morphism $A_1 \xrightarrow{\tilde{g}_1} A_1 \times_A A_2$ maps the $S$-table $T_1$ to the $S$-table $\tilde{T}_1 = tbl_S(\tilde{g}_1)(T_1) = \langle K_1, \tilde{t}_1 \rangle \in \text{Tbl}_S(A_1 \times_A A_2)$, with its tuple function $K_1 \xrightarrow{\tilde{t}_1} \text{tup}_S(A_1 \times_A A_2)$ defined by composition, $\tilde{t}_1 = t_1 \cdot \text{tup}_S(\tilde{g}_1)$. This is linked to the table $T_1$ by the $S$-table morphism $T_1 = \langle A_1, K_1, t_1 \rangle \xrightarrow{\langle 1, \tilde{t}_1 \rangle} \langle A_1 \times_A A_2, K_1, \tilde{t}_1 \rangle = \tilde{T}_1$. Similarly for $S$-table $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2)$.

- Union (§3.2) of the two expansion tables $\tilde{T}_1$ and $\tilde{T}_2$ in the context $\text{Tbl}_S(A_1 \times_A A_2)$ defines the data-type join $T_1 \oplus_S T_2 = \tilde{T}_1 \cup \tilde{T}_2 = \langle K_1 + K_2, \tilde{t}_1, \tilde{t}_2 \rangle$,

  $\text{tup}_S(S)$ is the comediator of the opspan $K_1 \xrightarrow{\tilde{t}_1} \text{tup}_S(A_1 \times_A A_2) \xrightarrow{\tilde{t}_2} K_2$,

resulting in the opspan $\tilde{T}_1 \xrightarrow{\tilde{t}_1} T_1 \oplus_S T_2 \xrightarrow{\tilde{t}_2} \tilde{T}_2$.

Expansion composed with join defines the opspan of $S$-table morphisms

\[ T_1 \xrightarrow{\langle \tilde{g}_1, \tilde{t}_1 \rangle} T_1 \oplus_S T_2 \xleftarrow{\langle \tilde{g}_2, \tilde{t}_2 \rangle} T_2, \]

which is the output for data-type join (Tbl. 11).

Data-type join is expansion followed by join. This is the two-step process

\[ T_1 \oplus_S T_2 \triangleq tbl_S(\tilde{g}_1)(T_1) \cup tbl_S(\tilde{g}_2)(T_2). \]

Aside: Theoretically this would represent pushout, the colimit (see the application discussion for co-completeness in §A.3) of an span $T_1 \xleftarrow{\langle \tilde{g}_1, \tilde{t}_1 \rangle} T_2 \xrightarrow{\langle \tilde{g}_2, \tilde{t}_2 \rangle}$ of $S$-tables. But practically, we are only given the constraint (opspan) $A_1 \xrightarrow{\tilde{g}_1} A_2$ of $X$-indexed type domains and the input tables $T_1 \in \text{Tbl}_S(A_1)$ and $T_1 \in \text{Tbl}_S(A_1)$ in Tbl. 11. Similar comments, which distinguish the practical from the theoretical, hold for the co-quotient operation in §5.1.

\[ \text{The data-type join contains the set of all tuples in } T_1 \text{ and } T_2, \text{ considered as taken from the data value set } Y_1 + Y_2. \]
An opspan of $S$-type domains

$$A_1 = \langle X, Y_1, \models_{A_1}, \langle (X, \gamma_1) \mapsto (X, Y_2, \models_{A_2} \rangle = A_1 \times_{A_2} A_2 \rangle$$

has as its pullback the span of $S$-type domain morphisms.

$$A_1 = \langle X, Y_1, \models_{A_1}, \langle (X, \gamma_1) \mapsto (X, Y_2, \models_{A_2} \rangle = A_1 \times_{A_2} A_2 \rangle$$

This consists of an opspan of data value functions $Y_1 \mapsto Y_1 + Y_2 \mapsto Y_2$ satisfying the condition $\text{ext}_{A_1 \times_{A_2} A_2} \cdot g_i^{-1} = \text{ext}_{A_1}$; or that $g_i^{-1}(A_1 \times_{A_2} A_2) = (A_1, x)$ for all $x \in X$. This implies that $(A_1 \times_{A_2} A_2) \models \gamma_{g_1}(A_1, x)$ for all $x \in X$. Hence, for an injective data value function $Y_1 + Y_2 \mapsto Y_1$, we have the inclusion $(A_1 \times_{A_2} A_2) \models A_1$ for all $x \in X$. Same for an injective data value function $Y_1 + Y_2 \mapsto Y_2$. Thus, for every sort $x \in X$ the direct images of data-types $A_{1x}$ and $A_{2x}$ are contained in data-type $(A_1 \times_{A_2} A_2)_x$.
Disjoint Sum. The disjoint sum is a special case of the data-type join. Let $A_1$ and $A_2$ be two $X$-type domains. These are linked by an opspan of $X$-type domains $A_1 \xrightarrow{\tilde{\sigma}_1} A_\top \xleftarrow{\tilde{\sigma}_2} A_2$ with the terminal $X$-type domain $A_\top = \langle X, \emptyset, \models_A \rangle$ and injection data value functions $Y_1 \xrightarrow{\tilde{\sigma}_1} \emptyset \xleftarrow{\tilde{\sigma}_2} Y_2$. This is the constraint for disjoint sum (Tbl. 12). It is a special case of the constraint for data-type join. The pullback (limiting cone) of this $X$-type domain opspan is the product $X$-type domain $A_1 \times A_2 = \langle X, Y_1 + Y_2, \models_{A_1 \times A_2} \rangle$ with the disjoint sum data value set $Y_1 + Y_2$ and projection $X$-type domain morphisms (span) $A_1 \xrightarrow{\tilde{\sigma}_1} A_1 \times A_2 \xrightarrow{\tilde{\sigma}_2} A_2$ with inclusion data value functions $Y_1 \xrightarrow{\tilde{\sigma}_1} Y_1 + Y_2 \xleftarrow{\tilde{\sigma}_2} Y_2$.

This is the construction for disjoint sum (Tbl. 12). Let $T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1)$ and $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2)$ be two $S$-tables with key sets and a tuple function $K_1 \xrightarrow{\tilde{\sigma}_1} \text{tup}_S(A_1)$ and $K_2 \xrightarrow{\tilde{\sigma}_2} \text{tup}_S(A_2)$. This is the input for disjoint sum (Tbl. 12). The disjoint sum $T_1 + T_2$ of the two $S$-tables $T_1$ and $T_2$ is a special case of data-type join — just link the tables through the opspan of tuple functions $\text{tup}_S(A_1) \xrightarrow{\text{tbl}_S(\tilde{\sigma}_1)} \text{tup}_S(A_1 \times A_2) \xleftarrow{\text{tbl}_S(\tilde{\sigma}_2)} \text{tup}_S(A_2)$, and then use expansion (twice) and union. The disjoint sum table $T_1 + T_2$, which has the binary sum (disjoint union) key set $K_1 + K_2$ with co-mediating tuple function $K_1 + K_2 \xrightarrow{[	ilde{\sigma}_1, \text{tbl}_S(\tilde{\sigma}_1), \tilde{\sigma}_2, \text{tbl}_S(\tilde{\sigma}_2)]} \text{tup}_A(S_1) \times \text{tup}_A(S_2)$, is linked to the component tables with the opspan of table morphisms

$$\text{tup}_A(S_1) \xrightarrow{\tilde{\sigma}_1, \text{tbl}_S(\tilde{\sigma}_1)} T_1 \xleftarrow{\tilde{\sigma}_2, \text{tbl}_S(\tilde{\sigma}_2)} T_2.$$ 

This is the output for disjoint sum (Tbl. 12).

| $A_1$ and $A_2$ | constraint |
|------------------|------------|
| $A_1 \xrightarrow{\tilde{\sigma}_1} A_1 \times A_2 \xrightarrow{\tilde{\sigma}_2} A_2$ | $\text{tup}_S(A_1) \xrightarrow{\text{tbl}_S(\tilde{\sigma}_1)} \text{tup}_S(A_1 \times A_2) \xleftarrow{\text{tbl}_S(\tilde{\sigma}_2)} \text{tup}_S(A_2)$ |

| $T_1 \in \text{Tbl}_S(A_1)$ and $T_2 \in \text{Tbl}_S(A_2)$ | input |
|-----------------------------|--------|
| $T_1 \xrightarrow{\tilde{\sigma}_1, \text{tbl}_S(\tilde{\sigma}_1)} T_1 + T_2 \xleftarrow{\tilde{\sigma}_2, \text{tbl}_S(\tilde{\sigma}_2)} T_2$ | output |

Table 12. FOLE Disjoint Sum I/O

\[41\] The tuple subset of the disjoint sum table $T_1 + T_2$ is the disjoint union of the tuple sets $\phi t_1(K_1) \subseteq \text{tup}_S(A_1) \subseteq \text{List}(Y_1)$ and $\phi t_2(K_2) \subseteq \text{tup}_S(A_2) \subseteq \text{List}(Y_2)$. 

Relational Operations in FOLE 47
5.3.1 Data-type Semi-join. Let \( S \) be a fixed signature. For any two \( S \)-tables \( T_1 \in \text{Tbl}_S(A_1) \) and \( T_2 \in \text{Tbl}_S(A_2) \) that are linked through an \( X \)-sorted type domain opspan \( A_1 \xrightarrow{\phi_1} A \xleftarrow{\phi_2} A_2 \) in \( \text{Cls}(X) \), the left data-type semi-join \( T_1 \circledast_S T_2 \) is the set of all tuples in \( T_1 \) plus tuples in \( T_2 \) with data values in \( Y_1 \). Hence, the left data-type semi-join is defined to be the restriction on the data-type join. The right data-type semi-join is similar. We use the following routes of flow from Fig. 22.

The constraint, construction and input for data-type semi-join are identical to that for natural join. Only the output is different.

**Constraint:** The constraint for data-type semi-join is the same as the constraint for data-type join (Tbl. 11): an \( X \)-sorted type domain opspan \( A_1 \xrightarrow{\phi_1} A \xleftarrow{\phi_2} A_2 \) in \( \text{Cls}(X) \).

**Construction:** The construction for data-type semi-join is the same as the construction for data-type join (Tbl. 11): the span \( A_1 \xleftarrow{\phi_1} A_1 \times_A A_2 \xrightarrow{\phi_2} A_2 \) of projection \( X \)-type domain morphisms with pullback type domain \( A_1 \times_A A_2 \).

**Input:** The input for data-type semi-join is the same as the input for data-type join (Tbl. 11): two tables \( T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1) \) and \( T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2) \).
**Output:** The output is data-type join followed by restriction.

- Data-type join results in the table $T_1 \oplus_S T_2 = tbl_S(\tilde{g}_1)(T_1) \vee tbl_S(\tilde{g}_2)(T_2)$ with key set $K_1 + K_2$ and tuple function $K_1 + K_2 \xrightarrow{[\tilde{f}_1, \tilde{f}_2]} \operatorname{tup}_A(S)$.

- Restriction $\xrightarrow{d_{\tilde{g}_1}} Tbl_S(A_1) \xleftarrow{Tbl_S(A_1 \times_A A_2)}$ (§ 3.3.2) along the tuple function of the $X$-type domain morphism $A_1 \xleftarrow{\tilde{g}_1} A_1 \times_A A_2$ maps the data-type join table $T_1 \oplus_S T_2$ to the left semi-join table $T_1 \oplus_S T_2 = tbl_S(\tilde{g}_1)(T_1 \oplus_S T_2) = \langle \hat{K}, \hat{t}_1 \rangle$ with key set $\hat{K}_1$ and tuple function $\hat{K}_1 \xrightarrow{\hat{f}_1} \operatorname{tup}_S(A_1)$ defined by pullback.

Data-type semi-join is data-type join followed by restriction. For left semi-join, this is the two-step process

$$T_1 \oplus_S T_2 = tbl_S(\tilde{g}_1)(T_1 \oplus_S T_2).$$

This defines the table morphism $T_1 \oplus_S T_2 \xrightarrow{\langle \tilde{g}_1, \tilde{K}_1 \rangle} T_1 \oplus_S T_2$. in $Tbl_S(A_1)$. There is a sub-table relationship $\xrightarrow{\tilde{m}_1} T_1 \oplus_S T_2$ in the small fiber table context $Tbl_S(A_1)$, which is the (left-side) output of data-type semi-join. The right data-type semi-join has a similar definition (Tbl. 11).

These factor $T_1 \xrightarrow{\tilde{m}_1} T_1 \oplus_S T_2 \xrightarrow{\langle \tilde{g}_1, \tilde{K}_1 \rangle} T_1 \oplus_S T_2 \xleftarrow{Tbl_S(A_1 \times_A A_2)} T_2 \xrightarrow{\tilde{m}_2} T_2$, the opspan of $S$-table morphisms $T_1 \xrightarrow{\tilde{m}_1 \circ \langle \tilde{g}_1, \tilde{K}_1 \rangle} T_1 \oplus_S T_2$ and $T_1 \oplus_S T_2 \xleftarrow{\tilde{m}_2 \circ \langle \tilde{g}_2, \tilde{K}_2 \rangle} T_2$, the output for data-type join.\(^{43}\)

---

\(^{42}\) The data-type semi-join of a disjoint sum gives one of the tables.

\(^{43}\) The data-type semi-join of a disjoint sum of non-empty tables gives either of the tables: the left data-type semi-join gives the left table, and the right gives the right.
5.3.2 Data-type Anti-join. The data-type anti-join operations are related to the data-type semi-join operations. The left (right) data-type anti-join of two tables is the complement of the left (right) data-type semi-join. Let $\mathcal{S}$ be a fixed signature. For any two $\mathcal{S}$-tables $T_1$ and $T_2$ that are linked through an $X$-sorted type domain opspan $A_1 \xrightarrow{\varphi_1} \mathcal{A} \xrightarrow{\varphi_2} A_2$, the left data-type anti-join $T_1 \odot_{\mathcal{S}} T_2$ is the set of all tuples in $T_1$ for which there is no tuple in $T_2$ equal on their common data values. We use the following routes of flow.

