Type theoretical databases

HENRIK FORSSELL, *Department of Informatics, University of Oslo, PO Box 1080, Blindern N-0316 Oslo, Norway.*

HÅKON ROBBESTAD GYLTERUD, *Department of Informatics, University of Oslo, PO Box 1080, Blindern N-0316 Oslo, Norway.*

DAVID I. SPIVAK, *Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.*
E-mail: hakon.gylterud@uib.no

Abstract

We show how the display-map category of finite (symmetric) simplicial complexes can be seen as representing the totality of database schemas and instances in a single mathematical structure. We give a sound interpretation of a certain dependent type theory in this model and show how it allows for the syntactic specification of schemas and instances and the manipulation of the same with the usual type-theoretic operations.

Keywords: Dependent type theory, Simplicial sets, Relational databases

1 Introduction

Databases being, essentially, collections of (possibly interrelated) tables of data, a foundational question is how to best represent such collections of tables mathematically in order to study their properties and ways of manipulating them. The relational model, essentially treating tables as structures of first-order relational signatures, is a simple and powerful representation. Nevertheless, areas exist in which the relational model is less adequate than in others. One familiar example is the question of how to represent partially filled out rows or missing information. Another, more fundamental perhaps, is how to relate instances of different schemas, as opposed to the relatively well understood relations between instances of the same schema. Adding to this, an increasing need to improve the ability to relate and map data structured in different ways suggests looking for alternative and supplemental ways of modelling tables, more suitable to 'dynamic' settings. It seems natural, in that case, to try to model tables of different shapes as living in a single mathematical structure, facilitating their manipulation across different schemas.

We investigate, here, a novel way of representing data structured in systems of tables which is based on simplicial sets and type theory rather than sets of relations and first-order logic.[1] Formally, we present a soundness theorem (Theorem 1) for a certain dependent type theory with respect to a rather simple category of (finite, abstract) simplicial complexes. An interesting type-theoretic feature of this is that the type theory has context constants, mirroring that our choice of 'display maps' does not include all maps to the terminal object. From the database perspective, however, the interesting

[1] We thank an anonymous referee for pointing out that using the categorical semantics of type theory to structure data was an explicit motivation even at its very conception (see (2; 3)).

Vol. 30, No. 1, © The Author(s) 2020. Published by Oxford University Press.

Advance Access Publication on 26 February 2020 doi:10.1093/logcom/exaa009

aspect is that this category can in a natural way be seen as a category of tables, collecting in a single mathematical structure—an indexed or fibered category—the totality of schemas and instances. It is the database perspective that motivates, or forces, our choice of display maps.

The representation can be introduced as follows. Let a schema S be presented as a finite set $\mathbf{A}$ of attributes and a set of relation variables over those attributes. One way of allowing for partially filled out rows is to assume that whenever the schema has a relation variable R, say over attributes $A_0, \ldots, A_n$, it also has relation variables over all non-empty subsets of $\{A_0, \ldots, A_n\}$. So a partially filled out row over R is a full row over such a 'partial' relation, or 'part-relation', of R. To this, we add the requirement that the schema does not have two relation variables over exactly the same attributes. This requirement means that a relation variable can be identified with the set of its attributes. Together with the first condition, this means that the schema can be seen as a downward closed sub-poset of the positive power set of the set of attributes $\mathbf{A}$. Thus a schema is an (abstract) *simplicial complex*—a combinatorial and geometric object familiar from algebraic topology.

The key observation is now that an instance of the schema S can also be regarded as a simplicial complex, by regarding the data as attributes and the tuples as relation variables. Accordingly, an instance over S is a schema of its own, and the fact that it is an instance of S is 'displayed' by a certain projection to S. Thus the category $\mathbb{S}$ of finite simplicial complexes and morphisms between them form a category of schemas which includes, at the same time, all instances of those schemas; where the connection between schema and instance is given by a collection D of maps in $\mathbb{S}$ called *display* maps.

We show, essentially, that $\mathbb{S}$ together with this collection D of maps form a so-called *display-map category* (8), a notion originally developed in connection with categorical models of dependent type theory. First, this means that the category $\mathbb{S}$ has a rich variety of ready-made operations that can be applied to schemas and instances. For example, the so-called dependent product operation can be seen as a generalization of the natural join operation. Second, it is a model of dependent type theory. We specify a dependent type theory and a sound interpretation that interprets contexts as schemas and types as instances. This interpretation is with respect to the display-map category $(\mathbb{S}, D)$ in its equivalent form as an indexed category. We introduce context constants interpreted as distinguished single relation variable schemas (or *relation schemas* in the terminology of (1)), reflecting the special status of such schemas.

The type theory allows for the syntactic specification of both schemas and instances. The elements of type-theoretically defined operations on these, such as the natural join, can be formally derived in the type theory. Accordingly, we see this work as being among first steps towards establishing closer links between databases and programming languages—here in the form of dependent theories. Towards this end, we put some emphasis on showing that the model can be equipped with a type-theoretic *universe*. An infinite instance coding all finite instances of a schema, the universe allows reasoning generically about classes of instances of in the type theory itself, without having to resort to the metalanguage. Thus it provides the basis for precise, formal definitions and analyses of further database-theoretic notions (such as *query*).

The representation of tables of data presented here takes the view that tables are collections of tuples, as in the relational model. Here, this is tightly linked (in the sense of Lemma 2) with the requirement that a schema does not have more than one relation variable over a given set of attributes.[2] An alternate view that tables are collections of 'keys', with the possibility that two keys can represent the same data, and that schemas can have any number of relation variables

[2] We do not believe that this restriction is of major practical significance, see Example 16.

with the same attributes, can be pursued by using (semi-)simplicial sets rather than simplicial complexes (see (11)). We take the former view in this paper as it gives a rather clear and simple picture of the representation of collections of tuples 'simplicially'. Also, we do not discuss extra levels of structure, like data types (as is done in (11)), but focus on the representation of schemas and instances as simplicial complexes and their type-theoretic aspects. Section 2 introduces the category of simplicial complexes and display maps and the representation of tables in that setting. We strive for the presentation to be as self-contained as possible and assume for the most part only knowledge of the very basic notions of category theory, such as category, functor and natural transformation. Section 2.2 contains the essential constructions and lemmas needed for the proof of the main soundness theorem (Theorem 1). That theorem and the presentation of the type theory is given in Section 3. Section 4 presents the Universe, type-theoretically and semantically, and gives some illustration of its use. Finally, Section 5 contains some additional examples, informally presented. These examples are intended to supplement Section 2, to give a feel for the 'simplicial' representation of tables. and to indicate uses of this representation to model such things as updates or missing data. This section can be read in parallel with Section 2.

2 The model

2.1 Complexes, schemas and instances

We fix the following terminology and notation, adjusting the standard terminology somewhat for our purposes. A background on simplicial complexes and simplicial sets can be found in e.g. (5; 6). For symmetric simplicial sets, see (7).

A simplicial complex can be thought of as a set of vertices together with a collection of faces, where the set of faces is a downward closed set of finite non-empty, non-singleton subsets of vertices. More formally, we use the following definitions.

Based poset. Let X be a poset. A subset $B \subseteq X$ is called a *basis of* X if the following hold:

1. for all $x, y \in X$, one has $x \leq y$if and only if $B_{\leq x} \subseteq B_{\leq y}$, where $B_{\leq x} = (\downarrow x) \cap B = \{z \in B | z \leq x\}$;
2. for all $g, h \in B$, one has $g \leq h \Rightarrow g = h$; and
3. $B_{\leq x}$ is inhabited and of bounded finite size for all $x \in X$. That is, there exists an $n \in \mathbb{N}$ such that for all $x \in X$, $1 \leq |B_{\leq x}| \leq n$.

If X has a basis, one sees easily that the basis is unique, and we say that X is a *based poset*.
Dimension. Let X be a based poset with basis B. Then define $X_n := \left\{x \in X \;\middle|\; |B_{\leq x}| = n+1\right\}$ to be the set of faces *of dimension n*. In particular, the set of vertices $X_0 = B$ can be considered as faces of dimension 0.
Simplicial complex. A based poset X is called a *simplicial complex* if for all $x \in X$ and $Y \subseteq B_{\leq x}$ there exists $y \in X$ such that $B_{\leq y} = Y$.

Example 2.1
For a set S, the poset $\mathcal{P}_{+}^{\leq n}(S)$ of non-empty subsets of S of cardinality at most n is a simplicial complex. So is any downward closed subposet of $\mathcal{P}_{+}^{\leq n}(S)$. If X is a simplicial complex, it is isomorphic to one of this (latter) form by the map $x \mapsto B_{\leq x}$.