Left data-type anti-join within the context $\text{Tbl}(\mathcal{S})$ is left data-type semi-join, followed by difference. This is the two-step process illustrated in Fig. 27. The constraint, construction and input for data-type anti-join are identical to that for data-type join. Only the output is different.

**Constraint:** The constraint for data-type anti-join is the same as the constraint for data-type join (Tbl. 11): an $X$-sorted type domain opspan $A_1 \xrightarrow{\varphi_1} \mathcal{A} \xrightarrow{\varphi_2} A_2$ in $\text{Cls}(X)$.

**Construction:** The construction for data-type anti-join is the same as the construction for data-type join (Tbl. 11): the span $A_1 \xleftarrow{\varphi_1} A_1 \times A_2 \xrightarrow{\varphi_2} A_2$ of projection $X$-type domain morphisms with pullback type domain $A_1 \times A_2$.

**Input:** The input for data-type anti-join is the same as the input for data-type join (Tbl. 11): a pair of tables $T_1 \in \text{Tbl}_{\mathcal{S}}(A_1)$ and $T_2 \in \text{Tbl}_{\mathcal{S}}(A_2)$.
Output: The output for data-type anti-join is semi-join followed by difference.

- Left data-type semi-join results in the table $T_1 \oplus_S T_2 = \text{tbl}_S(g_1)(T_1 \oplus_S T_2) = \langle \hat{K}, \hat{t}_1 \rangle$ with key set $\hat{K}_1$ and tuple function $\hat{K}_1 \xrightarrow{\hat{t}_1} \text{tup}_S(A_1)$.

- Difference in the small table fiber context $\text{Tbl}_S(A_1)$ gives the left data-type anti-join table $T_1 \ominus_S T_2 = T_1 - (T_1 \oplus_S T_2)$.

Data-type anti-join is data-type semi-join followed by difference. For left data-type anti-join, this is the two-step process

$$T_1 \ominus_S T_2 = T_1 - (T_1 \oplus_S T_2).$$

There is an inclusion morphism $T_1 \xrightarrow{\omega_1} T_1 \ominus_S T_2$ in the small table fiber context $\text{Tbl}_S(A_1)$, which is the output for left data-type anti-join. The right data-type anti-join has a similar definition (Tbl 11).

---

44 The data-type anti-join of a disjoint sum is the empty table.
5.4 Generic Join.

The generic join for tables is the relational counterpart of the logical disjunction for predicates. Where the join operation (§3.2) is the analogue for logical disjunction at the small scope $\text{Tbl}(D)$ of a signed domain table fiber, and the co-quotient operation (§5.1) and the data-type join operation (§5.3) are special cases of the analogue at the intermediate scope $\text{Tbl}(S)$ of a signature table fiber, the generic join operation is defined at the large scope $\text{Tbl}$ of all tables. We identify all of these concepts as colimits at different scopes.

In this section, we focus on tables in the full context $\text{Tbl}$ of all tables. These colimits are resolvable into expansion-projections followed by join. The generic join operation is dual to the generic meet operation (§4.4). The generic join operation only needs a sufficient collection of tables (Def. 3). To reiterate, we identify FULE generic joins with all colimits in the context $\text{Tbl}$.

Constraint: Consider a diagram $D : I^{op} \rightarrow \text{Dom}$ consisting of a linked collection of signed domains $\{D_i \buildrel \langle h, f, g \rangle \over \rightarrow D_j \}$. This is the constraint for generic join (Tbl. 13).

Construction: Let $\bar{D} = \int D$ be the limit in $\text{Dom}$ with projection signed domain morphisms $\{D_i \buildrel \langle h, f, g \rangle \over \rightarrow \bar{D} \mid i \in I \}$ that commute with the links in the constraint: $\langle h_i, f_i, g_i \rangle \circ \langle h, f, g \rangle = \langle h_j, f_j, g_j \rangle$. This is the construction for generic join (Tbl. 13).

Input: Let $I \xrightarrow{\mathcal{T}} \text{Tbl}$ be a sufficient indexed collection of tables (Def. 3) $\{\mathcal{T}_i = T(i) \in \text{Tbl}(D_i) \mid i \in I \}$ for some indexing set $I \subseteq \text{obj}(I)$. This is the input for generic join (Tbl. 13).

---

45 Generic meets and limits in the context of $\text{Tbl}$ (§4.4) can be constructed out of colimits in the context of signed domains $\text{Dom}$, the table restriction-inflation operation along signed domain morphisms, and meets (limits) in small table fibers.
Output: Generic join is projection/expansion \((i \in I\) times) followed by join.

- For each index \(i \in I\), projection/expansion \(\text{Tbl}(\mathcal{D}_i) \xrightarrow{\text{thl}(\tilde{h}_i, \tilde{f}_i, \tilde{g}_i)} \text{Tbl}(\tilde{D})\) (§ 3.3.3) along the tuple function of the signed domain morphism \(\mathcal{D}_i \xrightarrow{\langle \tilde{h}_i, \tilde{f}_i, \tilde{g}_i \rangle} \tilde{D} = \bigsqcap D\) maps the table \(T_i \in \text{Tbl}(\mathcal{D}_i)\) to the table \(\tilde{T}_i = \langle K_i, \tilde{t}_i \rangle \in \text{Tbl}(\tilde{D})\) with its tuple function \(K_i \xrightarrow{\tilde{t}_i} \text{tup}(\tilde{D})\) defined by composition, \(t_i \cdot \text{tup}(\tilde{h}_i, \tilde{f}_i, \tilde{g}_i) = \tilde{t}_i\). Here we have “horizontally abridged” and then “vertically extended” tuples in \(\text{tup}(\mathcal{D}_i) \subseteq \text{List}(Y_i)\) by composition along the tuple function \(\text{tup}(\mathcal{D}_i) \xrightarrow{\text{tup}(\tilde{h}_i, \tilde{f}_i, \tilde{g}_i)} \text{tup}(\tilde{D})\) (see LHS Fig. 11). This is linked to the table \(T_i\) by the table morphism \(T_i = \langle K_i, t_i \rangle \xrightarrow{\langle \tilde{h}_i, \tilde{f}_i, \tilde{g}_i \rangle \cdot K_i} \langle K_i, \tilde{t}_i \rangle = \tilde{T}_i\).

- Union (§ 3.2) of the tables \(\{\tilde{T}_i \mid i \in I\}\) in the fiber context \(\text{Tbl}(\tilde{D})\) defines the generic join \(\bigsqcap T = \tilde{T} = \sqrt[\big]{\{\tilde{T}_i \mid i \in I\}} = \bigvee_{i \in I}[\text{tbl}(\tilde{h}_i, \tilde{f}_i, \tilde{g}_i)(T_i) = \langle \tilde{K}, \tilde{t}\rangle, \text{whose key set is the disjoint union} \tilde{K} = +\{K_i \mid i \in I\}\) and whose tuple map is the mediating function \(\tilde{K} \xrightarrow{[\tilde{t}_i]} \text{tup}(\tilde{D})\) of the multi-opspan \(\{K_i \xrightarrow{\tilde{t}_i} \text{tup}(\tilde{D}) \mid i \in I\}\), resulting in the discrete multi-opspan (cocone) \(\{\tilde{T}_i \xrightarrow{\tilde{t}_i} \tilde{T} \mid i \in I\}\).

Projection-expansion composed with join defines the multi-opspan of table morphisms

\[
\{T_i \xrightarrow{\langle \tilde{h}_i, \tilde{f}_i, \tilde{g}_i \rangle \cdot K_i} \bigsqcap T = \tilde{T} \mid i \in I\},
\]

illustrated in Fig. 29, which is the output for generic join (Tbl. 13).

Generic join is projection/expansion \((i \in I\) times) followed by join. This is the two-step process

\[
\bigsqcap T = \bigvee \{\text{tbl}(\tilde{h}_i, \tilde{f}_i, \tilde{g}_i)(T_i) \mid i \in I\}.
\]

Aside: Theoretically this would represent the colimit of a diagram \(I \xrightarrow{T} \text{Tbl}\) consisting of a linked collection of tables. But practically, we are only given the constraint (a diagram) \(I^{\text{op}} \xrightarrow{D} \text{Dom}\) consisting of a linked collection of signed domains \(\{\mathcal{D}_i = D(i) \mid i \in I\}\) and the input \(I \xrightarrow{T} \text{Tbl}\) consisting of a sufficient indexed collection of tables (Def. 3) \(\{T_i = \text{Tbl}(\mathcal{D}_i) \mid i \in I \subseteq \text{obj}(I)\}\).
Table 13. FOLE Generic Join I/O

Fig. 29. FOLE Generic Join
6 Unorthodox Composite Operations

Operations are put into the unorthodox category for various reasons. Selection and select-join are special cases of a more general concept: selection is a special case of natural join, and select-join is a special case of natural multi-join. Filter join and data-type meet do not follow the dual concepts of either limit or colimit of category theory. Subtraction, division and outer-join are complex. Tbl. 14 lists the composite relational operations defined in this section.

| Operation          | Expression                                                                 | Type |
|--------------------|-----------------------------------------------------------------------------|------|
| selection          | $\sigma_R(T) = tbl_A(h)(R') \times T$                                     | $\in \mbox{Tbl}(A)$ |
| select-join        | $T_1 \bowtie A T_2 = \sigma_R(T_1 \bowtie A T_2)$                          | $\in \mbox{Tbl}(A)$ |
| filtered join      | $T_1 \ominus A T_2 = tbl_A(g_1)(T_1) \cup tbl_A(g_2)(T_2)$                | $\in \mbox{Tbl}(S)$ |
| data-type meet     | $T_1 \boxdot A T_2 = tbl_A(g_1)(T_1) \land tbl_A(g_2)(T_2)$               | $\in \mbox{Tbl}(S)$ |
| subtraction        | $T \sim T_2 = T - tbl_A(g)(T_2)$                                           | $\in \mbox{Tbl}(S)$ |
| division           | $T \div A T_2 = tbl_A(\xi_1)(T) - tbl_A(\xi_1)\{ tbl_A(\xi_1)(T) \times T_2 \} - T$ | $\in \mbox{Tbl}(A)$ |
| outer-join         | $T_1 \sqcap A T_2 = tbl_A(\xi_1)(T_1) \wedge tbl_A(\xi_2)(T_2)$ implies $T_1 \sqcup A T_2 = tbl_A(\emptyset)(T_1) \vee tbl_A(\emptyset)(T_2 - tbl_A(\xi_1)(T_2)) \times T_*$ | $\in \mbox{Tbl}$ |

Table 14. FOLE Unorthodox Composite Relational Operations
6.1 Selection.

Let $\mathcal{A}$ be a fixed type domain. General selection is a binary operation $\sigma_{R^\prime}(T)$, where $T$ is a table and $R^\prime$ is a relation that may represent a propositional formula on sorts in the header of $T$. The relation $R^\prime$ might consist of atoms $n_0\theta n_1$ or $n\theta n$ as in the examples in §A.2, plus the Boolean operators $\land$ (and), $\lor$ (or) and $\neg$ (negation). General selection selects all those tuples in $T$ for which $R^\prime$ holds. More precisely, the selection $\sigma_{R^\prime} p T q$ denotes all tuples in $T$ whose projection in $\text{tbl}_{A^p} h q$ is also in $R^\prime$; equivalently, a tuple is in the selection $\sigma_{R^\prime} p T q$ when it is in both $T$ and the inflation $\text{tbl}_{A^p} h q (R^\prime)$. Selection, within the context $\text{Tbl}_{A^p}$, is inflation followed by meet.

Constraint: Consider an $X$-sorted signature morphism $S^\prime \xrightarrow{h} S$ in $\text{List}(X)$ consisting of an index function $I^\prime \xrightarrow{h} I$. This forms a (trivial) $X$-sorted signature span $S^\prime \xleftarrow{I^\prime} S^\prime \xrightarrow{h} S$, which is a special case of the constraint for natural join (Tbl. 5). This is the constraint for selection (Tbl. 15).

Construction: The pushout of this constraint in $\text{List}(X)$ is the (trivial) opspan $S^\prime \xrightarrow{h} S \xleftarrow{I^\prime} S$ of injection $X$-signature morphisms with pushout signature $S^\prime + S^\prime = S$, which is a special case of the construction for natural join (Tbl. 5). This is the construction for selection (Tbl. 15).

Input: Consider a relation $R^\prime = \langle R^\prime, i^\prime \rangle \in \text{Rel}_{A}(S^\prime)$ and a table $T = \langle K, t \rangle \in \text{Tbl}_{A}(S)$. This is the input for selection (Tbl. 15), a special case of the input

---

46 Selection uses auxiliary relations. See §A.2 for some examples of auxiliary relations.
47 Selection is defined by using the fiber adjunction of tables (Disp. 1 of §3.3.1) of the $X$-sorted signature morphism $S^\prime \xrightarrow{h} S$. The left adjoint fiber passage $\text{Tbl}_{A}(S^\prime) \xrightarrow{\text{tbl}_{A}(h)} \text{Tbl}_{A}(S)$ defines projection by composition. The projection $T_h = \text{tbl}_{A}(h)(T) = \langle K, t \cdot \text{tup}_{A}(h) \rangle$ consists of the columns of table $T$ under sub-header $S^\prime$; elements of the set $\text{tup}_{A}(S^\prime)$. Some of these may be in $R$, and some in the complement $\text{tup}_{A}(S^\prime) - R$. The purpose of selection is to choose those tuples of $T$ whose $S^\prime$-component is in $R$. 
for natural join (Tbl. 5), since any relation is a table by reflection § 3.1.