Simplicial schema. We say that a poset X is a *simplicial schema* if X^{op}—the poset obtained by reversing the ordering—is a simplicial complex. The elements of X_0 are called *attributes* and the elements of X_{n+1} are called *relation variables*. We consider a simplicial schema as a category and use arrows $\delta^x_y : x \longrightarrow y$ to indicate order. Thus the arrow δ^x_y exists iff $y \leq x$ in the simplicial complex X^{op}. We reserve the use of arrows to indicate order in the schema X and $\leq$ to indicate the order in the complex X^{op}. We use the notation $B_{\leq x}$ also in connection with schemas, where it means, accordingly, the set of attributes A such that there is an arrow δ^x_A.

Morphisms. Suppose that X and Y are based posets with bases B and C, respectively. A poset morphism $f : X \longrightarrow Y$ is called *based* if for all $x \in X$, we have $f(B_{\leq x}) = C_{\leq f(x)}$. A morphism of simplicial complexes is a based poset morphism. A morphism of simplicial schemas is a morphism of posets $f : X \longrightarrow Y$ such that $f^{\mathrm{op}} : X^{\mathrm{op}} \longrightarrow Y^{\mathrm{op}}$ is a morphism of simplicial complexes. Note that a based poset morphism $f : X \longrightarrow Y$ is completely determined by its restriction to the basis $f_0 : X_0 \longrightarrow Y_0$.

Display maps. A morphism $f : X \longrightarrow Y$ of simplicial schemas is a *display map* if f restricts to a family of maps $f_n : X_n \to Y_n$ (one could say that it 'preserves dimension'). It is straightforward to see that this is equivalent to the condition that for all $x \in X^{\mathrm{op}}$, the restriction $f \upharpoonright_{(\downarrow x)}: (\downarrow x) \to (\downarrow f(x))$ is an isomorphism of sets (equivalently, of simplicial complexes).

REMARK 2.2
With respect to the usual notion of schema, a simplicial schema X can be thought of as given in the usual way by a finite set of attributes $X_0 = \{A_0, \dots, A_{n-1}\}$ and a set of relational variables $X = \{R_0, \dots R_{m-1}\}$, each with a specification of column names in the form of a subset of X_0, but with the restrictions (i) that no two relation variables are over exactly the same attributes and (ii) for any nonempty subset of the attributes of a relation variable there exists a relation variable over (exactly) those attributes.

Large and small, the categories $\mathbb{S}$ and $\mathbb{S}_d$. Since we aim to represent database schemas and instances, we are interested primarily in the finite case. However, we shall need to consider the infinite case in connection with type theoretical universes. Say that a simplicial schema or complex is *small* if it is finite and *large* otherwise. For simplicity of presentation, we restrict to the finite case in the rest of this section. The restriction is, however, mostly inessential, the definitions and lemmas generalize to the infinite case. Let $\mathbb{S}$ be the category of small simplicial schemas and morphisms and $\mathbb{S}_d$ the subcategory of small schemas and display maps. In the sequel, we shall drop the word 'simplicial' and simply say 'complex' and 'schema'.

Simplices. The category $\mathbb{S}_d$ contains in particular (the opposites of) the n-simplices Δ_n and the face maps. Recall that the n-simplex Δ_n is the complex given by the full positive power set on $[n] = \{0, \dots, n\}$. A face map $d^n_i : \Delta_n \longrightarrow \Delta_{n+1}$ between two simplices is the based poset morphism defined by $k \mapsto k$, if $k < i$ and $k \mapsto k+1$ else. These satisfy the simplicial identity $d^{n+1}_i \circ d^n_j = d^{n+1}_{j-1} \circ d^n_i$ if $i < j$. As a schema, Δ_n is the schema of a single relation on $n+1$ attributes named by numbers $0, \dots, n$ (and all its 'generated' sub-relations). A face map $d^n_i : \Delta_n \longrightarrow \Delta_{n+1}$ is the inclusion of the relation $[n+1] - \{i\}$ into $[n+1]$. These schemas and morphisms play a special role in Section 3 where they are used to specify general schemas and instances. The permutations of Δ_n are also in $\mathbb{S}_d$; we have not assumed that attributes are ordered and that (display) maps preserve order.

EXAMPLE 2.3
Let S be the schema with attributes A, B, C and relation variables $R : AB$ and $Q : BC$. From a 'simplicial' point of view, S is the category given below left. It can also be regarded, more

geometrically, as the ‘horn’ below right.

$$\begin{tikzcd}
A & R \arrow[l, "\delta^R_A"'] \arrow[r, "\delta^R_B"] & B & Q \arrow[l, "\delta^Q_B"'] \arrow[r, "\delta^Q_C"] & C
\end{tikzcd}
\qquad
\begin{tikzcd}
 & C \arrow[d, dash, "Q"] \\
A \arrow[r, dash, "R"'] & B
\end{tikzcd}$$

For another example, the 2-simplex Δ_2 can be seen as a schema on attributes 0,1 and 2, with relation variables $\{0,1,2\}$, $\{0,1\}$, $\{0,2\}$ and $\{1,2\}$. The function f_0 given by $A \mapsto 0$, $B \mapsto 1$ and $C \mapsto 2$ defines a morphism $f : S \longrightarrow \Delta_2$ of schemas/complexes. f is a display map. For an example of a display map that is not an inclusion, consider the morphism $f' : S \longrightarrow \Delta_1$ defined by $A \mapsto 0$, $B \mapsto 1$ and $C \mapsto 0$

Relational instances, the categories Rel(X). Let X be a schema. A functor $F : X \longrightarrow$ **FinSet** from X to the category of finite sets and functions can be regarded as an instance of the schema X. The set $F(x)$ can be regarded as a set of ‘keys’ or ‘row-names’; for $A \in B_{\leq x}$ the ‘value’ $k[A]$ of such a key $k \in F(x)$ at attribute A is the element $k[A] := F(d^x_A)(k) \in F(A)$. Accordingly, there is a mapping $F(x) \longrightarrow \prod_{A \in B_{\leq x}} F(A)$ defined by mapping k to the function $A \mapsto k[A]$. For arbitrary F, this mapping is not 1–1, that is, there can be distinct keys with the same values at all attributes. We say that F is a *relational instance* if this does not happen. That is to say, a relational instance is a functor $F : X \longrightarrow$ **FinSet** such that for all $x \in X$, the functions $\left\{F(\delta^x_A) \;\middle|\; A \in B_{\leq x}\right\}$ are jointly injective. Let Rel(X) be the category of relational instances and natural transformations between them. (Notice that a natural transformation between relational instances is the same thing as a homomorphism.)

EXAMPLE 2.4
Let S be the schema of Example 2.3. Let an instance I be given by

R	A	B
1	a	b
2	a'	b

Q	B	C
1	b	c
2	d	e

Then I is the functor

$$\begin{tikzcd}
I(A) = \{a, a'\} & I(R) = \{1,2\} \arrow[l, "I(\delta^R_A)"'] \arrow[r, "I(\delta^R_B)"] & I(B) = \{b, d\} & I(Q) = \{1,2\} \arrow[l, "I(\delta^Q_B)"'] \arrow[r, "I(\delta^Q_C)"] & I(C) = \{c, e\}
\end{tikzcd}$$

with $I(\delta^R_A)(1) = a$, $I(\delta^R_B)(1) = b$ and so on.

EXAMPLE 2.5
Let J be the instance $J : \Delta_2 \longrightarrow$ **FinSet** given by $J(\{0,1,2\}) = \{\langle a,b,c\rangle\}$, $J(\{0,1\}) = \{\langle a,b\rangle, \langle a',b\rangle\}$, $J(\{1,2\}) = \{\langle b,c\rangle, \langle d,e\rangle\}$, $J(\{0,2\}) = \{\langle a,c\rangle, \langle a',c\rangle\}$, $J(0) = \{a,a'\}$, $J(1) = \{b,d\}$, $J(2) = \{c,e\}$ and functions $J(\delta^-_-)$ the expected projections. Writing this up in table form, we obtain:

	0	1	2
	a	b	c

	0	1
	a	b
	a'	b

	0	2
	a	c
	a'	c

	1	2
	b	c
	d	e

0
a
a'

1
b
d

2
c
e

Strict relational instances and strictification. Say that a relational instance is *tuplified* or *strict* if the keys are tuples and the δ's are (mapped to) projections. Accordingly, in a strict instance I on schema X, we have that $I(x) \subseteq \prod_{A \in B_{\leq x}} I(A)$. (We reserve the product symbol $\prod$ for the dependent product in the category **Set** of sets. The dependent product in the type theory will be denoted by Π.) Thus Example 2.5 is strict. It is clear that a relational instance is naturally isomorphic to exactly one strict relational instance with the same values. We say that the latter is the *tuplification* or *strictification* of the former.