**Output:** The output is in inflation followed by meet.

- Inflation $\text{Tbl}_A(S') \xrightarrow{\text{ tbl}_A(h)} \text{Tbl}_A(S)$ (§ 3.3.1) along the tuple function of the $X$-signature morphism $S' \xrightarrow{h} S$ maps the relation $R'$ to the $A$-table $\hat{R}' = \text{ tbl}_A(h)(R') = \langle \hat{R}', \hat{i}' \rangle \in \text{Tbl}_A(S)$ with its tuple function $\hat{R}' \xrightarrow{\hat{i}'} \text{ tup}_A(S)$ defined by pullback, $\hat{i} \cdot \hat{i}' = i' \cdot \text{ tup}_A(h)$. This is linked to the relation $R'$ by the $A$-table morphism $\hat{R}' \xrightarrow{\hat{i}'} \text{ tup}_A(S)$ along the tuple function of the $X$-signature morphism $S_1 \xrightarrow{h} S$. Inflation of table $\mathcal{T}$ along the identity signature morphism is identity.

- Intersection (§ 3.2) of the two inflation tables $\hat{R}'$ and $\mathcal{T}$ in the context $\text{Tbl}_A(S)$ defines the selection table $\sigma_{R'}(\mathcal{T}) = \hat{R}' \wedge \mathcal{T}$, whose key set $\hat{R}'$ is the pullback and whose tuple map is the mediating function $\hat{R}' \xrightarrow{(i', \hat{j})} \text{ tup}_A(S)$ of the opspan $\hat{R}' \xrightarrow{\sigma_{R'}(\mathcal{T})} \text{ tup}_A(S)$ resulting in the span $\hat{R}' \xrightarrow{\sigma_{R'}(\mathcal{T})} \mathcal{T}$.

Inflation composed with meet defines the span of $A$-table morphisms

$$\hat{R}' \xrightarrow{(h, \hat{k})} \sigma_{R'}(\mathcal{T}) \xrightarrow{\hat{g}_2} \mathcal{T},$$

which is the output for selection. This is a special case $\sigma_{R'}(\mathcal{T}) = R' \Join_A \mathcal{T}$ of the output for natural join (Tbl. 5).

The selection flowchart input/output is displayed in Tbl. 15. Selection within the context $\text{Tbl}(A)$ is inflation followed by meet. This is a two-step process

$$\sigma_{R'}(\mathcal{T}) = \text{ tbl}_A(h)(R') \wedge \mathcal{T}.$$
6.2 Select-join.

If we want to combine tuples from two tables, where the combination condition is not simply the equality of shared attributes, then it is convenient to have a more general form of the natural join operator, which is known as the select-join. The select-join is a ternary operator that is written as $T_1 \natural_\mathcal{R} T_2$. The result of the select-join operation consists of all combinations of tuples in $T_1$ and $T_2$ that satisfy $\mathcal{R}$. The definition of this operation in terms of more fundamental operations is $T_1 \natural_\mathcal{R} T_2 = \sigma_{\mathcal{R}}(T_1 \natural_\mathcal{A} T_2)$. The input/output for the natural join component is independent, with the input/output for the selection component depending on it.

**Constraint/Construction:** The constraint and construction is an interleaved process for the natural join and the selection aspects.

1. Consider an $X$-sorted signature span $S_1 \xrightarrow{h_1} S \xrightarrow{h_2} S_2$ in $\text{List}(X)$ consisting of a span of index functions $I_1 \xrightarrow{h_1} I \xrightarrow{h_2} I_2$. This is the constraint for the natural join aspect (Tbl. 5). This has pushout signature $S_1 +_S S_2$ and injection $X$-signature morphisms $S_1 \xrightarrow{\iota_1} S_1 +_S S_2 \xrightarrow{\iota_2} S_2$ with index function opspan $I_1 \xrightarrow{\iota_1} \langle I_1 +_I I_2, \{s_1, s_2\} \rangle \xrightarrow{\iota_2} I_2$.

2. Consider a connecting $X$-sorted signature morphism $S_3 \xrightarrow{\iota_3} S_1 +_S S_2$ in $\text{List}(X)$, which is the RHS of a trivial $X$-sorted signature span. This is the constraint for the selection aspect. The construction for the selection aspect is the pushout of this constraint in $\text{List}(X)$, which is the (trivial) opspan whose LHS is the $X$-sorted signature morphism $S_3 \xrightarrow{\iota_3} S_1 +_S S_2$.

---

48 The $\theta$-join is a special case, where the headers of the table $T_1$ and $T_2$ are disjoint, so that we use the Cartesian product. When the relation $\mathcal{R}$ represents the equality operator ($=$) on two attributes, this join is called an equi-join.
Input/Output: The input and output is an interleaved process for the natural join and the selection aspects.

1. Consider a pair of tables $T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_A(S_1)$ and $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_A(S_2)$. This is the input for natural join (Tbl. 5). The natural join from § 4.3 is the top of the $A$-table span

$$T_1 = \langle S_1, K_1, t_1 \rangle \xrightarrow{(s_1, k_1)} T_1 \bowtie_A T_2 \xrightarrow{(s_2, k_2)} \langle S_2, K_2, t_2 \rangle = T_2$$

with underlying $X$-signature opspan

$$S_1 \xrightarrow{s_1} S_1 + S_2 \xrightarrow{k_2} S_2.$$

This is the output for natural join (Tbl. 5).

2. Consider a relation $R = \langle R, i \rangle \in \text{Rel}_A(S_3)$ and the table $T_1 \bowtie_A T_2 = \langle \hat{R}_{12}, (\hat{t}_1, \hat{t}_2) \rangle \in \text{Tbl}_A(S_1 + S_2)$. This is the input for selection (Tbl. 15).

Selection is inflation followed by meet. The $A$-relation $R = \langle R, i \rangle \in \text{Rel}_A(S_3)$ is mapped by inflation to the $A$-relation $\hat{R} = \text{tbl}_A(S_3)(R) = \langle \hat{R}, \hat{i} \rangle \in \text{Rel}_A(S_1 + S_2)$ with its tuple function $\hat{R} \xrightarrow{\hat{t}_1} \text{tup}_A(S_1 + S_2)$.

This is linked to relation $R$ by the $A$-relation morphism $R = \langle S_1, R, i \rangle \xrightarrow{(s_1, \hat{i}, r, \hat{k})} \langle S_1 + S_2, R, \hat{i} \rangle = \hat{R}$. Intersection, within the context $\text{Tbl}_A(S_1 + S_2)$, of $A$-relation $\hat{R}$ with $A$-table $T_1 \bowtie_A T_2$ results in the table

$$\sigma_R(T_1 \bowtie_A T_2) = i^*_R(R) \land (T_1 \bowtie_A T_2) = i^*_A(R) \land i^*_R(T_1) \land i^*_A(T_2).$$

This defines the span of $A$-table morphisms

$$R \xrightarrow{(s_1, \hat{i}, r, \hat{k})} \sigma_R(T_1 \bowtie_A T_2) = T_1 \bowtie_A T_2 \xrightarrow{\hat{R}} T_1 \bowtie_A T_2,$$

which is the output for select-join.

The select join flowchart input/output is displayed in Tbl. 16. Select-join is natural join, followed by selection. This is the two-step process

$$T_1 \bowtie_A T_2 \xrightarrow{s_R} \sigma_R(T_1 \bowtie_A T_2).$$

| $\text{Tbl}_A(S_1)$ | $\text{Tbl}_A(S_2)$ | $\text{Tbl}_A(S_1 + S_2)$ | $\text{Tbl}_A(S_2)$ | $\text{Tbl}_A(S_1 + S_2)$ |
|---------------------|---------------------|------------------------|---------------------|------------------------|
| $\hat{R} \in \text{Tbl}_A(S_1)$ | $\hat{R} \in \text{Tbl}_A(S_2)$ | $\text{Tbl}_A(S_1 + S_2)$ | $\text{Tbl}_A(S_2)$ | $\text{Tbl}_A(S_1 + S_2)$ |
| $\text{input select}$ | $\text{input nat-join}$ | $\text{output select}$ | $\text{output nat-join}$ |

Table 16. FOLE Select-Join I/O
Note 1. From another standpoint, select-join is a form of multi-join, where we start with the star-shaped diagram of X-signature morphisms below-left, and end with the star-shaped diagram of table morphisms below-right.

\[
\begin{array}{cc}
S_1 & S_2 \\
\downarrow^{i_1} & \downarrow^{i_2} \\
S_1 \cup S_2 & S_2 \\
\end{array}
\begin{array}{cc}
T_1 & T_2 \\
\downarrow^{\langle i_1, k_1 \rangle} & \downarrow^{\langle i_2, k_2 \rangle} \\
T_1 \cup^R T_2 & T_2 \\
\end{array}
\]

Here, select-join can also be defined by the two-step process

\[
T_1 \cup^R T_2 = tbl_A(t_3)(T_1) \land tbl_A(t_3)(T_2) \land tbl_A(t_3)(R)
\]

of inflation (thrice) followed by multi-meet.

**Proposition 8.** Select-join \( \hat{\boxdot} \) preserves union \( \lor \) and intersection \( \land \).

**Proof.** Prop. 4 and Prop. 7.

\[\square\]
6.3 Filtered Join.

Let \( S \) be a signature. We are given two \( S \)-tables \( T_1 = \langle A_1, K_1, t_1 \rangle \) and \( T_2 = \langle A_2, K_2, t_2 \rangle \) connected by an \( X \)-sorted type domain opspan \( A_1 \xrightarrow{g_1} A \xrightarrow{g_2} A_2 \) via a third type domain \( A \), and consisting of a span of data value functions \( Y_1 \xrightarrow{g_1} Y \xrightarrow{g_2} Y_2 \). The set \( Y \) represents authentic data values. Filtered join is a binary operation \( T_1 / S T_2 \) that filters out the tuples with non-authentic data values, and then joins the results. When the data value functions are injections \( Y_1 \xleftarrow{g_1} Y \xleftarrow{g_2} Y_2 \), \( T_1 \) and \( T_2 \) will be tables whose tuples

\[
\text{tup}_S(A_1) \subseteq \varphi\text{List}(Y_1) \supseteq \varphi\text{List}(Y) \subseteq \varphi\text{List}(Y_2) \supseteq \text{tup}_S(A_2)
\]

have un-authenticated data values in \( Y_1 \backslash Y \) and \( Y_2 \backslash Y \). Filtered join \( T_1 \sqsubseteq S T_2 \) restricts to only those tuples in \( \varphi\text{List}(Y) \) with the authentic data values in \( Y \).

A tuple is in the filtered join \( T_1 \sqsubseteq S T_2 \) when it is either in the restriction \( \text{tbl}_S(g_1)(T_1) \) or in the restriction \( \text{tbl}_S(g_2)(T_2) \). Filtered join, within the context \( \text{Tbl}(S) \), is restriction followed by join. We use the following routes of flow.

Similar to natural join and data-type join, we can define filtered join for any number of tables \( \{ T_1, T_2, T_3, \ldots, T_n \} \) with a comparable constraint.

**Constraint:** Consider an \( X \)-sorted type domain opspan \( A_1 \xrightarrow{g_1} A \xrightarrow{g_2} A_2 \) in \( \text{Cls}(X) \) consisting of a span of data value functions \( Y_1 \xrightarrow{g_1} Y \xrightarrow{g_2} Y_2 \). This is the constraint for filtered join (Tbl. 17).

**Construction:** The construction for filtered join is the same as the constraint for filtered join (Tbl. 17).

**Input:** Consider a pair of tables \( T_1 = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1) \) and \( T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2) \). This is the input for filtered join (Tbl. 17).

\[\text{49} \text{ See the discussion of authenticity w.r.t. restriction in } \S 3.3.2.\]
Output: The output is restriction (twice) followed by join.

- Restriction (§ 3.3.2) TblS(A1) \xrightarrow{\text{tbl}_S(g_1)} TblS(A) along the tuple function of the X-type domain morphism A1 \xrightarrow{g_1} A, maps the table T1 to the table \( \widehat{T_1} = \text{tbl}_S(g_1)(T_1) = \langle \widehat{K_1}, \widehat{t_1} \rangle \in TblS(A) \), with its tuple function \( \widehat{K_1} \xrightarrow{\widehat{t_1}} \text{tup}_S(A) \) defined by pullback, \( k_1 \cdot t_1 = \widehat{t_1} \cdot \text{tup}_S(g_1) \). This is linked to table T1 by the S-table morphism \( \tau_1 = \langle A_1, K_1, t_1 \rangle \xrightarrow{(g_1,k_1)} \langle A, K, t \rangle = \widehat{T_1} \). The same process can be defined for S-table \( T_2 \).

- Union (§ 3.2) of the two restriction tables \( \widehat{T_1} \) and \( \widehat{T_2} \) in the context TblS(A) defines the filtered join \( T_1 \circledast_T T_2 = \widehat{T_1} \vee \widehat{T_2} = \langle \widehat{K_1} + \widehat{K_2}, [\widehat{t_1}, \widehat{t_2}] \rangle \), whose key is the disjoint union \( \widehat{K_1} + \widehat{K_2} \) and whose tuple map \( \langle \widehat{K_1} + \widehat{K_2}, [\widehat{t_1}, \widehat{t_2}] \rangle \xrightarrow{\text{tup}_A(S)} \text{comediator of the opspan} \langle \widehat{t_1}, \widehat{t_2} \rangle \xrightarrow{\text{tup}_A(A_1 \times A_2)} \langle \widehat{t_1}, \widehat{t_2} \rangle, \) resulting in the span \( \langle \widehat{t_1}, \widehat{t_2} \rangle \xrightarrow{\circledast_T} T_1 \circledast_T T_2 \xrightarrow{\circledast_T} T_2 \).

Restriction composed with join defines the \((M\text{-shaped})\) multi-opspan of \( S \)-table morphisms

\[
\tau_1 \xrightarrow{(g_1,k_1)} \widehat{T_1} \xrightarrow{\hat{t_1}} T_1 \circledast_T T_2 \xrightarrow{\hat{t_2}} \tau_2 \xrightarrow{(g_2,k_2)} T_2,
\]

which is the output for filtered join (Tbl. 17).