Working with relational instances up to strictification, or restricting to the strict ones, resolves the coherence issues so typical of categorical models of type theory. To have the 'strict' instances be those 'on tuple form' presents itself as a natural choice, not least because of the connection to the relational model. Thus, formally we shall consider ourselves to work in the category of strict relational instances. We proceed informally, however, by allowing arbitrary relational instances but not distinguishing between relational instances that are equal up to tuplification.

Substitution. Let $f : X \longrightarrow Y$ be a morphism of schemas, and let $I : Y \longrightarrow$ **FinSet** be a relational instance. Then it is easily seen that the composite $I \circ f : X \longrightarrow$ **FinSet** is a relational instance. We write $I[f] := I \circ f$ and say it is the *substitution of I along f*.

Substitution does not preserve strict instances. If $f : X \longrightarrow Y$ is display, however, and $I : Y \longrightarrow$ **FinSet** is strict, then we have that $I \circ f(x) \subseteq \prod_{A' \in B_{\leq f(x)}} \cong \prod_{A \in B_{\leq x}}$. The strictification of $I[f]$ is then just the reindexing of the tuples along the bijection $f_0 : B_{\leq x} \longrightarrow B_{\leq f(x)}$ (for all faces x). In contrast, if f is not display then there must exist a face, say $\{A, B\}$, such that $f(A) = f(B) = f(x)$, and then $f \circ I(x)$ need not be a set of tuples at all. (Accordingly, if we took an ordered perspective on schemas and morphisms and defined strict instances in terms of cartesian products, then display morphisms would be exactly the morphisms that preserve strict instances on the nose.) We display this for emphasis.

LEMMA 2.6
Let $f : X \longrightarrow Y$ be a morphism of schemas. Then f is display if and only if substitution along f preserves instances on tuple form (up to reindexing).

EXAMPLE 2.7
Consider the morphism $f : S \longrightarrow \Delta_2$ of Example 2.3 and the instances I and J of Example 2.4. Then $J[f]$ is the strictification of I, modulo the reindexing given by f_0.

The schema induced by an instance The connection between display maps, relational instances and simplicial schemas is given by the following. Let X be a schema and $F : X \longrightarrow$ **FinSet** an arbitrary functor. Recall, e.g. from (9), that the category of elements $\int_X F$ has objects $\langle x, a\rangle$ with $x \in X$ and $a \in F(x)$. A morphism $\delta^{\langle x,a\rangle}_{\langle y,b\rangle} : \langle x, a\rangle \longrightarrow \langle y, b\rangle$ is a morphism $\delta^x_y : x \longrightarrow y$ with $F(\delta^x_y)(a) = b$. The projection $p : \int_X F \longrightarrow X$ is defined by $\langle x, a\rangle \mapsto x$ and $\delta^{\langle x,a\rangle}_{\langle y,b\rangle} \mapsto \delta^x_y$. We then have

LEMMA 2.8
Let X be a simplicial schema and $F : X \longrightarrow$ **FinSet** be a functor. Then F is a relational instance if and only if $\int_X F$ is a simplicial schema and $p : \int_X F \to X$ is a display morphism.

PROOF. Let F be a relational instance. It is clear that there is at most one morphism between any two objects in $\int_X F$, and it is easy to see that $(\int_X F)^{\text{op}}$ is a based poset with base $\{\langle A,a\rangle \mid A \in X_0,\ a \in F(A)\}$, satisfying the condition for being a simplicial complex. Furthermore, $(\int_X F)_n^{\text{op}} = \{\langle x,c\rangle \mid x \in X_n,\ c \in F(x)\}$ so the projection $p : \int_X F \to X$ is a display morphism.

Conversely, if F is not relational, then the first condition for being a based poset is violated by any two keys with the same data. □

When F is a relational instance, we write $X.F$ for $\int_X F$ and refer to it as the *canonical schema* of F. We refer to p as the *canonical projection*.

EXAMPLE 2.9
The instance J of Example 2.4 has the canonical schema given by the attribute set $\{\langle 0,a\rangle, \langle 0,a'\rangle, \langle 1,b\rangle, \langle 1,d\rangle, \langle 2,e\rangle, \langle 2,c\rangle\}$ and relation variables, e.g. $\langle\{0,1,2\},\langle a,b,c\rangle\rangle$.

Terminal instances. A schema X induces a canonical instance of itself by filling out the relations by a single row each, consisting of the attributes of the relation. This instance is terminal in the category of instances of X; that is, every other instance of X has a unique morphism to it. It is of course isomorphic to the strict instance defined by $A \mapsto 1$, for $A \in X_0$ and $1 = \{*\}$ a fixed singleton set and $x \mapsto\ ! : B_{\leq x} \longrightarrow 1$ for $x \in X \setminus X_0$. We take this (latter) instance to be the *terminal instance* $1_X : X \longrightarrow$ **FinSet**. For notational reasons, however, we allow ourself below to think of it and write it as the functor defined by $x \mapsto \{x\}$.

Full tuples and induced sections. A *full* or *matching tuple* t of an instance I over schema X is a natural transformation $t : 1_X \Rightarrow I$. We write $\text{Trm}_X(I)$ for the set of full tuples (indicating that we see them as terms type-theoretically). A full tuple t of a strict instance can be considered as an element in $\prod_{A\in X_0} I(A)$ satisfying the condition that for all $x \in X$, we have that $t\restriction_{B_{\leq x}} \in I(x)$.

Given a full tuple $t : 1_X \Rightarrow I$, the *induced section* is the morphism $\hat{t} : X \longrightarrow X.I$ defined by $x \mapsto \langle x, t_x(x)\rangle$. Notice that the induced section is always a display morphism.

EXAMPLE 2.10
The instance I of Example 2.4 has precisely two full tuples, one of which is given by $R \mapsto 1$, $Q \mapsto 1$, $A \mapsto a$, $B \mapsto b$ and $C \mapsto c$. A full tuple can be seen as a tuple over the full attribute set of the schema with the property that for all relation variables, the projection of the tuple is a row of that relation. The two full tuples of I are, then, $\langle a,b,c\rangle$ and $\langle a',b,c\rangle$. The instance J of Example 2.4 has precisely one full tuple $\langle a,b,c\rangle$.

2.2 *Structure of the model*

We have a functor $\text{Rel}(-) : \mathbb{S}_d{}^{\text{op}} \longrightarrow \text{Cat}$ that maps X to $\text{Rel}(X)$ and $f : X \longrightarrow Y$ to $\text{Rel}(f) = (-)[f] : \text{Rel}(Y) \longrightarrow \text{Rel}(X)$. We denote this indexed category by $\mathfrak{R}$ and think of it as a 'category of databases' in which the totality of databases and schemas are collected. It is a model of a certain dependent type theory with context constants that we give in Section 3. We briefly outline some of the relevant structure available in $\mathfrak{R}$.

DEFINITION 2.11
For $f : X \longrightarrow Y$ in $\mathbb{S}_d$ and $J \in \text{Rel}(Y)$ and $t : 1_Y \Rightarrow J$ in $\text{Trm}_Y(J)$:

1. Define $t[f] \in \text{Trm}_X(J[f])$ by $x \mapsto t(f(x)) \in J[f](x)$. Note that $t[f][g] = t[f \circ g]$.
2. With $p_J : Y.J \longrightarrow Y$ the canonical projection, let $v_J : 1_{Y.J} \Rightarrow J[p_J]$ be the full tuple defined by $\langle y,a\rangle \mapsto a$. (We elsewhere leave subscripts on v and p determined by context.)

3. Denote by $\tilde{f} : X.J[f] \longrightarrow Y.J$, the schema morphism defined by $\langle x, a\rangle \mapsto \langle f(x), a\rangle$. Notice that since f is display, so is $\tilde{f}$.

LEMMA 2.12
The following equations hold:

1. For X in $\mathbb{S}_d$ and $I \in \mathrm{Rel}(X)$ and $t \in \mathrm{Trm}_X(I)$, we have $p \circ \hat{t} = \mathrm{id}_X$ and $t = v[\hat{t}]$.
2. For $f : X \longrightarrow Y$ in $\mathbb{S}_d$ and $J \in \mathrm{Rel}(Y)$ and $t \in \mathrm{Trm}_Y(J)$, we have
   a. $p \circ \tilde{f} = f \circ p : X.J[f] \longrightarrow Y$;
   b. $\tilde{f} \circ \widehat{t[f]} = \hat{t} \circ f : X \longrightarrow Y.J$; and
   c. $v_J[\tilde{f}] = v_{J[f]} : 1_{X.J[f]} \Rightarrow J[f][p]$.
3. For $f : X \longrightarrow Y$ and $g : Y \longrightarrow Z$ in $\mathbb{S}_d$ and $J \in \mathrm{Rel}(Z)$, we have $\widetilde{g \circ f} = \tilde{g} \circ \tilde{f}$.
4. For $X \in \mathbb{S}_d$ and $I \in \mathrm{Rel}(X)$, we have $\tilde{p} \circ \hat{v} = \mathrm{Id}_{X.I}$.

PROOF. 1) $v[\hat{t}]$ is the full tuple of $J[p][\hat{t}] = J[p \circ \hat{t}] = J[\mathrm{id}_X]$ defined by $x \mapsto v(\hat{t}(x)) = v(\langle x, t(x)\rangle) = t(x)$. 2.a) We have $p \circ \tilde{f}(x, a) = p(f(x), a) = f(x) = f \circ p(x, a)$. 2.b) We have $\tilde{f} \circ \widehat{t[f]}(x) = \tilde{f}(x, t(fx)) = \langle f(x), t(fx)\rangle = \hat{t}(fx) = \widetilde{\hat{t} \circ f}(x)$. 2.c) $v_J[\tilde{f}](x, a) = v_J(fx, a) = a = v_{J[f]}(x, a)$. 3) We have $\tilde{g}(\tilde{f}(x, a)) = \tilde{g}(fx, a) = (gfx, a) = \widetilde{g \circ f}(x, a)$. 4) We have $\tilde{p} \circ \hat{v}(x, a) = \tilde{p}(\langle x, a\rangle, a) = \langle x, a\rangle$. □

Finally, we present the instance-forming operations of dependent product, dependent sum, 0 and 1, identity and disjoint product.

Dependent product. Let $X \in \mathbb{S}$, $I \in \mathrm{Rel}(X)$ and $J \in \mathrm{Rel}(X.J)$. We define the instance $\Pi_J I : X \longrightarrow$ **FinSet** as the right Kan-extension of J along p. Explicitly, we define the following strict instance (assuming also I and J strict).

For A in X_0 define

$$\Pi_J I(A) = \prod_{a \in I(A)} J(A, a)$$

Let $x \in X, f \in \prod_{A \in B_{\leq x}} \prod_{a \in I(A)} J(A, a)$, $y \leq x$ and $s \in I(y)$. Define

$$\hat{f}_{y,s} \in \prod_{\{\langle A,a\rangle \;\mid\; A \in B_{\leq y}, a = s(A)\}} J(A, a)$$

by $\langle A, a\rangle \mapsto f(A)(a))$. For $x \in X$ define

$$\Pi_I J(x) = \left\{ f \in \prod_{A \in B_{\leq x}} \prod_{a \in I(A)} J(A, a) \;\middle|\; \forall y \leq x.\, \forall s \in I(y).\, \hat{f}_{y,s} \in J(y, s) \right\}.$$

Next, let $f \in \mathrm{Trm}_X(\Pi_I J)$ be a full tuple of the dependent product. We consider f as an element in $\prod_{A \in X_0} \Pi_I J(A)$ satisfying the condition that for all $x \in X$, we have that $f \restriction_{B_{\leq x}} \in \Pi_I J(x)$. Consider the element $\mathrm{Ap}_f \in \prod_{\langle A,a\rangle \in (X.I)_0} J(A, a)$ given by $\langle A, a\rangle \mapsto f(A)(a)$. Then for any $\langle y, s\rangle \in X.I$, we have $\mathrm{Ap}_f \restriction_{B_{\leq\langle y,s\rangle}} \in J(y, s)$ by the definition of $\Pi_J I(x)$, so $\mathrm{Ap}_f \in \mathrm{Trm}_{X.I}(J)$.

Finally, let $t \in \mathrm{Trm}_{X.I}(J)$, considered as an element of $\prod_{\langle A,a\rangle \in (X.I)_0} J(A, a)$. Define the element λt of $\prod_{A \in X_0} \Pi_I J(A)$ by $\lambda t(A)(a) = t(A, a)$. Then for all $x \in X$, we have that $\lambda t \restriction_{B_{\leq x}} \in \Pi_I J(x)$ since t is a full tuple.

LEMMA 2.13
Let $f : X \longrightarrow Y$ be a display morphism in $\mathbb{S}$, $I \in \mathrm{Rel}(Y)$, $J \in \mathrm{Rel}(Y.I)$, $g \in \mathrm{Trm}_Y(\Pi_I J)$ and $t \in \mathrm{Trm}_{(Y.I)}(J)$.

1. $\mathrm{Ap}_{\lambda t} = t$ and $\lambda\,\mathrm{Ap}_g = g$.
2. $(\Pi_I J)[f] = \Pi_{I[f]} J[\tilde{f}]$.
3. $(\lambda t)[f] = \lambda(t[\tilde{f}])$.
4. $\mathrm{Ap}_g[f] = \mathrm{Ap}_{g[f]}$.

PROOF. Tedious but straightforward. □

EXAMPLE 2.14
Consider the schema S and instance I of Examples 2.3 and 2.4. Corresponding to the display map $f : S \longrightarrow \Delta_2$, we can present S an instance of Δ_2 as (ignoring strictification for readability) $S : \Delta_2 \longrightarrow \mathbf{FinSet}$ by $S(0) = \{A\}$, $S(1) = \{B\}$, $S(2) = \{C\}$, $S(01) = \{R\}$, $S(12) = \{Q\}$ and $S(02) = S(012) = \emptyset$. Notice that, modulo the isomorphism between S as presented in Example 2.3 and $\Delta_2.S$, the morphism $f : S \longrightarrow \Delta_2$ is the canonical projection $p : \Delta_2.S \longrightarrow \Delta_2$. Similarly, we have $I \in \Delta_2.S$ as (in tabular form, using subscript instead of pairing for elements in $\Delta_2.S$, and omitting the three single-column tables)

R_{01}	A_0	B_1
	a	b
	a'	b

Q_{12}	B_1	C_2
	b	c
	d	e

Then $\Pi_S I$ is, in tabular form,

	0	1	2
	a	b	c
	a'	b	c

	0	1
	a	b
	a'	b

	0	2
	a	c
	a'	c
	a	e
	a'	e

	1	2
	b	c
	d	e

	0
	a
	a'

	1
	b
	d

	2
	c
	e

Notice that the three-column 'top' table of $\Pi_S I$ is the natural join $R_{01} \bowtie Q_{12}$. The type theory of the next section will syntactically derive the rows of this table from the syntactic specification of S and I and the rules for the dependent product.

We present the remaining instance-forming operations more briefly. In particular, we omit the statements and (straightforward) proofs that all defined instances and terms are stable under substitution.

0 and 1 instances. Given $X \in \mathbb{S}$ the terminal instance 1_X has already been defined. The *initial* instance 0_X is the constant 0 functor, $x \mapsto \emptyset$. Note that $X.0_X$ is the empty schema.

Dependent sum. Let $X \in \mathbb{S}$, $I \in \mathrm{Rel}(X)$ and $J \in \mathrm{Rel}(X.I)$. We define the instance $\Sigma_I J : X \longrightarrow \mathbf{FinSet}$ (up to tuplification) by $x \mapsto \{\langle a, b\rangle \mid a \in I(x),\ b \in J(x,a)\}$. For δ^x_y in X, $\Sigma_I J(\delta^x_y)(a,b) = \langle \delta^x_y(a), \delta^{x,a}_{y,\delta^x_y(a)}(b)\rangle$.