Filtered join is restriction followed by join. This is a two-step process

\[
T_1 \circledast_T T_2 = \text{tbl}_S(g_1)(T_1) \vee \text{tbl}_S(g_2)(T_2).
\]

\[\text{Table 17. FOLE Filtered Join I/O}\]

\[
\begin{array}{|c|c|c|}
\hline
\text{A}_1 & \xrightarrow{g_1} & \text{A}_2 \\
\hline
\text{A}_1 & \xrightarrow{g_2} & \text{A}_2 \\
\hline
\tau_1 \in \text{Tbl}_S(A_1) \text{ and } \tau_2 \in \text{Tbl}_S(A_2) & \xrightarrow{\circledast_T} & \tau_1 \circledast_T \tau_2 \xrightarrow{\circledast_T} \tau_2 \\
\hline
\end{array}
\]

\footnote{The filtered join, \( \tau_1 \xrightarrow{(g_1,k_1)} \widehat{T_1} \xrightarrow{\hat{t_1}} T_1 \circledast_T T_2 \xrightarrow{\hat{t_2}} \tau_2 \xrightarrow{(g_2,k_2)} T_2 \), is shielded from and has no direct connection to either table \( T_1 \) or table \( T_2 \). This is comparable with the data-type meet in \( \S 6.4, \tau_1 \xrightarrow{(1,g_1)} \widehat{T_1} \xrightarrow{\hat{t_1}} T_1 \circledast_S T_2 \xrightarrow{\hat{t_2}} \tau_2 \xrightarrow{(1,g_2)} T_2 \), which also is shielded from and has no direct connection to either table \( T_1 \) or table \( T_2 \).}
6.4 Data-type Meet.

The data-type meet for tables is the relational counterpart (like the natural join) of the logical conjunction for predicates. Where the meet operation is the analogue for logical conjunction at the small scope \( \text{Tbl}(D) \) of a signed domain table fiber, the data-type meet is defined at the intermediate scope of a signature table fiber (in contrast to the natural join, which is define at the intermediate scope of a type domain table fiber). We focus on tables in the context \( \text{Tbl}(S) \) for fixed signature (header) \( S = \langle I, x, X \rangle \). We use the following routes of flow.

**Constraint:** The constraint for data-type meet is the same as the constraint for data-type join: an \( X \)-sorted type domain opspan \( A_1 \xrightarrow{\varphi_1} A \xrightarrow{\varphi_2} A_2 \) consisting of a span of data value functions \( Y_1 \xrightarrow{\varphi_1} Y \xrightarrow{\varphi_2} Y_2 \) (Tbl. 18).

**Construction:** The construction for data-type meet is the same as the construction for data-type join: the span \( A_1 \xrightarrow{\varphi_1} A_1 \times_A A_2 \xrightarrow{\varphi_2} A_2 \) (Tbl. 18).

**Input:** The input for data-type meet is the same as the input for data-type join (Tbl. 18): a pair of tables \( \mathcal{T}_1 \in \text{Tbl}_S(A_1) \) and \( \mathcal{T}_2 \in \text{Tbl}_S(A_2) \).
Output: The output is expansion (twice) followed by meet.

- Expansion $\text{Tbl}_S(A_1) \overset{\text{tbl}_S(g_1)}{\longrightarrow} \text{Tbl}_S(A_1 \times_A A_2)$ along the tuple function of the $X$-type domain morphism $A_1 \overset{g_1}{\rightarrow} A_1 \times_A A_2$ maps the $S$-table $T_1$ to the $S$-table $\tilde{T}_1 = \text{tbl}_S(g_1)(T_1) = \langle K_1, t_1 \rangle \in \text{Tbl}_S(A_1 \times_A A_2)$, with its tuple function $K_1 \overset{t_1}{\rightarrow} \text{tup}_S(A_1 \times_A A_2)$ defined by composition, $t_1 = t_1 \cdot \text{tup}_S(g_1)$. This is linked to the table $\tilde{T}_1$ by the $S$-table morphism $T_1 = \langle A_1, K_1, t_1 \rangle \overset{(1,g_1)}{\rightarrow} \langle A_1 \times_A A_2, K_1, t_1 \rangle = \tilde{T}_1$. Similarly for $S$-table $T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2)$.

- Intersection (§3.2) of the two expansion tables $\tilde{T}_1$ and $\tilde{T}_2$ in the context $\text{Tbl}_S(A_1 \times_A A_2)$ defines the data-type meet $T_1 \sqcap S T_2 = \tilde{T}_1 \cap \tilde{T}_2 = \langle K_1 \times K_2, (t_1, t_2) \rangle$, whose key set is the product $K_1 \times K_2$ and whose tuple map $K_1 \times K_2 \overset{(t_1, t_2)}{\rightarrow} \text{tup}_S(A)$ maps a pair of keys $(k_1, k_2) \in K_1 \times K_2$ to the common tuple $t_1(k_1) = t_2(k_2) \in \text{tup}_S(A_1 \times_A A_2)$. Intersection is the product in $\text{Tbl}_S(A)$ with span $\tilde{T}_1 \overset{1}{\leftarrow} \tilde{T}_1 \cap \tilde{T}_2 \overset{2}{\rightarrow} \tilde{T}_2$.

Expansion composed with meet defines the (W-shaped) multi-span of $S$-table morphisms

$$
\begin{align*}
\quad & T_1 \overset{(1,g_1)}{\rightarrow} \tilde{T}_1 \overset{1}{\leftarrow} T_1 \sqcap S T_2 \overset{2}{\rightarrow} \tilde{T}_2 \overset{(1,g_2)}{\rightarrow} T_2,
\end{align*}
$$

which is the output for data-type meet (Tbl.11).

Data-type meet is expansion followed by meet. This is the two-step process

$$
T_1 \sqcap S T_2 = \text{tbl}_S(g_1)(T_1) \cap \text{tbl}_S(g_2)(T_2).
$$

| $A_1 \overset{g_1}{\rightarrow} A_1 \times_A A_2$ | constraint |
| $A_1 \overset{g_2}{\rightarrow} A_1 \times_A A_2$ | construction |
| $T_1 \in \text{Tbl}_S(A_1)$ and $T_2 \in \text{Tbl}_S(A_2)$ | input |
| $T_1 \overset{(1,g_1)}{\rightarrow} \tilde{T}_1 \overset{1}{\leftarrow} T_1 \sqcap S T_2 \overset{2}{\rightarrow} \tilde{T}_2 \overset{(1,g_2)}{\rightarrow} T_2$ | output |

Table 18: FOLE Data-type Meet I/O

---

51 The data-type meet of a disjoint sum is the empty table.
52 The data-type meet, $T_1 \overset{(1,g_1)}{\rightarrow} \tilde{T}_1 \overset{1}{\leftarrow} T_1 \sqcap S T_2 \overset{2}{\rightarrow} \tilde{T}_2 \overset{(1,g_2)}{\rightarrow} T_2$, is shielded from and has no direct connection to either table $T_1$ or table $T_2$. This is comparable with the filtered join in §6.3, $T_1 \overset{(g_1,k_1)}{\rightarrow} \tilde{T}_1 \overset{1}{\leftarrow} T_1 \sqcap S T_2 \overset{2}{\rightarrow} \tilde{T}_2 \overset{(g_2,k_2)}{\rightarrow} T_2$, which also is shielded from and has no direct connection to either table $T_1$ or table $T_2$. 
6.5 Subtraction.

Here we define the operation of subtraction in terms of the basic operations. Let \( S = \langle I, x, X \rangle \) be a fixed signature. Let \( T \) and \( T_2 \) be two \( S \)-tables. For subtraction, we assume that the data values in the body of \( T_2 \) are a subset of those of \( T \). Hence, assume that \( T \) is a table in \( \text{Tbl}_S(A) \), \( T_2 \) is a table in \( \text{Tbl}_S(A_2) \), and that these are connected with an \( X \)-type domain morphism \( A \overset{g}{\to} A_2 \) with data value function \( Y \overset{\omega}{\leftarrow} Y_2 \). The subtraction \( T \sim T_2 \) consists of the expansion of \( T_2 \) along the \( X \)-type domain morphism \( A \overset{g}{\to} A_2 \) to the \( X \)-type domain \( A \), followed by subtraction from \( T \) in the signed domain \( \langle S, A \rangle \). Subtraction is defined by the two-step process illustrated in Fig. 34.

**Constraint/Construction:** The constraint and construction for subtraction are the same (Tbl. 19): an \( X \)-type domain morphism \( A \overset{g}{\to} A_2 \) with data value function \( Y \overset{\omega}{\leftarrow} Y_2 \).

**Input:** Consider a pair of tables \( T = \langle K, t \rangle \in \text{Tbl}_S(A) \) and \( T_2 = \langle K_2, t_2 \rangle \in \text{Tbl}_S(A_2) \). This is the input for subtraction (Tbl. 19).

**Output:** Subtraction is expansion (once) followed by difference.

- Expansion \( \text{Tbl}_S(A) \overset{\text{tbl}_S(g)}{\longrightarrow} \text{Tbl}_S(A_2) \) along the tuple function of the \( X \)-type domain morphism \( A \overset{g}{\to} A_2 \) maps the \( S \)-table \( T_2 \in \text{Tbl}_S(A_2) \) to the \( S \)-table \( \widetilde{T}_2 = \text{tbl}_S(g)(T_2) = \langle K_2, \tilde{t}_2 \rangle \in \text{Tbl}_S(A) \) with its tuple function \( K_2 \overset{\tilde{t}_2}{\to} \text{tup}_S(A) \) defined by composition, \( \tilde{t}_2 = t_2 \cdot \text{tup}_S(g) \). This is linked to the table \( T \) by the \( S \)-table morphism \( T_2 = \langle A_2, K_2, \tilde{t}_2 \rangle \overset{(1, 0)}{\longrightarrow} \langle A, K, t \rangle = \widetilde{T}_2 \).

- The difference \( T \sim T_2 = T - \widetilde{T}_2 = T - \text{tbl}_S(g)(T_2) = \langle K_2, \tilde{t}_2 \rangle \) is the desired table in \( \text{Tbl}_S(A) \) with key set \( K_2 \) and tuple map \( \tilde{t}_2 : K_2 \overset{k}{\to} K_2 \overset{i_2}{\to} \text{tup}_S(A) \).

Expansion composed with difference defines the inclusion \( S \)-table morphism \( T \overset{\omega}{\longrightarrow} T \sim T_2 \), which is the output for subtraction (Tbl. 19).
Subtraction is expansion followed by difference. This is the two-step process

\[ T \sim T_2 \doteq T - tbl_S(g)(T_2). \]

| \( A \rightarrow A_2 \) | constraint |
| \( A \rightarrow A_2 \) | construction |
| \( T \in Tbl_S(A) \) and \( T_2 \in Tbl_S(A_2) \) | input |
| \( T \leftarrow T \sim T_2 \) | output |

Table 19. FOLE Subtraction I/O

**Proposition 9.** There is a table morphism.

\[ T \rightarrow (T \sim T_2) \oplus T_2. \]

**Proof.** \( T \sim T_2 \) has all the tuples in \( T \), except for those in \( T_2 = tbl_S(g)(T_2) \). The latter can be “added” back with the join \((T \sim T_2) \oplus T_2 \cong (T - T_2) \lor T_2\), thus defining a table morphism \( T \rightarrow (T \sim T_2) \oplus T_2 \). □

---

53 Analogous to \( B \subseteq (B \setminus A) \cup A = B \cup A \).

54 Argument in terms of the underlying relations in the reflection Prop. 1.
6.6 Division.

Here we define the operation of division in terms of the basic operations. Let $\mathcal{A} = \langle X, Y, \models_{\mathcal{A}} \rangle$ be a fixed type domain. Let $\mathcal{T}$ and $\mathcal{T}_2$ be two $\mathcal{A}$-tables. For division, we assume that the attribute names in the header of $\mathcal{T}_2$ are a subset of those of $\mathcal{T}$. The division $\mathcal{T} \div_{\mathcal{A}} \mathcal{T}_2$ consists of the contraction of tuples in $\mathcal{T}$ to the attribute names unique to $\mathcal{T}$ (the projection to $\mathcal{S}_1$), for which it holds that all their combinations with tuples in $\mathcal{T}_2$ (the Cartesian product) are present in $\mathcal{T}$ (take the difference in $\mathcal{S}_1 \setminus \mathcal{S}_2$, then project to $\mathcal{S}_1$, finally take the difference in $\mathcal{S}_1$). Division is defined by the multi-step process in Fig. 35.

**Constraint:** The constraint for division (Tbl. 20) is the same as the constraint for Cartesian product: two $X$-signatures $\mathcal{S}_1$ and $\mathcal{S}_2$. These are linked by the span of $X$-signatures $\mathcal{S}_1 \xrightarrow{0_{I_1}} \mathcal{S}_2 \xrightarrow{0_{I_2}} \mathcal{S}_2$ in $\text{List}(X)$ consisting of a span of injection index functions $I_1 \xrightarrow{0_{I_1}} \mathcal{T}_1 \subset \mathcal{T}_2 \xrightarrow{0_{I_2}} I_2$.

**Construction:** The construction for division (Tbl. 20) is the same as the construction for Cartesian product: the opspan $\mathcal{S}_1 \xleftarrow{\iota_1} \mathcal{S}_1 + \mathcal{S}_2 \xleftarrow{\iota_2} \mathcal{S}_2$ of injection $X$-signature morphisms with coproduct $X$-signature $\mathcal{S}_1 + \mathcal{S}_2$ and injection index function opspan $I_1 \xleftarrow{\iota_1} \langle I_1 + I_2, [s_1, s_2] \rangle \xleftarrow{\iota_2} I_2$.