Identity. Given $X \in \mathbb{S}$ and $J \in \mathrm{Rel}(X)$ the *Identity instance* $\mathrm{Id}_J \in \mathrm{Rel}(X.J.J[p])$ is defined, up to tuplification, by $\langle\langle x, a\rangle, b\rangle \mapsto 1$ if $a = b$ and $\langle\langle x, a\rangle, b\rangle \mapsto \emptyset$ else. The full tuple $\mathrm{refl} \in \mathrm{Trm}_{(X.J)}(\mathrm{Id}_J[\hat{v}])$ is defined by $\langle x, a\rangle \mapsto *$.
Disjoint union. Given $X \in \mathbb{S}$ and $I, J \in \mathrm{Rel}(X)$, the instance $I + J \in \mathrm{Rel}(X)$ is defined by $x \mapsto \{\langle n, a\rangle \mid \text{Either } n = 0 \wedge a \in I(x) \text{ or } n = 1 \wedge a \in J(x)\}$. We have full tuples left $\in \mathrm{Trm}_{X.I}((I+J)[p])$ defined by $\langle x, a\rangle \mapsto \langle 0, a\rangle$ and right $\in \mathrm{Trm}_{X.J}((I+J)[p])$ defined by $\langle x, a\rangle \mapsto \langle 1, a\rangle$.

3 The type theory

We introduce a Martin-Löf style type theory (10), with explicit substitutions (in the style of (4)), extended with context and substitution constants representing simplices and face maps. The type theory contains familiar constructs such as Σ- and Π-types. It contains a universe that is closed under the other constructions, which we will describe in more detail in the next section. For this type theory, we give an interpretation in the indexed category $\mathfrak{R}$ of the previous section. The goal is to use the type theory as a formal language for databases. We give examples how to specify instances and schemas formally in the theory and remark on how to use the universe to talk about queries.

3.1 *The type theory $\mathcal{T}$*

The type system has the following eight judgements, with intended interpretations.

Judgement	Interpretation
$- : \texttt{Context}$	$[\![-]\!]$ is a schema
$- : \texttt{Type}(\Gamma)$	$[\![-]\!]$ is an instance of the schema Γ
$- : \texttt{Elem}(A)$	$[\![-]\!]$ is an full tuple in the instance A
$- : \Gamma \longrightarrow \Lambda$	$[\![-]\!]$ is a (display) schema morphism
$\Gamma \equiv \Lambda$	$[\![\Gamma]\!]$ and $[\![\Lambda]\!]$ are equal schemas
$A \equiv B : \texttt{Type}(\Gamma)$	$[\![A]\!]$ and $[\![B]\!]$ are equal instances of $[\![\Gamma]\!]$
$t \equiv u : \texttt{Elem}(A)$	$[\![t]\!]$ and $[\![u]\!]$ are equal full tuples in $[\![A]\!]$
$\sigma \equiv \tau : \Gamma \longrightarrow \Lambda$	the morphisms $[\![\sigma]\!]$ and $[\![\tau]\!]$ are equal

The type theory $\mathcal{T}$ has the rules listed in Figure 1. The interpretation of these are given by the constructions in the previous section and summarized in Figure 2.

Each rule introduces a context, substitution, type or element. We will apply usual abbreviations such as $A \longrightarrow B$ for $\Pi_A B[\downarrow_A]$ and $A \times B$ for $\Sigma_A B[\downarrow_A]$. In addition to these term introducing rules, there are a number of equalities which should hold, such as the simplicial identities $d_i^{n+1} \circ d_j^n \equiv d_{j-1}^{n+1} \circ d_i^n : \Delta_n \longrightarrow \Delta_{n+2}$. We list the definitional equalities in Figure 3.

These all hold in our model. (The equalities for substitution are verified in Lemma 3. The remaining equations are mostly routine verifications.) We display this for reference.

THEOREM 3.1
The intended interpretation $[\![-]\!]$ yields a sound interpretation of the type theory $\mathcal{T}$ in $\mathfrak{R}$.

3.2 *Instance specification as type introduction*

The intended interpretation of $\Gamma \vdash A$ $\texttt{type}$ is that A is an instance of the schema Γ. But context extension allows us to view every instance as a schema in its own right; for every instance $\Gamma \vdash$

$\sigma : \Gamma \longrightarrow \Delta,\ \tau : \Delta \longrightarrow \Theta$	$\vdash \tau \circ \sigma : \Gamma \longrightarrow \Theta$
$\Gamma : \texttt{Context}$	$\vdash \mathrm{id}_\Gamma : \Gamma \longrightarrow \Gamma$
$A : \texttt{Type}(\Gamma),\ \sigma : \Delta \longrightarrow \Gamma$	$\vdash A[\sigma] : \texttt{Type}(\Gamma)$
$a : \texttt{Elem}(A),\ \sigma : \Delta \longrightarrow \Gamma$	$\vdash a[\sigma] : \texttt{Elem}(A[\sigma])$
$A : \texttt{Type}(\Gamma)$	$\vdash \Gamma.A : \texttt{Context}$
$A : \texttt{Type}(\Gamma)$	$\vdash \downarrow_A : \Gamma.A \longrightarrow \Gamma$
$A : \texttt{Type}(\Gamma)$	$\vdash v : \texttt{Elem}(A[\downarrow_A])$
$a : \texttt{Elem}(A)$	$\vdash a\uparrow : \Gamma \longrightarrow \Gamma.A$
$A : \texttt{Type}(\Gamma),\ \sigma : \Delta \longrightarrow \Gamma$	$\vdash \sigma.A : \Delta.A[\sigma] \longrightarrow \Gamma.A$
$A : \texttt{Type}(\Gamma),\ B : \texttt{Type}(\Gamma.A)$	$\vdash \Pi_A B : \texttt{Type}(\Gamma)$
$b : \texttt{Elem}(B)$	$\vdash \lambda b : \texttt{Elem}(\Pi_A B)$
$f : \texttt{Elem}(\Pi_A B)$	$\vdash \mathrm{apply}_f : \texttt{Elem}(B)$
$A : \texttt{Type}(\Gamma),\ B : \texttt{Type}(\Gamma.A)$	$\vdash \Sigma_A B : \texttt{Type}(\Gamma)$
$A : \texttt{Type}(\Gamma),\ B : \texttt{Type}(\Gamma.A)$	$\vdash \mathrm{pair} : \texttt{Elem}(\Sigma_A B[\downarrow_A][\downarrow_B])$
$C : \texttt{Type}(\Gamma.\Sigma_A B),$	
$c_0 : \texttt{Elem}(C[(\downarrow_A \circ \downarrow_B).\Sigma_A B][\mathrm{pair}\uparrow]$	$\vdash \mathrm{rec}_\Sigma\ c_0 : \texttt{Elem}(C)$
$A : \texttt{Type}(\Gamma)$	$\vdash \mathrm{Id}_A : \texttt{Type}(\Gamma.A.A[\downarrow_A])$
$A : \texttt{Type}(\Gamma)$	$\vdash \mathrm{refl} : \texttt{Elem}(\mathrm{Id}_A[v\uparrow])$
$C : \texttt{Type}(\Gamma.A.A[\downarrow_A].\mathrm{Id}_A),$	
$c_0 : \texttt{Elem}(C[(v\uparrow).\mathrm{Id}_A][\mathrm{refl}\uparrow])$	$\vdash \mathrm{rec}_{\mathrm{Id}}\ c_0 : \texttt{Elem}(C)$
$A : \texttt{Type}(\Gamma)\ B : \texttt{Type}(\Gamma)$	$\vdash A + B : \texttt{Type}(\Gamma)$
$A : \texttt{Type}(\Gamma)\ B : \texttt{Type}(\Gamma)$	$\vdash l_{A,B} : \texttt{Elem}((A+B)[\downarrow_A])$
$A : \texttt{Type}(\Gamma)\ B : \texttt{Type}(\Gamma)$	$\vdash r_{A,B} : \texttt{Elem}((A+B)[\downarrow_B])$
$C : \texttt{Type}(\Gamma.(A+B)),$	
$c_0 : \texttt{Elem}(C[(\downarrow).(A+B)][l_{A,B}])$	
$c_1 : \texttt{Elem}(C[(\downarrow).(A+B)][r_{A,B}])$	$\vdash \mathrm{rec}_+\ c_0\, c_1 : \texttt{Type}(C)$
$\Gamma : \texttt{Context}$	$\vdash 0 : \texttt{Type}(\Gamma)$
$A : \texttt{Type}(\Gamma.0)$	$\vdash \mathrm{rec}_0 : \texttt{Elem}(A)$
$\Gamma : \texttt{Context}$	$\vdash 1 : \texttt{Type}(\Gamma)$
$\Gamma : \texttt{Context}$	$\vdash * : \texttt{Elem}(1)$
$A : \texttt{Type}(\Gamma.1), a : A[*\uparrow]$	$\vdash \mathrm{rec}_1\, a : \texttt{Elem}(A)$
	$\vdash \Delta_n : \texttt{Context}$
	$\vdash d_i^n : \Delta_n \longrightarrow \Delta_{n+1}$

where $n, i, j \in \mathbb{N}$ are such that $i < j \leq n + 2$.

FIGURE 1 Rules of the type theory.

A `type`, we get a schema *Γ.A*. It turns out that the most convenient way to specify a schema is by introducing a new type/instance over one of the simplex schemas Δ_n. To specify a schema, with a maximum of *n* attributes, may be seen as introducing a type in the context Δ_n. A relation variable

$$\begin{array}{cccc}
[\![\sigma \circ \tau]\!] = [\![\sigma]\!] \circ [\![\tau]\!] & [\![A[\sigma]]\!] = [\![A]\!]\,[[\![\sigma]\!]] & [\![id_\Gamma]\!] = id_{[\![\Gamma]\!]} & [\![a[\sigma]]\!] = [\![a]\!]\,[[\![\sigma]\!]] \\
[\![\Gamma.A]\!] = [\![\Gamma]\!].\,[\![A]\!] & [\![\downarrow_A]\!] = p & [\![v]\!] = v & [\![a\uparrow]\!] = \widehat{[\![a]\!]} \\
[\![\sigma.A]\!] = \widetilde{[\![\sigma]\!]} & [\![\Pi_A B]\!] = \Pi_{[\![A]\!]}\,[\![B]\!] & [\![\lambda f]\!] = \lambda\,[\![f]\!] & [\![\text{apply}]\!] = Ap \\
[\![\Sigma_A B]\!] = \Sigma_{[\![A]\!]}\,[\![B]\!] & [\![\text{pair}]\!] = \text{pair} & [\![\text{rec}_\Sigma]\!] = \text{rec}_\Sigma & [\![\text{Id}_A]\!] = \text{Id}_{[\![A]\!]} \\
[\![\text{refl}]\!] = \text{refl} & [\![\text{rec}_{\text{Id}}]\!] = \text{rec}_{\text{Id}} & [\![\Delta_n]\!] = \Delta_n & [\![d_i^n]\!] = d_i^n
\end{array}$$

FIGURE 2 Interpretation of the type theory

$$\begin{array}{ccc}
(v \circ \tau) \circ \sigma \equiv v \circ (\tau \circ \sigma) & id_\Gamma \circ \sigma \equiv \sigma & \sigma \circ id_\Gamma \equiv \sigma \\
a[\sigma \circ \tau] \equiv a[\sigma][\tau] & a[id_\Gamma] \equiv a & \downarrow_A \circ (a\uparrow) \equiv id_\Gamma \\
v[a\uparrow] \equiv a & \downarrow_A \circ (\sigma.A) \equiv \sigma \circ \downarrow_{A[\sigma]} & v[f.A] \equiv v \\
(\sigma.A) \circ (a[\sigma]\uparrow) \equiv a\uparrow \circ \sigma & (\sigma.A) \circ (\tau.A[\sigma]) \equiv (\sigma \circ \tau).A & \downarrow .A \circ v\uparrow \equiv Id_{\Gamma.A} \\
\Pi_A B[\sigma] \equiv \Pi_{A[\Sigma]} B[\sigma.A] & \text{apply}(\lambda b) \equiv b & \\
d_i^{n+1} \circ d_j^n \equiv d_{j-1}^{n+1} \circ d_i^n & &
\end{array}$$

FIGURE 3 Definitional equalities in the type theory.

with *k* attributes in the schema is introduced as an element of the schema substituted into Δ_k. Names of attributes are given as elements of the schema substituted down to Δ_0.

EXAMPLE 3.2
We construct the rules of the schema *S* presented as an instance of Δ_2 as in Example 2.14. The introduction rules tells us the names of tables and attributes in S.

$$\begin{array}{ll}
S : \texttt{Type}(\Delta_2) & A \equiv R[d_1] : \texttt{Elem}(S[d_2 \circ d_1]) \\
A : \texttt{Elem}(S[d_2 \circ d_1]) & B \equiv R[d_0] : \texttt{Elem}(S[d_2 \circ d_0]) \\
B : \texttt{Elem}(S[d_2 \circ d_0]) & B \equiv Q[d_1] : \texttt{Elem}(S[d_0 \circ d_1]) \\
C : \texttt{Elem}(S[d_0 \circ d_0]) & C \equiv Q[d_0] : \texttt{Elem}(S[d_2 \circ d_0]) \\
R : \texttt{Elem}(S[d_2]) & \\
Q : \texttt{Elem}(S[d_0]) &
\end{array}$$

From these introduction rules, we can generate an elimination rule. The elimination rule tells us how to construct full tuples in an instance over the schema S. Another interpretation of the elimination rule is that it formulates that the schema S contains only what is specified by the above introduction rules, it specifies the schema up to isomorphism.

$$\begin{aligned}&I : \mathtt{Type}(\Delta_2.S), a : \mathtt{Elem}(\Delta_0, I[(d_2 \circ d_1).S][A\uparrow]),\\ &b : \mathtt{Elem}(\Delta_0, I[(d_2 \circ d_0).S][B\uparrow]), c : \mathtt{Elem}(\Delta_0, I[(d_0 \circ d_0).S][C\uparrow]),\\ &r : \mathtt{Elem}(\Delta_1, I[d_2.S][R\uparrow]), q : \mathtt{Elem}(\Delta_1, I[d_0.S][Q\uparrow]),\\ &r[d_1] \equiv a, r[d_0] \equiv b, q[d_1] \equiv b, q[d_0] \equiv c\\ &\vdash \mathrm{rec}_S\ a\,b\,c\,r\,q : \mathtt{Elem}(\Delta_2.S, I).\end{aligned}$$

An instance of a schema is a type depending in the context of the schema. Therefore, instance specification is analogous to schema specification.

EXAMPLE 3.3
Let S be the schema from the previous example. The following set of introductions presents an instance I of S.

$$\begin{array}{ll}\Delta_2.S \vdash I\mathtt{type} & \\ \vdash a : \mathtt{Elem}(\Delta_0, I[(d2 \circ d_1).S][A\uparrow]) & \vdash r_0[d_1] \equiv a\\ \vdash a' : \mathtt{Elem}(\Delta_0, I[(d2 \circ d_1).S][A\uparrow]) & \vdash r_0[d_0] \equiv b\\ \vdash b : \mathtt{Elem}(\Delta_0, I[(d2 \circ d_0).S][B\uparrow]) & \vdash r_1[d_1] \equiv a'\\ \vdash d : \mathtt{Elem}(\Delta_0, I[(d2 \circ d_0).S][B\uparrow]) & \vdash r_1[d_0] \equiv b\\ \vdash c : \mathtt{Elem}(\Delta_0, I[(d_0 \circ d_0).S][C\uparrow]) & \vdash q_0[d_1] \equiv b\\ \vdash e : \mathtt{Elem}(\Delta_0, I[(d_0 \circ d_0).S][C\uparrow]) & \vdash q_0[d_0] \equiv c\\ \vdash r_0 : \mathtt{Elem}(\Delta_1, I[(d2).S][R\uparrow]) & \vdash q_1[d_1] \equiv d\\ \vdash r_1 : \mathtt{Elem}(\Delta_1, I[(d2).S][R\uparrow]) & \vdash q_1[d_0] \equiv e\\ \vdash q_0 : \mathtt{Elem}(\Delta_1, I[(d_0).S][Q\uparrow]) & \\ \vdash q_1 : \mathtt{Elem}(\Delta_1, I[(d_0).S][Q\uparrow]) & \end{array}$$

The above is clearly very verbose, and can be compressed, at the cost of loosing control over the naming of attributes, into the following.

$$\begin{array}{ll}\vdash I : \mathtt{Type}(\Delta_2.S) & \\ \vdash r_0 : \mathtt{Elem}(\Delta_1, I[(d2).S][R\uparrow]) & \vdash r_0[d_0] \equiv r_1[d_0]\\ \vdash r_1 : \mathtt{Elem}(\Delta_1, I[(d2).S][R\uparrow]) & \vdash q_0[d_1] \equiv r_0[d_0]\\ \vdash q_0 : \mathtt{Elem}(\Delta_1, I[(d_0).S][Q\uparrow]) & \\ \vdash q_1 : \mathtt{Elem}(\Delta_1, I[(d_0).S][Q\uparrow]) & \end{array}$$

We omit the elimination rule.

4 Universe

We construct the universe of finite instances of a schema. This is a large instance (in the sense of the small/large distinction of Section 2). Thus we allow in this section that schemas and instances can be large, i.e. have infinitely many attributes, tables and rows.