**Input:** Consider a pair of tables $\mathcal{T} \in \text{Tbl}_{\mathcal{A}}(\mathcal{S}_1 + \mathcal{S}_2)$ and $\mathcal{T}_2 \in \text{Tbl}_{\mathcal{A}}(\mathcal{S}_2)$. This is the input for division (Tbl. 20).
Output: Division is projection, Cartesian product, difference, projection and difference.

- Project table $\mathcal{T} \in \text{Tbl}_{A}(S_{1}+S_{2})$ back along the injection $X$-signature morphism $S_{1} \hookrightarrow S_{1}+S_{2}$ to its unique attribute names $S_{1}$, getting the table $\text{tbl}_{A}(t_{1})(\mathcal{T}) \in \text{Tbl}_{A}(S_{1})$.

- Form the Cartesian product with the table $T_{2} \in \text{Tbl}_{A}(S_{2})$, getting the table $\mathcal{T} = \text{tbl}_{A}(t_{1})(\mathcal{T}) \times T_{2} \in \text{Tbl}_{A}(S_{1}+S_{2})$.

- The difference $\mathcal{T} = \mathcal{T} - \mathcal{T} \in \text{Tbl}_{A}(S_{1}+S_{2})$ gives the unwanted tuples.

- The subtraction $\mathcal{T} \div A T_{2} = \text{tbl}_{A}(t_{1})(\mathcal{T}) - \text{tbl}_{A}(t_{1})(\mathcal{T}) \in \text{Tbl}_{A}(S_{1})$ is the desired table.

Hence, the division of $\mathcal{T}$ by $T_{2}$ is defined by the multi-step process:

$$\mathcal{T} \div A T_{2} \equiv \text{tbl}_{A}(t_{1})(\mathcal{T}) - \text{tbl}_{A}(t_{1})\left( (\text{tbl}_{A}(t_{1})(\mathcal{T}) \times T_{2}) - \mathcal{T} \right).$$

| $S_{1}$ and $S_{2}$ | constraint |
|---------------------|------------|
| $S_{1} \hookrightarrow S_{1}+S_{2} \hookrightarrow S_{2}$ | construction |
| $\mathcal{T} \in \text{Tbl}_{A}(S_{1}+S_{2})$ and $T_{2} \in \text{Tbl}_{A}(S_{2})$ | input |
| $\mathcal{T} \div A T_{2}$ | output |

Table 20. FOLE Division I/O

**Proposition 10.** There is a table morphism $^{55}$

$$\mathcal{T} \leftrightarrow (\mathcal{T} \div A T_{2}) \times T_{2}.$$  

**Proof.** $^{56}$ Suppose $t \in (\mathcal{T} \div A T_{2}) \times T_{2}$, so that $t = (t_{1}, t_{2})$ for $t_{1} \in (\mathcal{T} \div A T_{2})$ and $t_{2} \in T_{2}$. By definition, $t_{1} \in \text{tbl}_{A}(t_{1})(\mathcal{T})$ and $t_{1} \notin \text{tbl}_{A}(t_{1})\left( (\text{tbl}_{A}(t_{1})(\mathcal{T}) \times T_{2}) - \mathcal{T} \right)$. This implies that $t = (t_{1}, t_{2}) \in (\text{tbl}_{A}(t_{1})(\mathcal{T}) \times T_{2})$. Suppose $t \notin \mathcal{T}$. Then $t \in (\text{tbl}_{A}(t_{1})(\mathcal{T}) \times T_{2}) - \mathcal{T}$. Hence, $t_{1} = \text{tbl}_{A}(t_{1})(t) \in \text{tbl}_{A}(t_{1})\left( (\text{tbl}_{A}(t_{1})(\mathcal{T}) \times T_{2}) - \mathcal{T} \right)$, a contradiction. Hence, $t \in \mathcal{T}$.  

$^{55}$ Analogous to $\mathcal{T} \cong (7 \times 3) / 2$ or modus ponens $q \leftrightarrow ((q \leftrightarrow p) \land p)$.

$^{56}$ Argument in terms of the underlying relations in the reflection Prop. 1.
6.7 Outer-join.

In this section we assume the index functions are injective when \( y \) to the type domain \( A \) be the restriction of \( A \) under the coproduct signature in the opspan \( X \). Let \( A_1 \) \( A_2 \) \( I \) \( Y \), \( \bar{1} \), \( \bar{2} \), \( 0 \) to expand the type domain \( A \) for all \( x \in X \) and \( y \in Y \); and (ii) \( \models_{A_2} x \) for all \( x \in X \). Hence, in \( A_1 \), the (data type) extent of any sort \( x \in X \) is the subset \( \text{ext}_{A_1}(x) = A_1 + \{\} \). The inclusion data value function \( Y + \{\} \xrightarrow{0} Y \) defines an \( X \)-sorted type domain morphism \( A_1 \xrightarrow{0} A \). This defines a fiber adjunction of tables \( \text{Tbl}_2(A_1) \xrightarrow{\text{th}_{A_1}(I_1) - \text{th}_{A_1}(I_2)} \text{Tbl}_2(A_2) \), where the left adjoint defines expansion. Consider a third table \( A_3 \) \( A_2 \) \( I_3 \) \( I_2 \) with underlying signature \( S_2 = \langle I_2, \bar{2} \rangle \) that consists of a single tuple of null values.

---

57 In this section we assume the index functions are injective \( I_1 \xrightarrow{h_1} I \xrightarrow{h_2} I_2 \).
**Constraint:** The constraint for outer-join has two parts: (1) an $X$-sorted signature span $S_1 \xRightarrow{f_1} S \xRightarrow{f_2} S_2$ in List($X$), the constraint for natural join (Tbl. 5); and (2) an $X$-sorted type domain morphism $A_\ast \xleftarrow{0} A$ in Cls($X$).

**Construction:** The flowchart for left outer-join is illustrated in Fig. 36. Since outer-join involves two type domains, only the set of sorts $X$ is in common. Taking together the discussions about signatures and type domains, this defines an opspan of signed domain morphisms

$$\langle S_1, A_\ast \rangle \xRightarrow{\langle \tau_1, 1_X, 0 \rangle} \langle S_1 + S_2, A_\ast \rangle \xRightarrow{\langle \tau_2, 1_X, 0 \rangle} \langle S_2, A_\ast \rangle.$$  

which factors as visualized in the following two squares.

$$\begin{array}{c|c}
\langle S_1, A_\ast \rangle & \langle S_1 + S_2, A_\ast \rangle \\
\hline
\langle 1_{S_1}, 0 \rangle \quad \text{=} \quad \langle \tau_1, 1_X, 0 \rangle \\
\langle 1_{S_2}, 0 \rangle \quad \text{=} \quad \langle \tau_2, 1_X, 0 \rangle
\end{array}$$

Comparing the above figure with Fig. 12 in § 3.3.4, we see that flow in the square is appropriate.

**Input:** The input for outer-join is a pair of tables $T_1 \in \text{Tbl}_A(S_1)$ and $T_2 \in \text{Tbl}_A(S_2)$ as in the input for natural join (Tbl. 5).

**Output:** Left outer-join is defined by the following multi-step process.

1. The **natural join** (§ 4.3) $T_{12} = T_1 \boxtimes_A T_2 = \text{tbl}_A(\langle \tau_1 \rangle)(T_1) \wedge \text{tbl}_A(\langle \tau_2 \rangle)(T_2)$ is the meet of inflations along $X$-signature opspan $S_1 \xleftarrow{\circ} S_1 + S_2 \xleftarrow{\circ} S_2$.

2. The **left semi-join** (§ 4.3.1) $T_1 \boxslash_A T_2 = \text{tbl}_A(\langle \tau_1 \rangle)(T_1) \boxslash_A T_2 \in \text{Tbl}_A(S_1)$ is the projection (§ 3.3.1) of the natural join along the $X$-signature morphism $S_1 \xleftarrow{\circ} S_1 + S_2$.

3. The **left anti-join** (§ 4.3.2) $\bar{T}_1 = T_1 \boxslash_A T_2 = T_1 - (T_1 \boxslash_A T_2) \in \text{Tbl}_A(S_1)$ is the difference between the table $T_1$ and the semi-join $T_1 \boxslash_A T_2$.

4. The **expansion** (§ 3.3.2) $\hat{T}_1 = \text{tbl}_S(\hat{0})(T_1 \boxslash_A T_2) \in \text{Tbl}_A(S_1, A_\ast)$ is the left adjoint flow along the $X$-sorted type domain morphism $A_\ast \xleftarrow{0} A$.

5. The **Cartesian product** (§ 4.3) $\hat{T}_1 \times \mathfrak{T}_* = \text{tbl}_A(\langle \tau_1 \rangle)(\hat{T}_1) \wedge \text{tbl}_A(\langle \tau_2 \rangle)(\mathfrak{T}_*)$ is the meet of inflations along the $X$-signature opspan $S_1 \xleftarrow{\circ} S_1 + S_2 \xleftarrow{\circ} S_2$.

Here $\hat{T}_1 \times \mathfrak{T}_* \in \text{Tbl}_A(S_1 + S_2) \equiv \text{Tbl}_A(S_1, S_2)$.

6. The **expansion** (§ 3.3.2) $\hat{T}_{12} = \text{tbl}_S(\hat{0})(T_1 \boxslash_A T_2) \in \text{Tbl}_A(S_1 + S_2, A_\ast)$ is the left adjoint flow along the $X$-sorted type domain morphism $A_\ast \xleftarrow{0} A$.

7. The left outer-join $T_1 \boxslash_A T_2 = T_{12} \cup (\hat{T}_1 \times \mathfrak{T}_*)$ is union of the expansion $\hat{T}_{12}$ and the Cartesian product $\hat{T}_1 \times \mathfrak{T}_*$. The right outer-join has a similar definition.

\[58\] The table $\mathfrak{T}_* \in \text{Tbl}_A(S_2)$ is an implicit table that helps build the output.
7 Conclusion and Future Work

Conclusion. This paper describes a well-founded semantics for relational algebra in the first-order logical environment FoLE, thus providing a theoretical foundation for relational databases. Here we have defined a typed semantics for the flowcharts used in analyzing, designing, documenting and managing database query processing. In the FoLE approach to relational algebra, each relational operator is a composite concept. The structure of a relational operator is represented by a flowchart made up of basic components. The basic components are divided into three categories: reflection, Boolean operators, and adjoint flow. Each flowchart is typed: the inputs are type and linked, each step in the flowchart is typed, and the output is typed. These types are of three kinds: either a type domain \( \mathcal{A} \), a signature \( \mathcal{S} \), or a signed domain \( \mathcal{D} \). Implicit in the background of each non-generic flowchart is a particular tabular component that is fixed: either a type domain, a signature, or a signed domain. The paper demonstrates that the FoLE approach for representing the relational model is very natural, providing a clear approach to its implementation.

Future Work. Two forms for the first-order logical environment FoLE have been developed: the **classification form** and the **interpretation form**. The classification form of FoLE is developed in the papers “The **ERA** of FoLE: Foundation” [8] and “The **ERA** of FoLE: Superstructure” [9]. The interpretation form of FoLE is developed in the papers “The **FoLE** Table” [10] and the unpublished paper “The **FoLE** Database” [11]. Both of the latter two papers expand on material found in the paper “Database Semantics” [6]. In the two papers [8] and [9] that develop the classification form of FoLE, the classification concept of information flow [1] is used at the two **ERA** levels of entities and attributes. By using a slight generalization for the classification concept for entities, the unpublished paper “The **FoLE** Equivalence” [12] establishes the equivalence between between the classification and interpretation forms of FoLE.

---

\[ \begin{array}{c}
S_1 \overset{t_{11}}{\longrightarrow} S \overset{t_{22}}{\longrightarrow} S_2 \text{ and } \mathcal{A}_{\cdot \cdot} \\
\langle S_1, \mathcal{A}_{\cdot \cdot} \rangle \overset{(t_{11}, x_0 : \cdot)}{\longrightarrow} \langle S_1 +_{2} S_2, \mathcal{A} \rangle \overset{(t_{22}, x_0 : \cdot)}{\longrightarrow} \langle S_2, \mathcal{A}_{\cdot \cdot} \rangle \\
\langle S_1 +_{2} S_2, \mathcal{A} \rangle \overset{(t_{22}, x_0 : \cdot)}{\longrightarrow} \langle S_1 +_{2} S_2, \mathcal{A} \rangle \\
T \in \text{Tbl}_{\mathcal{A}}(S_1) \text{ and } T_2 \in \text{Tbl}_{\mathcal{A}}(S_2) \\
T_1 \equiv_{\mathcal{A}} T_2 \\
\end{array} \]

**Table 21. FoLE Outer-Join I/O**
Here, we review the various concepts that arise in the FOLE approach to relational algebra. We first review tables and their components: type domains, signatures and signed domains. We next review the completeness and co-completeness of the mathematical context of tables, ending with the definitions of sufficiency and adequacy that are useful for simplifying the input for relational operations.