4.1 Constructing the universe

An essential part of type theory is the notion of a universe of types. A universe of types is a family of types that is closed under some set of type constructors. This allows powerful reasoning about types in type theory itself. In this section we will, for each schema X, construct a universe of finite instances of X.

More precisely, a universe in type theory, in a context Γ, consists of a type $U : \mathtt{type}(\Gamma)$ and a family $T : \mathtt{type}(\Gamma.U)$. We think of the type U as the type of codes for types in the universe and the family T as decoding the codes into actual types (by substitution).

$$\begin{tikzcd}
X.U_X \arrow[d, "p"] \arrow[drr, "T_X"] & & \\
X \arrow[u, bend left, "\widehat{t}"] \arrow[rr, "T_X[\widehat{t}]"'] & & \mathbf{FinSet}
\end{tikzcd}$$

An important feature of a type-theoretic universe is its closure under typeforming operations such as Π- and Σ-types. This is what allows reasoning about types internally in the type theory.

In order to encode the collection of finite instances into an instance of its own, we need a small set of tuples to work with. Let us therefore fix a set of values V closed under making lists, in the sense that $V^i \subseteq V$ for every finite subset $i \subseteq V$. This allows us to iterate tuple forming constructions such as Π and Σ.

DEFINITION 4.4
Define $D_n(V)$ to be the set of strict instances of the schema Δ_n that have values in V.

DEFINITION 4.5
Given a schema X, we define $U_X : X \to \mathit{Set}$ by

$$U_X(A) = \{\langle n, k, I\rangle \mid n \in \mathbb{N} \wedge k \in [n] \wedge I \in D_n(V)\} \text{ for attributes} A$$

$$U_X(x) = \left\{ t : \prod_{A \in B_{\leq x}} U_X(A) \mid \pi_1 \circ t \text{ injective} \wedge \pi_2 \circ t \text{ constant}\} \wedge \pi_2 \circ t \text{ constant} \right\}$$

for faces x.

In order to define the decoding instance of the universe, which takes a code to its instance, we need easy access to the instances on attribute level.

DEFINITION 4.6
Given a schema X, a face x in X and an element $t \in U_X(x)$, we denote by $d(x,t) : \mathbb{N}$ the unique number such that $\pi_0(t(A)) = d(x,t)$ for all attributes A of x. Let $I(x,t)$ be the uniquely defined instance such that $\pi_2(t(A)) = I(x,t)$ for all attributes A of x. We denote by $\alpha(x,t) : \downarrow x \to \Delta_{d(x,t)}$, the morphism defined by $\alpha(x,t)(A) = \pi_1(tA)$ on attributes.

Lemma 4.7
Given $\sigma : X \to Y$ then for all $x \in X$ and $t \in U_X(x)$, we have that

1. $d(\sigma(x), t \circ \sigma) = d(x,t)$
2. $I(\sigma(x), t \circ \sigma) = I(x,t)$
3. $\alpha(\sigma(x), t \circ \sigma) = \alpha(x,t) \circ \sigma$.

PROOF. 1. and 2. follows since $U_X(A) = U_X(\sigma(A))$ for all attributes. 3. is evident since $\alpha(\sigma(x), t \circ \sigma)(A) = \pi_1((t \circ \sigma)A) = \pi_1(t(\sigma(A)) = \alpha(x,t)(\sigma(A))$, by definition for all A □

DEFINITION 4.8
Given a schema X, define $T_X : X.U_X \to \mathit{Set}$ by,

$$T_X(x,t) = I(x,t) \circ \alpha(x,t).$$

PROPOSITION 4.9
The constructions U_- and T_- are invariant under substitution.

PROOF. U_- is invariant by construction; it is a strict instance that is constant on attributes.
T_- is invariant by the calculation, given $\sigma : X \to Y$

$$\begin{aligned} T_Y(\sigma(x), t\circ\sigma) &= I(\sigma(x), t\circ\sigma) \circ \alpha(\sigma(x), t\circ\sigma) \\ &= I(x,t) \circ \alpha(x,t) \circ \sigma \\ &= T_X(x,t) \circ \sigma. \end{aligned}$$

□

PROPOSITION 4.10
Given a schema X, a full tuple $a : 1 \Rightarrow U_X$ and a full tuple $b : 1 \Rightarrow U_{X.T_X[\hat{a}]}$, there is a full tuple $\sigma : 1 \Rightarrow U_X$ such that $T_X[\hat{\sigma}] = \Sigma_{T_X[\hat{a}]}(T_{X.T_X[\hat{a}]}[\hat{b}])$.

PROOF. Let $Q = \Sigma_{T_X[\hat{a}]}(T_{X.T_X[\hat{a}]}[\hat{b}])$
For any face x of X, define an instance $S_x : \Delta_{d(x,a_x(*))} \to \mathit{Set}$

$$S_x(k) = \begin{cases} Q(x') & \text{where } x' \text{ is such that } \exists x''\ x \le x'' \wedge x' \le x'' \wedge \alpha(x', a_{x'}(*))(x') = k \\ \emptyset & \text{if no such } x' \text{ exists.} \end{cases}$$

Let $\sigma_x(*)(A) = \langle d(x, a_x(*)), \alpha(x, a_x(*))(A), S_x\rangle$. This defines a term in U_X, since $\Sigma_{T_X[\hat{a}]}(T_{X[\hat{a}]}[\hat{b}])$ has values in V, and the definition of S_x is constant upwards and downwards in X.
Furthermore: $T_X[\hat{\sigma}] = \Sigma_{T_X[\hat{a}]}(T_{X[\hat{a}]}[\hat{b}])$, by construction. □

PROPOSITION 4.11
Given a schema X, a full tuple $a : 1 \Rightarrow U_X$ and a full tuple $b : 1 \Rightarrow U_{X.T_X[\hat{a}]}$, there is a full tuple $\pi : 1 \Rightarrow U_X$ such that $T_X[\hat{\pi}] = \Pi_{T_X[\hat{a}]}(T_{X.T_X[\hat{a}]}[\hat{b}])$.

PROOF. Let $Q = \Pi_{T_X[\hat{a}]}(T_{X.T_X[\hat{a}]}[\hat{b}])$
For any face x of X, define an instance $S_x : \Delta_{d(x,a_x(*))} \to \mathit{Set}$

$$S_x(k) = \begin{cases} Q(x') & \text{where } x' \text{ is such that } \exists x''\ x \le x'' \wedge x' \le x'' \wedge \alpha(x', a_{x'}(*))(x') = k \\ \emptyset & \text{if no such } x' \text{ exists.} \end{cases}$$

Let $\pi_x(*)(A) = \langle d(x, a_x(*)), \alpha(x, a_x(*))(A), S_x\rangle$. This defines a term in U_X, since $\Pi_{T_X[\hat{a}]}(T_{X[\hat{a}]}[\hat{b}])$ has values in V, and the definition of S_x is constant upwards and downwards in X.
Furthermore: $T_X[\hat{\pi}] = \Pi_{T_X[\hat{a}]}(T_{X[\hat{a}]}[\hat{b}])$, by construction. □

4.2 Using the universe in the type theory

The rules for the universe in the type theory itself can be summarized as

$$\Gamma : \texttt{Context} \vdash U : \texttt{Type}(\Gamma)$$

$$\Gamma : \texttt{Context} \vdash T : \texttt{Type}(\Gamma.U),$$

along with rules for invariance under substitution and closure rules for each type-formation rule we previously had. For instance, the closure rule for Π is the following:

$$\begin{array}{ll} \Gamma : \mathtt{Context} & \vdash \pi_\Gamma : \mathtt{Elem}(U[\downarrow_U][\downarrow_{T\longrightarrow U}]) \\ a : \mathtt{Elem}(U), & \\ f : \mathtt{Elem}(T[a\uparrow] \longrightarrow U) & \vdash \pi[a\uparrow.(T \longrightarrow U)][f\uparrow] \equiv \Pi_{T[a\uparrow]} T[\mathrm{apply}_f \uparrow] : \mathtt{Type}(\Gamma). \end{array}$$

The universe, $U_\Gamma : \mathtt{Type}(\Gamma)$ along with $T_\Gamma : \mathtt{Type}(\Gamma.U_\Gamma)$, allows reasoning generically about classes of instances of Γ in the type theory itself, without having to resort to the metalanguage. Since schemas can be though of as instances, they too can be constructed using the universe. In particular, given a schema Γ, the type $\Omega_\Gamma := \Sigma_{U_\Gamma} \Pi_{T_\Gamma} \Pi_{T_\Gamma[\downarrow_{T_\Gamma}]} \mathrm{Id}_{T_\Gamma}$ is the large type of subschemas of Γ. Its elements are decoded to instances by the family $\mathcal{O}_\Gamma := T[\downarrow_\Omega .U][\pi_0\uparrow]$. Given $t : \mathtt{Elem}(\Omega_\Gamma)$, the subschema it encodes is $\Gamma.\mathcal{O}[t\uparrow]$.