A.1 Tabular Components

Signatures. A signature [10] is a list, which represents the header of a relational table; it provides typing for the tuples permitted in the table. A signature $S = \langle I, s, X \rangle$ consists of a set of sorts $X$, an indexing set (arity) $I$, and a map $I \rightarrow X$ from indexes $I$ to sorts $X$. A signature morphism (list morphism) $S' \xrightarrow{(h,f)} S$, from signature (list) $S' = \langle I', x', X' \rangle$ to signature (list) $S = \langle I, x, X \rangle$, consists of a sort function $X' \xrightarrow{f} X$ and an arity function $I' \xrightarrow{h} I$ satisfying the naturality condition $h \cdot s = s' \cdot f$. Let List denote the mathematical context of signatures and signature morphisms.

\[
\begin{array}{ccc}
I' & \xrightarrow{h} & I \\
\downarrow \quad s' & & \quad \downarrow s \\
X' & \xrightarrow{f} & X
\end{array}
\]

Fig. 37. Signature Morphism

Let $X$ be a fixed sort set. An $X$-signature $S = S$ is a signature with the sort set $X$. An $X$-signature morphism $S' = \langle S' \rangle \xrightarrow{h} S = S$ is a signature morphism with an identity sort function $X \xrightarrow{1_X} X$ and satisfying the naturality condition $h \cdot s = s'$. Let $\text{List}(X)$ denote the fiber context of $X$-signatures and $X$-signature morphisms. For fixed arity function $I' \xrightarrow{h} I$, the naturality condition $h \cdot s = s' \cdot f$ gives two alternate and adjoint fiber passages: $\text{List}(X') \xrightarrow{\text{list}(f)} \text{List}(X)$. In terms of fibers, a signature morphism consists of a sort function $X' \xrightarrow{f} X$ and either a morphism $S' \xrightarrow{h} f^*(S)$ in the fiber context $\text{List}(X')$ or adjointly a morphism $\Sigma_f(S') \xrightarrow{h} S$ in the fiber context $\text{List}(X)$.  

60 As we remarked in [10], the use of lists for signatures (and tuples) follows Codd’s recommendation to use attribute names to index the tuples of a relation instead of a numerical ordering.

61 $f^*(S)$ is the pullback of signature $S$ back along sort function $X' \xrightarrow{f} X$ and $\Sigma_f(S')$ is the composition of signature $S'$ forward along sort function $X' \xrightarrow{f} X$.

62 For more on this see § 2.1 of [10].
Relational Operations in **FOLE** 73

**Type Domains.** In the **FOLE** theory of data-types [8], a classification \( A = \langle X, Y, \models_A \rangle \) [1] is known as a type domain. A type domain [10] consists of a set of sorts (data types) \( X \), a set of data values (instances) \( Y \), and a binary (classification) relation \( \models_A \) between data values and sorts. The extent of any sort (data type) \( x \in X \) is the subset \( \text{ext}_A(x) = A_x = \{ y \in Y \mid y \models_A x \} \). Hence, a type domain is equivalent to be a sort-indexed collection of subsets of data values \( A = \{ A_x \subseteq Y \mid x \in X \} \); or more abstractly, \( X \xrightarrow{A} \varphi Y : x \mapsto \text{ext}_A(x) = A_x \). By being so explicit, we have more exact control over the data.

---

Some examples of data-types useful in databases are as follows. The real numbers might use sort symbol \( \mathbb{R} \) with extent \( \{ -\infty, \ldots, 0, \ldots, \infty \} \). The alphabet might use sort symbol \( \varnothing \) with extent \( \{a, b, c, \ldots, x, y, z\} \). Words, as a data-type, would be lists of alphabetic symbols with sort symbol \( \varnothing \) and extent \( \{a, b, c, \ldots, x, y, z\} \). The periodic table of elements might use sort symbol \( E \) with extent \( \{\text{H}, \text{He}, \text{Li}, \text{Be}, \ldots, \text{Hs}, \text{Mt}\} \). Of course, chemical elements can also be regarded as entities [8] in a database with various attributes such as name, symbol, atomic number, atomic mass, density, melting point, boiling point, etc.

---

For a given type domain \( A \), the list classification \( \text{List}(A) = \langle \text{List}(X), \text{List}(Y), \models_{\text{List}(A)} \rangle \) has \( X \)-signatures as types and \( Y \)-tuples as instances, with classification by common arity and universal \( A \)-classification: a \( Y \)-tuple \( x \) is classified by an \( X \)-signature \( S \) when \( J = I \) and \( t_k \models_A s_k \) for all \( k \in J = I \). Hence, a list type domain is equivalent to be a header-indexed collection of subset of tuples \( \text{List}(A) = \{ \text{List}(A)\mathcal{S} \subseteq \text{List}(Y) \mid \mathcal{S} \in \text{List}(X) \} \); or more abstractly, \( \text{List}(X) \xrightarrow{\text{List}(A)} \varphi \text{List}(Y) : \mathcal{S} \mapsto \text{ext}_{\text{List}(A)}(\mathcal{S}) = \text{List}(A)\mathcal{S} \).

![Fig. 38. **FOLE** Type Domain Morphism](image)

In the **FOLE** theory of data-types [8], an infomorphism \( A' \xrightarrow{(f,g)} A \) [1] is known as a type domain morphism, and consists of a sort function \( X' \xrightarrow{f} X \) and a data value function \( Y' \xrightarrow{g} Y \) that satisfy any of the following equivalent

---

\[63\] In [5] a classification is known as a formal context.

\[64\] In particular, when \( I = 1 \) is a singleton, an \( X \)-signature \( \langle 1, s \rangle \) is the same as a sort \( s() = x \in X \), a \( Y \)-tuple \( \langle 1, t \rangle \) is the same as a data value \( t() = y \in Y \), and \( \text{ext}_{\text{List}(A)}(1, s) = \text{List}(A)_{1,s} = A_x \).
conditions for any source sort \( x' \in X' \) and target data value \( y \in Y \):

\[
\begin{align*}
g(y) & \models_{A'} x' \quad \text{iff} \quad y \models_A f(x'); \\
g(y) \in A'_{x'} & = \text{ext}_{A'}(x') \quad \text{iff} \quad y \in A_{f(x')} = \text{ext}_A(f(x')); \\
g^{-1}(A'_{x'}) & = g^{-1}(\text{ext}_{A'}(x')) = \text{ext}_A(f(x')) = A_{f(x')}.
\end{align*}
\]

(4)

The condition on a type domain morphism gives two alternate definitions.\(^{65}\)

**Yin:** (RHS Fig. 39) In terms of fibers, a type domain morphism consists of a data value function \( Y' \overset{g}{\to} Y \), and an infomorphism \( g^{-1}(A') \overset{(f,1_y)}{\longrightarrow} A \) from the (Yin) classification \( g^{-1}(A') = \langle X',Y,\models_{g^{-1}(A')} \rangle \) defined by \( y \models_{g^{-1}(A')} x' \) when \( g(y) \models_{A'} x' \).

**Yang:** (LHS Fig. 39) In terms of fibers, a type domain morphism consists of a sort function \( X' \overset{f}{\to} X \), and an infomorphism \( A' \overset{(1_x,g)}{\longrightarrow} f^{-1}(A) \) to the (Yang) classification \( f^{-1}(A) = \langle X',Y,\models_{f^{-1}(A)} \rangle \) defined by \( y \models_{f^{-1}(A)} x' \) when \( y \models_A f(x') \).

An infomorphism \( A' \overset{(1_x,g)}{\longrightarrow} f^{-1}(A) \) is identical to the composite infomorphism

\[
A' \overset{(1_x,s)}{\longrightarrow} f^{-1}(A) = g^{-1}(A') \overset{(f,1_y)}{\longrightarrow} A.
\]

When the sort function is an injection \( X' \overset{f}{\to} X \), the target set \( X \) contains the source set \( X' \); hence, we can think of the set of sorts as becoming larger as we move to the right. Similarly, when the data value function is an injection \( Y' \overset{g}{\to} Y \), the target set \( Y \) is contained in the source set \( Y' \); hence, we can think of the set of data values as becoming smaller as we move to the right.

![Fig. 39. Type Domain Yin-Yang](image)

\(^{65}\) Yin-Yang is the interplay of two opposite but complementary forces that interact to form a dynamic whole.
Signed Domains. A signed domain [10] is a fundamental component used in the definition of database tables and in the database interpretation of FOLE. Signed domains are used to denote the valid tuples for a database header (signature).

A signed (headed/typed) domain \( \mathcal{D} = \langle S, A \rangle \) consists of a type domain \( A = \langle X, Y, \models_{A} \rangle \) and a signature (database header) \( S = \langle I, x, X \rangle \) with a common sort set \( X \). A signed domain morphism \( \mathcal{D}' = \langle S', A' \rangle \xrightarrow{(h,f,g)} \langle S, A \rangle = \mathcal{D} \) consists of a signature morphism \( S' = \langle I', x', X' \rangle \xrightarrow{(h,f)} \langle I, x, X \rangle = S \) and a type domain morphism \( A' = \langle X', Y', \models_{A'} \rangle \xrightarrow{(f,g)} \langle X, Y, \models_{A} \rangle = A \) with a common sort function \( X' \xrightarrow{f} X \). Let Dom denote the mathematical context of signed domains.

\[
\begin{align*}
\text{Fig. 40. FOLE Signed Domain Morphism}
\end{align*}
\]

A.2 Tables, Tabular Flow and Relations.

The FOLE relational operations are defined on tables. The mathematical contexts of relations and tables are used for satisfaction and interpretation [8], relations for traditional interpretation and tables for database interpretation. In FOLE, both relational algebra and the tuple relational calculus are based upon the table concept.

**Definition 1.** There is a tuple passage [10] \( \text{tup} : \text{Dom}^{\text{op}} \rightarrow \text{Set} \).

The tuple passage \( \text{Dom}^{\text{op}} \xrightarrow{\text{tup}} \text{Set} \) maps a signed domain \( \langle S, A \rangle \) to its set of tuples \( \text{tup}(S,A) \), \(^{66}\) and maps a signed domain morphism \( \langle I_2, s_2, A_2 \rangle \xrightarrow{(h,f,g)} \langle I_1, s_1, A_1 \rangle \) to its tuple function \( \text{tup}(I_2, s_2, A_2) \xrightarrow{(h,f,g)} \text{tup}(I_1, s_1, A_1) \); or visually, \( \cdots g(t_h(i_2)) \cdots i_2 \in I_2 \xleftarrow{\cdots t_i} \cdots i_1 \in I_1 \).

\(^{66}\) This important concept can intuitively be regarded as the set of legal tuples under the database header \( S = \langle I, x, X \rangle \). It is define to be the extent in the list type domain \( \text{List}(A) \): \( \text{tup}(S,A) = \text{ext}_{\text{List}(A)}(S) = \{ (J, t) \in \text{List}(Y) : (J, t) \models_{\text{List}(A)} S \} \). Various notations are used for this concept depending upon circumstance: \( \text{tup}(S,A) = \text{tup}_S(A); \text{tup}_S(S) = \text{tup}_A(S); \text{tup}_S(A) = \text{tup}_S(A) = \text{tup}_S(Y, \models_A) \).
Tables. A FOLE table $\mathcal{T} = \langle D, K, t \rangle$ consists of a signed domain $D = \langle S, A \rangle$, a set $K$ of (primary) keys and a tuple function $K \xrightarrow{k} \text{tup}(D) = \text{tup}_A(S)$ mapping keys to $D$-tuples. A FOLE table morphism $\mathcal{T}' = \langle D', K', t' \rangle \xleftarrow{\langle h, f, g \rangle, k} \langle D, K, t \rangle = \mathcal{T}$ consists of a signed domain morphism $D' = \langle S', A' \rangle \xrightarrow{\langle h, f, g \rangle} \langle S, A \rangle = D$ and a key function $K' \xleftarrow{k} K$, which satisfy the naturality condition $k \cdot t' = t \cdot \text{tup}(h, f, g)$. Let $\text{Tbl}$ denote the mathematical context of tables.

We next show that the context of tables is a fibered context over signed domains; we first define the table fiber for fixed signed domain; we then define adjoint flow along signed domain morphisms.

Small Fibers. Let $D = \langle S, A \rangle = \langle S, A \rangle$ be a fixed signed domain. The fiber mathematical context of of $D$-tables is denoted by $\text{Tbl}(D) = \text{Tbl}_A(S) = \text{Tbl}_A(S)$. A $D$-table $\mathcal{T} = \langle K, t \rangle$ consists of a set $K$ of (primary) keys and a tuple function $K \xrightarrow{k} \text{tup}_A(S)$ mapping each key to its descriptor $A$-tuple of type (signature) $S$. A $D$-table morphism $\mathcal{T}' = \langle K', t' \rangle \xleftarrow{k} \langle K, t \rangle = \mathcal{T}$ is a key function $K' \xleftarrow{k} K$ that preserves descriptors by satisfying the naturality condition $k \cdot t' = t$. As we show in §3.1 on reflection, this means that we have the relation morphism condition $\varphi t'(K') \supseteq \varphi t(K)$; that is, all the tuples in $\mathcal{T}$ are tuples in $\mathcal{T}'$. Use the notation $\mathcal{T}' \supseteq \mathcal{T}$ for this assertion.

Tabular Flow Adjunction. As defined above, a table morphism $\mathcal{T}' = \langle D', K', t' \rangle \xleftarrow{\langle h, f, g \rangle, k} \langle D, K, t \rangle = \mathcal{T}$ consists of a signed domain morphism $D' \xrightarrow{\langle h, f, g \rangle} D$ and a key function $K' \xleftarrow{k} K$ satisfying the naturality condition $k \cdot t' = t \cdot \text{tup}(h, f, g)$. This condition gives two alternate and adjoint definitions. In terms of fibers,

\[
\begin{array}{ccc}
K' & \xleftarrow{k} & K \\
\downarrow{\text{tup}(D')} & & \downarrow{\text{tup}(D)} \\
T' & \xleftarrow{\text{tbl}(h, f, g)(\mathcal{T})} & \mathcal{T} \\
\end{array}
\]

\[
\begin{array}{ccc}
K' & \xleftarrow{k} & K \\
\downarrow{\text{tup}(D')} & & \downarrow{\text{tup}(D)} \\
T' & \xleftarrow{\text{tbl}(h, f, g)(\mathcal{T}') \leftarrow \mathcal{T}} & \text{Tbl}(D) \\
\end{array}
\]

Fig. 41. Table Morphism: Signed Domain

\[\text{FOLE Table} \] [10].
phism $\text{tbl}(h, f, g)(\mathcal{T}') \xleftarrow{k'} \mathcal{T}$ in the fiber context $\text{Tbl}(\mathcal{D})$. The $\mathcal{D}'$-table morphism $\mathcal{T}' \xleftarrow{k} \text{tbl}(h, f, g)(\mathcal{T})$ is the composition (RHS of Fig. 41) of the fiber morphism $\text{tbl}(h, f, g)(\mathcal{T}') \xleftarrow{k'} \mathcal{T}$ with the $\mathcal{T}'$th counit component $\mathcal{T}' \xleftarrow{} \text{tbl}(h, f, g)(\mathcal{T}')$ for the fiber adjunction

$$\text{Tbl}(\mathcal{D}') \xleftarrow{\langle \text{tbl}(h, f, g) \dashv \text{tbl}(h, f, g) \rangle} \text{Tbl}(\mathcal{D}).$$

(5)

This fiber adjunction is a component of the signed domain indexed adjunction of tables $\text{Dom}^{\text{op}} \xrightarrow{\text{tbl}} \text{Adj}$. For more on this see the paper [10].

Tuple Passage Factorization. Since a signed domain morphism $\langle \mathcal{S}', \mathcal{A}' \rangle \xrightarrow{\langle h, f, g \rangle} \langle \mathcal{S}, \mathcal{A} \rangle$ factors into three parts

$$\langle \mathcal{S}', \mathcal{A}' \rangle \xrightarrow{\langle 1, 1, 1 \rangle} \langle \mathcal{S}', g^{-1}(\mathcal{A}) \rangle \xrightarrow{\langle 1, f, 1 \rangle} \langle \Sigma_f(\mathcal{S}'), \mathcal{A} \rangle \xrightarrow{\langle h, 1, 1 \rangle} \langle \mathcal{S}, \mathcal{A} \rangle,$$

its tuple function of Def. 1 factors into five parts:

![Diagram](https://via.placeholder.com/150)

both linearly and 2-dimensionally.

**Proposition 11.** The tuple function $\text{tup}_{\mathcal{S}'}(g^{-1}(\mathcal{A}')) \xleftarrow{\text{tup}_{\mathcal{S}'}(g^{-1}(\mathcal{A}'))} \text{tup}_{\mathcal{A}}(\Sigma_f(\mathcal{S}'))$ is a hidden function. Composition with it gives the identities in Fig. 42.

**Proof.** $\text{tup}_{f^{-1}(\mathcal{A})}(\mathcal{S}') = \text{tup}_{\mathcal{A}}(\Sigma_f(\mathcal{S}'))$ by the Yin-Yang in §A.1. To see that $\text{tup}_{f^{-1}(\mathcal{A})}(\mathcal{S}') = \text{tup}_{\mathcal{A}}(\Sigma_f(\mathcal{S}')) = \text{tup}_{\mathcal{A}}(\mathcal{S}', f)$, let $(I', t) \in \text{List}(\mathcal{Y})$ be any $\mathcal{Y}$-tuple. Then $(I, t) \in \text{tup}_{f^{-1}(\mathcal{A})}(\mathcal{S}')$ iff $\forall v \in v(t) \ | [f(\mathcal{S}')] = f^{-1}(\mathcal{A}) \ s'(i') \ | [f(\mathcal{S}')] = \mathcal{A} f(s'(i')) \ | [f(\mathcal{S}')] = \mathcal{A} f(s'(i'))$. Hence, $\text{tup}_{\mathcal{A}}(g^{-1}(\mathcal{A}')) = \text{tup}_{\mathcal{A}}(\Sigma_f(\mathcal{S}'))$. The tuple function is the identity $\text{tup}_{\mathcal{S}'}(g^{-1}(\mathcal{A}')) \xleftarrow{\text{tup}_{\mathcal{S}'}(g^{-1}(\mathcal{A}'))} \text{tup}_{\mathcal{A}}(\Sigma_f(\mathcal{S}'))$, since defined as pre-post composition with the identity function.

In §3 the basic operations of projection/inflation and restriction/expansion are explained in terms of the encircled tuple functions in Fig. 42. The adjoint flow factorization (Disp. 3) is visualize in Fig. 43.

---

68 For (1) the definitions of the tuple bridges $\hat{\tau}_{(f, g)}$, $\tilde{\tau}_{(f, g)}$, and $\tau_{(h, f)}$ and (2) the signature/type domain tuple function factorization, see the paper [10].

69 For tuple set notation, see footnote 66.
Examples include a partial order of subsets of \(R\) with inclusion tuple function \(\sigma:\) and an inclusion key function \(\rho:\) of the relation \(R\). Auxiliary relations. Here are some examples of auxiliary relations.

**Example 1.** \(\sigma_{\neq, \#}(T)\): Let \(\hat{x} \in X\) be a sort whose data-type \(A_{\hat{x}} = \text{tup}_A(1, \hat{x}) = P\) contains the elements of some binary relation \(\langle P, \theta\rangle\) with \(\theta \subseteq P \times P\). Then, \(X\)-signature (header) \(\langle 1, \hat{x} \rangle\) with sort function \(1 : P \rightarrow X\) has the extent

\[\langle 1, \hat{x} \rangle \rightarrow X\]
Generalized selection, written as $\sigma_\varphi(T)$, where $\varphi$ is a propositional formula that consists of atoms as allowed in the normal selection and the logical operators $\land$ (and), $\lor$ (or) and $\neg$ (negation). This selection chooses all those tuples in $T$ for which $\varphi$ holds.
A.3 Completeness and Co-completeness.

The following proposition using comma contexts appears as Prop. 7 in “The FOLE Table” [10].

**Proposition 12.** The contexts of signed domains $\text{Dom}$ and tables $\text{Tbl}(\mathcal{D}) = \text{Tbl}_A(\mathcal{S})$ are associated with the following passage opspans:

| comma context | passage opspan |
|---------------|----------------|
| $\text{Dom} = (\text{Set} \downarrow \text{sort})$ | $\text{Set} \xrightarrow{\text{arity}} \text{Dom} \xrightarrow{\text{data}} \text{Cls}$ |
| $\text{Tbl}(\mathcal{D}) = (\text{Set} \downarrow \text{tup}(\mathcal{D}))$ | $\text{Set} \xrightarrow{\text{key}(\mathcal{D})} \text{Set} \xrightarrow{\text{tup}(\mathcal{D})} \text{set}$ |

respectively. Hence, they are complete and co-complete and their projections \(^{72}\)

$$
\begin{align*}
\text{Set} & \xrightarrow{\text{arity}} \text{Dom} \xrightarrow{\text{data}} \text{Cls} \\
\text{Set} & \xleftarrow{\text{key}(\mathcal{D})} \text{Tpl}(\mathcal{D}) \xrightarrow{\text{1}}
\end{align*}
$$

are continuous and co-continuous.

**Proof.** The contexts $\text{Set}$, $\text{Cls}$ and $\text{1}$ are complete and co-complete; the passages $\text{Set} \xrightarrow{\text{arity}} \text{Dom} \xrightarrow{\text{data}} \text{Cls}$, $\text{Set} \xrightarrow{\text{key}(\mathcal{D})} \text{Set}$, and $\text{1} \xrightarrow{\text{tup}(\mathcal{S})} \text{Set}$ are continuous; and the passage $\text{Set} \xrightarrow{\text{set}} \text{Set}$ is continuous and co-continuous.

The following proposition using the Grothendieck construction appears as Prop. 9 in “The FOLE Table” [10]. This provides a framework for relational operations.

**Proposition 13.** The fibered context of tables $\text{Tpl}$ is complete and co-complete and the projection $\text{Tpl} \xrightarrow{\text{dom}} \text{Dom}^{\text{op}}$ is continuous and co-continuous.

**Proof.** The fibered context of tables $\text{Tpl} \xrightarrow{\text{dom}} \text{Dom}^{\text{op}}$ is the Grothendieck construction of the signed domain indexed adjunction $\text{Dom}^{\text{op}} \xrightarrow{\text{th}l} \text{Adj}$ (Thm. 3 in [10]).

\(^{72}\) The completeness and co-completeness of the small fiber context of tables $\text{Tpl}(\mathcal{D}) = \text{Tpl}_A(\mathcal{S})$ is represented by the data-type operations of §3.2.
A.3.1 Completeness. For the completeness of \( \text{Tbl} \) and the continuity of \( \text{Tbl} \xrightarrow{\text{dom}^{op}} \text{Dom}^{op} \), use the right adjoint pseudo-passage \( \text{Dom} \xrightarrow{\text{tbl}} \text{Cxt} \) (See Fact. 3 of [10]): the indexing context \( \text{Dom}^{op} \) is complete; the fiber context \( \text{Tbl}(\mathcal{D}) \) of §3.2 is complete (meets exist) for each signed domain \( \mathcal{D} \); and, the fiber passage \( \text{Tbl}(\mathcal{D}') \xrightarrow{\text{tbl}(h,f,g)} \text{Tbl}(\mathcal{D}) \) of §3.3.3 (restrict-inflate operation) is continuous for each signed domain morphism \( \mathcal{D}' \xrightarrow{(h,f,g)} \mathcal{D} \).

Given a diagram of tables \( \mathcal{I} \xrightarrow{T} \text{Tbl} \) consisting of an indexed (and linked) collection of tables \( \{ \mathcal{T}_i = T(i) \in \text{Tbl} \mid i \in \mathcal{I} \} \) with links \( \mathcal{T}_i \xleftarrow{\langle(h_i,f_i,g_i),k_i \rangle} \mathcal{T}_i \) for each \( i' \xleftarrow{p} i \) in \( \mathcal{I} \), Prop. 13 states that the diagram of tables \( T \) has a limit table \( \prod T \) with projection bridge \( \Delta(\prod T) \xrightarrow{\pi} T \) consisting of an indexed collection of table morphisms \( \{ \prod T \xrightarrow{(h_i,f_i,g_i),k_i} \mathcal{T}_i \mid i \in \mathcal{I} \} \) satisfying naturality.

Input: We use a sufficient subset of tables (Def. 2)
\[
\{ \mathcal{T}_i = T(i) \in \text{Tbl} \mid i \in \mathcal{I} \subseteq \text{obj}(\mathcal{I}) \}
\]

reachable to other tables in the collection \( \{ \mathcal{T}_i = T(i) \in \text{Tbl} \mid i \in \mathcal{I} \} \).

Constraint: The projection \( \text{Tbl}^{op} \xrightarrow{\text{dom}^{op}} \text{Dom}^{op} \) maps the diagram of tables \( T \) to a diagram of signed domains \( I^{op} \xrightarrow{D} \text{Dom}^{op} \) consisting of an indexed collection \( \mathcal{D}_i = D(i) \in \text{Dom} \mid i \in \mathcal{I} \) with links \( \{ \mathcal{D}_i \xrightarrow{(h_i,f_i,g_i),k_i} \mathcal{D}_i \mid i \in \mathcal{I} \} \).

Construction: Since \( \text{Tbl} \xrightarrow{\text{dom}^{op}} \text{Dom}^{op} \) is continuous (Prop. 13), it maps the limit table with projection bridge to the colimit signed domain \( \prod D = \mathcal{D} \) with injection bridge \( \Delta(\mathcal{D}) \xleftarrow{\mathcal{I}} D \) consisting of an indexed collection of signed domain morphisms \( \{ \mathcal{D} \xleftarrow{(h_i,f_i,g_i),k_i} \mathcal{D}_i \mid i \in \mathcal{I} \} \) satisfying naturality.

Output: Using the completeness aspect of Prop. 13, the limit table is the meet (intersection) in the fiber context \( \text{Tbl}(\mathcal{D}) \) of the restriction-inflation of the component tables \( \mathcal{T}_i \) along the injections \( \{ \mathcal{D}_i \xrightarrow{(h_i,f_i,g_i),k_i} \mathcal{D}_{i'} \mid i \in \mathcal{I} \} \).
- For each index \( i \in \mathcal{I} \), restriction/inflation \( \text{Tbl}(\mathcal{D}_i) \xrightarrow{\text{tbl}(h_i,f_i,g_i)} \text{Tbl}(\mathcal{D}) \) (§3.3.3) maps the table \( \mathcal{T}_i \in \text{Tbl}(\mathcal{D}_i) \) by pullback to the table \( \mathcal{T}_i \xleftarrow{(h_i,f_i,g_i),k_i} \mathcal{T}_i \in \text{Tbl}(\mathcal{D}) \). This defines a table morphism \( \mathcal{T}_i = \langle K_i, t_i \rangle \xleftarrow{(h_i,f_i,g_i),k_i} \langle K_i, t_i \rangle = \mathcal{T}_i \).
- Intersection (§3.2) of the tables \( \{ \mathcal{T}_i \mid i \in \mathcal{I} \} \) in the fiber context \( \text{Tbl}(\mathcal{D}) \) defines the generic meet \( \prod T = \mathcal{T} \xleftarrow{\mathcal{T}_i} \mathcal{T} \mid i \in \mathcal{I} \) = \( \langle K, t \rangle \), resulting in the discrete multi-span (cone) \( \{ \mathcal{T}_i \xleftarrow{(h_i,f_i,g_i),k_i} \mathcal{T} \mid i \in \mathcal{I} \} \). Restriction-inflation composed with meet defines the span of table morphisms \( \{ \mathcal{T}_i \xleftarrow{(h_i,f_i,g_i),k_i} \prod T \} \).
Aside: To construct the table $\prod T$ it is not necessary to use all of the tables and table morphisms in the indexed collection above. Let $T' \xrightarrow{(h,f,g,k)} T_i$ be part of the diagram $T$ satisfying the naturality condition $\pi_i = \pi_i \circ T(e)$ for some morphism $i' \subseteq i$ in $I$. Let $D' \xrightarrow{(h,f,g,k)} D_i$ be its signed domain morphism satisfying the naturality condition $\lambda_i = D(e) \circ \lambda_i$.

Using adjoint flow (Disp. 5), the table morphism $\pi_i \xrightarrow{(h,f,g)} \pi_i$ implies existence of a morphism $\text{tbl}(h,f,g)(T') \xrightarrow{\lambda_i} T_i$ in the fiber context $\text{Tbl}(D)$. For all practical purposes, by reflection (§3.1) we essentially have the inclusion $\text{tbl}(h,f,g)(T') \supseteq T_i$. Applying restriction-inflation to the naturality condition $\lambda_i = D(e) \circ \lambda_i$, get $\text{tbl}(\lambda_i) = \text{tbl}(h,f,g) \cdot \text{tbl}(\mu)$. Hence, $\bar{T}' = \text{tbl}(\lambda_i)(T') = \text{tbl}(\mu)(\text{tbl}(h,f,g)(T')) \supseteq \text{tbl}(\lambda_i) = \bar{T}_i$. Thus, it is clear that we do not need to use the table $\bar{T}'$ to define the meet $\prod T = \bigwedge \{ \bar{T}_i \mid i \in I \}$, but we do need at least one table, such as $\bar{T}_i$, from which $\bar{T}'$ can be connected. We only need a discrete collection of tables

$$\{ \bar{T}_i = T(i) \in \text{Tbl} \mid i \in I \subseteq \text{obj}(I) \}$$

reachable to other tables in the collection $\{ \bar{T}_i = T(i) \in \text{Tbl} \mid i \in I \}$.

**Definition 2.** We will call such an collection of tables a sufficient collection. The minimal such collection can be called an adequate collection.

**Special Cases:** Three limits are of special interest.

- **Equalizer:** the constraint diagram is of shape $1 \rightarrow 0$. The subsets $\{0,1\}$ and $\{1\}$ are sufficient, with the subset $\{1\}$ being adequate. The subsets $\{0\}$ and $\emptyset$ are not sufficient. *Quotient* (§4.1) models equalizer, using the minimal table index set $\{1\}$ with just one table and no table morphisms.

- **Pullback:** the constraint diagram is of shape $1 \leftarrow 0 \rightarrow 2$. The subsets $\{0,1,2\}$ and $\{1,2\}$ are sufficient, with the subset $\{1,2\}$ being adequate. The subsets $\{0\}$, $\{1\}$, $\{2\}$ and $\emptyset$ are not sufficient. *Natural join* (§4.3) models pullback, using the minimal table index set $\{1,2\}$ with just two tables and no table morphisms.

- **Limit:** the constraint diagram is of shape $I$ with indexes $i \in \text{obj}(I)$ linked by morphisms $i \sim i'$. *Generic meet* (§4.4) models limit, using a sufficient (Def. 2) table index set $\{i \in I \subseteq \text{obj}(I)\}$ with no table morphisms. The linked collection of tables $I \xrightarrow{T} \text{Tbl}$ is replaced by the collection of tables $I \xrightarrow{T} \text{Tbl}$ with $\{T_i = T(i) \in \text{Tbl}(D) \mid i \in I \subseteq \text{obj}(I)\}$. 

A.3.2 Co-completeness. For the co-completeness of $\text{Tbl}$ and the co-continuity of $\text{Tbl} \xrightarrow{\text{dom}^{-1}} \text{Dom}^\text{op}$, use the left adjoint pseudo-passage $\text{Dom}^\text{op} \xrightarrow{\text{tbl}} \text{Cxt}$ (See Fact. 4 of [10]): the indexing context $\text{Dom}^\text{op}$ is co-complete; the fiber context $\text{Tbl}(\mathcal{D})$ of §3.2 is co-complete (joins exist) for each signed domain $\mathcal{D}$; and the fiber passage $\text{Tbl}(\mathcal{D}') \xrightarrow{\text{tbl}(h,f,g)} \text{Tbl}(\mathcal{D})$ of §3.3.3 (project-expand operation) is co-continuous for each signed domain morphism $\mathcal{D}' \xrightarrow{(h,f,g)} \mathcal{D}$. 

Given a diagram of tables $\mathbf{I} \xrightarrow{T} \text{Tbl}$ consisting of an indexed (and linked) collection of tables $\{T_i = T(i) \in \text{Tbl} \mid i \in \mathbf{I}\}$ with links $T_i \xleftarrow{(h_i,f_i,g_i)} T_i'$ for each $i' \xhookleftarrow{i}$ in $\mathbf{I}$, Prop. 13 states that the diagram of tables $T$ has a colimit table $\bigsqcup T$ with injection bridge $\Delta(\bigsqcup T) \xleftarrow{(h,f,g),k} T$, consisting of an indexed collection of table morphisms $\{\bigsqcup T \xleftarrow{(h,f,g),h_i,k_i} T_i \mid i \in \mathbf{I}\}$ satisfying naturality.

**Input:** We use a sufficient subset of tables (Def. 3)
$$\{T_i = T(i) \in \text{Tbl} \mid i \in \mathbf{I} \subseteq \text{obj}(\mathbf{I})\}$$
reachable from other tables in the collection $\{T_i = T(i) \in \text{Tbl} \mid i \in \mathbf{I}\}$.

**Constraint:** The projection $\text{Tbl}^\text{op} \xrightarrow{\text{dom}} \text{Dom}$ maps the diagram of tables $T$ to a diagram of signed domains $\mathbf{I}^\text{op} \xrightarrow{\mathcal{D}} \text{Dom}$ consisting of an indexed collection $\mathcal{D}_i = D(i) \in \text{Dom} \mid i \in I$ with links $\{D_i \xrightarrow{(h_i,f_i,g_i)} D_i \mid i \in \mathbf{I}\}$.

**Construction:** Since $\text{Tbl} \xrightarrow{\text{dom}^{-1}} \text{Dom}^\text{op}$ is co-continuous (Prop. 13), it maps the colimit table with injection bridge to the limit signed domain $\bigsqcup D = \mathcal{D}$ with projection bridge $\Delta(\mathcal{D}) \xrightarrow{(h,f,g),k} \mathcal{D}$ consisting of an indexed collection of signed domain morphisms $\{\mathcal{D} \xrightarrow{(h,f,g),h_i,k_i} D_i \mid i \in \mathbf{I}\}$ satisfying naturality.

**Output:** Using the co-completeness aspect of Prop. 13, the colimit table is the join (union) in the fiber context $\text{Tbl}(\mathcal{D})$ of the projection-expansion of the component tables $T_i$ along the projections $\{D_i \xleftarrow{(h_i,f_i,g_i)} \mathcal{D} \mid i \in \mathbf{I}\}$.

- For each index $i \in I$, projection-expansion $\text{Tbl}(\mathcal{D}_i) \xrightarrow{\text{tbl}(h_i,f_i,g_i)} \text{Tbl}(\mathcal{D})$ (§3.3.3) maps the table $T_i \in \text{Tbl}(\mathcal{D}_i)$ by composition to the table $\tilde{T}_i = \langle K_i, t_i \rangle \in \text{Tbl}(\mathcal{D})$. This defines a table morphism $T_i = \langle K_i, t_i \rangle \xleftarrow{(h_i,f_i,g_i),1_{K_i}} \langle K_i, t_i \rangle = T_i$.
- Union (§3.2) of the tables $\{\tilde{T}_i \mid i \in I\}$ in the fiber context $\text{Tbl}(\mathcal{D})$ defines the generic join $\bigsqcup T = \tilde{T} = \bigvee\{\tilde{T}_i \mid i \in I\} = \langle \tilde{K}, \tilde{t} \rangle$, resulting in the discrete
multi-opspan (cocone) \( \{ \bar{T}_i \xrightarrow{\iota_i} \bar{T} \mid i \in I \} \). Projection-expansion composed with join defines the opspan of table morphisms \( \{ \bar{T}_i \xrightarrow{\langle \bar{h}_i, \bar{f}_i, \bar{g}_i \rangle, \bar{k}_i \rangle} \bigoplus T \} \). 

**Aside:** To construct the table \( \bigsqcup T \) it is not necessary to use all of the tables and table morphisms in the indexed collection above. Let \( T_i \xrightarrow{\langle \bar{h}_i, \bar{f}_i, \bar{g}_i \rangle, \bar{k}_i \rangle} T \) be part of the diagram \( T \) satisfying the naturality condition \( T(e) \circ \iota_{i'} = \iota_i \) for some morphism \( i' \xleftarrow{e} i \) in \( I \). Let \( D_i \xrightarrow{\langle \bar{h}_i, \bar{f}_i, \bar{g}_i \rangle, \bar{k}_i \rangle} D \) be its signed domain morphism satisfying the naturality condition \( \pi_{i'} \circ D(e) = \pi_i \).

Using adjoint flow (Disp. 5), the table morphism \( T_i \xrightarrow{\langle \bar{h}_i, \bar{f}_i, \bar{g}_i \rangle, \bar{k}_i \rangle} T \) implies existence of a morphism \( T_i \xleftarrow{\langle \bar{h}_i, \bar{f}_i, \bar{g}_i \rangle, \bar{k}_i \rangle} \) \( \text{tbl}(h, f, g) \) \( (T_i) \) in the fiber context \( \text{tbl}(D') \). For all practical purposes, by reflection (§3.1) we essentially have the inclusion \( T_i \supseteq \text{tbl}(h, f, g)(T_i) \). Applying projection-expansion to the naturality condition \( \pi_{i'} \circ D(e) = \pi_i \), get \( \text{tbl}(\pi_i) = \text{tbl}(h, f, g) \cdot \text{tbl}(\pi_{i'}) \). Hence, \( \bar{T}_i = \text{tbl}(\pi_i)(T_i) \supseteq \text{tbl}(\pi_{i'})(T_i) = \text{tbl}(\pi_i)(T_i) = \bar{T}_i \). Thus, it is clear that we do not need to use the table \( T_i \) to define the join \( \bigsqcup T = \bigsqcup \{ \bar{T}_i \mid i \in I \} \), but we do need at least one table, such as \( T_i \), to which \( T_i \) can be connected. We only need a discrete collection of tables \( \{ \bar{T}_i = T(i) \in \text{Tbl} \mid i \in I \subseteq \text{obj}(I) \} \) reachable from other tables in the collection \( \{ T_i = T(i) \in \text{Tbl} \mid i \in I \} \).

**Definition 3.** We will call such a collection of tables a sufficient collection. The minimal such collection can be called an adequate collection.
Special Cases: Three colimits are of special interest.

- **Coequalizer:** the constraint diagram is of shape $1 \rightrightarrows 0$. The subsets $\{0,1\}$ and $\{1\}$ are sufficient, with the subset $\{1\}$ being adequate. The subsets $\{0\}$ and $\emptyset$ are not sufficient. Co-quotation (§5.1) models coequalizer, using the minimal table index set $\{1\}$ with just one table and no table morphisms.

- **Pushout:** the constraint diagram is of shape $1 \rightarrow 0 \xleftarrow{2} 2$. The subsets $\{0,1,2\}$ and $\{1,2\}$ are sufficient, with the subset $\{1,2\}$ being adequate. The subsets $\{0\}$, $\{1\}$, $\{2\}$ and $\emptyset$ are not sufficient. Data-type join (§5.3) models pushout, using the minimal table index set $\{1,2\}$ with just two tables and no table morphisms.

- **Colimit:** the constraint diagram is of shape $I$ with indexes $i \in \text{obj}(I)$ linked by morphisms $t' \leftarrow i$. Generic join (§5.4) models colimit, using a sufficient (Def. 3) table index set $\{i \in I \subseteq \text{obj}(I)\}$ with no table morphisms. The linked collection of tables $I \xrightarrow{T} Tbl$ is replaced by the collection of tables $I \xrightarrow{T} Tbl$ with $\{T_i = T(i) \in Tbl(D_i) \mid i \in I \subseteq \text{obj}(I)\}$.

References

1. Barwise, J., and Seligman, J.: *Information Flow: The Logic of Distributed Systems*. Cambridge University Press, Cambridge (1997).
2. Chen, P.: “The Entity-Relationship Model — Toward a Unified View of Data.” (1976). *ACM Trans. on Database Syst.* 1 (1): 9–36.
3. Codd, E.F.: “A Relational Model of Data for Large Shared Data Banks.” (1970). *Comm. of the ACM.* 13 (6): 377–387.
4. Codd, E.F.: “The Relational Model for Database Management (Version 2 ed.).” (1990). *Addison Wesley*, Boston: 371–388. ISBN 0-201-14192-2.
5. Ganter, B., and Wille, R.: *Formal Concept Analysis: Mathematical Foundations*. Springer, New York (1999).
6. Kent, R.E.: “Database Semantics.” (2011). Available online: [http://arxiv.org/abs/1209.3054](http://arxiv.org/abs/1209.3054).
7. Kent, R.E.: “The First-order Logical Environment.” In: Pfeiffer, H.D., Ignatov, D.I., Poelmans, J., and Nagarjuna G. (eds.) *Conceptual Structures in Research and Education*, LNCS (7735): 210–230. Springer, Heidelberg (2013). Available online: [arxiv.org/abs/1305.5240](http://arxiv.org/abs/1305.5240).
8. Kent, R.E.: “The ERA of FOLE: Foundation.” (2015). Available online: [arxiv.org/abs/1512.07430](http://arxiv.org/abs/1512.07430).
9. Kent, R.E.: “The ERA of FOLE: Superstructure.” (2016). Available online: [arxiv.org/abs/1602.04268](http://arxiv.org/abs/1602.04268).
10. Kent, R.E.: “The FOLE Table.” Available online: [arxiv.org/abs/1810.12100](http://arxiv.org/abs/1810.12100).
11. Kent, R.E.: “The FOLE Database”. Unpublished paper.
12. Kent, R.E.: “The FOLE Equivalence”. Unpublished paper.
13. Mac Lane, S., *Categories for the Working Mathematician*. Springer-Verlag (1971).
14. *The Information Flow Framework (IFF)*. The Standard Upper Ontology (SUO) working group IEEE P1600.1. Available online: [http://web.archive.org/web/20121008145548/http://suo.ieee.org/IFF/](http://web.archive.org/web/20121008145548/http://suo.ieee.org/IFF/).