A query can then be seen as an operation that takes an instance of a schema to another instance of a related schema. Given codes for a source subschema $t : \mathtt{Elem}(\Omega)$ and a target subschema $u : \mathtt{Elem}(\Omega)$, the type of queries from t to u is thus $(\mathcal{O}[t\uparrow] \longrightarrow U) \longrightarrow (\mathcal{O}[u\uparrow] \longrightarrow U)$. Having given a concrete type of queries leads the way to investigations as to exactly which queries can be expressed in the language. For illustration, we present an example query formulated in this way.

EXAMPLE 4.12
In the spirit of Example 2.14, let $a : \mathtt{Elem}(\Omega)$ be the code for a subschema covering the schema Γ, in the sense that the set of attributes are the same. The query taking the dependent product, or natural join, of an instance of this subschema is expressed by the term

$$q = \lambda\lambda(\pi[(\downarrow_\Omega .U) \circ (\pi_0\uparrow)][a\uparrow.(T \longrightarrow U)][\downarrow_1]) : \mathtt{Elem}((\mathcal{O}[a\uparrow] \longrightarrow U) \longrightarrow (1_\Gamma \longrightarrow U)).$$

5 Representing data simplicially

We collect in this section some further examples of tables of data organized 'simplicially' and discuss briefly certain aspects of this representation. The sparse examples of Section 2 were meant to illustrate the technical definitions and were kept rather to a minimum. To begin with, the following examples give some further illustrations of context extension and Σ and Π-types. (We exploit in these examples, to simplify them a bit, the rather immediate back-and-forth translation between 'simplicial' schemas/instances and the traditional schemas/instances of the relational model. We omit part-relation tables when they can be assumed to be projections, for instance. We also make some cosmetic simplifications such as not necessarily writing the attributes of induced schemas as pairs.)

EXAMPLE 5.1
Let S be the schema and I the instance as follows:

State	Head of state
Monarchy	King
Monarchy	Queen
Republic	President

We write $S \vdash I$ for 'S is a schema and I is an instance over S'. Then the schema $S.I$ induced by I (p. 16) is:

Monarchy	King

Monarchy	Queen

Republic	President

Now let $S.I \vdash J$ be defined as follows:

Monarchy	Queen
United Kingdom	Elizabeth II
Denmark	Margrethe II

Monarchy	King
Norway	Harald V
Sweden	Carl XVI Gustav

Republic	President
Finland	Sauli Niinistö
Iceland	Ȯlafur Ragnar Grímsson

The dependent sum $S \vdash \Sigma_I J$ (p. 26) is then the instance collecting all the tables into the table of the original schema S:

State	Head of state
⟨ Monarchy, Norway⟩	⟨ King, Harald V⟩
⟨ Monarchy, Sweden⟩	⟨ King, Carl XVI Gustav⟩
⟨ Monarchy, United Kingdom⟩	⟨ Queen, Elizabeth II⟩
⟨ Monarchy, Denmark⟩	⟨ Queen, Margretha II⟩
⟨ Republic, Finland⟩	⟨ President, Sauli Niinistö⟩
⟨ Republic, Iceland⟩	⟨ President, Ólafur Ragnar Grímsson⟩

EXAMPLE 5.2
Let $R \vdash K$ be the schema and instance

Vehicle	Type	Wheels

Vehicle	Type
Vehicle	Type

Vehicle	Wheels

Type	Wheels
Type	Wheels

K is a ‘subsingleton’ or ‘subterminal’ instance. It has not more than one row in every table and is thereby a subinstance of the terminal instance 1_R that has exactly one row in each table (p. 16). Accordingly, K can be regarded as a subschema of R.

We form the induced schema $R.K$. Let $R.K \vdash L$ be the following instance:

Vehicle	Type
Ford Model T	Car
Triumph Tiger 100	Motorcycle
Peugeot Type 3	Car

Type	Wheels
Car	4
Motorcycle	2

The dependent product $R \vdash \Pi_A B$ (p. 20) can in this case be computed by instantiating the tables given by R in ascending order by taking the natural join of the tables in L lying (not necessarily properly) below it, thus the attribute level tables are those of L

Vehicle
Ford Model T
Triumph Tiger 100
Peugeot Type 3

Type
Car
Motorcycle

Wheels
2
4

And, similarly, the [Vehicle | Type] and [Type | Wheels] tables are those of L. While the [Vehicle | Wheels] table, for which L provides no information, is the natural join of [Vehicle] and [Wheels]

Vehicle	Wheels
Ford Model T	4
Ford Model T	2
Triumph Tiger 100	4
Triumph Tiger 100	2
Peugeot Type 3	4
Peugeot Type 3	2

And, finally, [Vehicle | Type | Wheels] is the natural join of [Vehicle | Type], [Type | Wheels] and [Vehicle | Wheels] as just given, thus:

Vehicle	Type	Wheels
Ford Model T	Car	4
Triumph Tiger 100	Motorcycle	2
Peugeot Type 3	Car	4

EXAMPLE 5.3
To give an example of the dependent product where the 'middle' instance is not a subschema, we can compute $S \vdash \Pi_I J$ for $S.I \vdash J$ of Example 5.1. Unfortunately, J has no full tuples; no choice of values for the attributes Monarchy-Republic-King-Queen-President yields a full tuple, as there are no monarchies in J with both a queen and a king head of state. Thus $S \vdash \Pi_I J$ becomes:

State	Head of state

State
⟨Denmark, Finland⟩
⟨Denmark, Iceland⟩
⟨Norway, Finland⟩
⟨Norway, Iceland⟩
⟨Sweden, Finland⟩
⟨Sweden, Iceland⟩
⟨United Kingdom, Finland⟩
⟨United Kingdom, Iceland⟩

Head of state
⟨Elizabeth, Harald, Niinistö⟩
⟨Elizabeth, Harald, Grímsson⟩
⟨Elizabeth, Carl Gustav, Niinistö⟩
⟨Elizabeth, Carl Gustav, Grímsson⟩
⟨Margrethe, Harald, Niinistö⟩
⟨Margrethe, Harald, Grímsson⟩
⟨Margrethe, Carl Gustav, Niinistö⟩
⟨Margrethe, Carl Gustav, Grímsson⟩

But if we let J be, instead

Monarchy	Queen
Swaziland	Ndlovukati
United Kingdom	Elizabeth II
Denmark	Margrethe II

Monarchy	King
Swaziland	Ngwenyama
Norway	Harald V
Sweden	Carl XVI Gustav

Republic	President
Finland	Sauli Niinistö
Iceland	Ólafur Ragnar Grímsson

Then the State–Head of state table of $\Pi_I J$ becomes

State	Head of state
⟨ Swaziland, Finland⟩	⟨Ndlovukati, Ngwenyama, Niinistö⟩
⟨ Swaziland, Iceland⟩	⟨Ndlovukati, Ngwenyama, Grímsson⟩

representing the two full tuples of J.

We proceed with an example suggesting the use of context extension—the built in possibility to enter data related to the instance into tables formed by its rows—for updates; an update I' of an instance I over S is the instance over $S.I$ obtained by writing the new (or old or empty) row in the table formed by the row to be replaced (or kept or deleted). Adding new rows can be done by writing I' over $S.I + 1$ instead, as $I + 1$ has a copy of S over which new additions can be entered. (Multiple copies of S, and indeed of I, can be added if need be; notice that polynomial expressions over I such as $2I + 3$ yield meaningful instances over S). In this way, a current update occurs in a context formed by a string of previous updates, thus displaying provenance. Applying the dependent product operation gives an instance over the original schema S, if desired. (In the following example, we return to writing the attributes of induced schemas as attribute-value pairs. For esthetic reasons, we write pairs as A:a, and we airbrush away the pairs with 0 and 1 used for the elements of the disjoint union.)

EXAMPLE 5.4
Let $P \vdash M$ be the schema and instance:

First name(s)	Last name	SSN	Department
Jim T.	Kirk	333	Astrophysics
James	Moriarty	222	Criminology

$P \vdash M + 1$ (p. 26) adds a dummy row:

First name(s)	Last name	SSN	Department
Jim T.	Kirk	333	Astrophysics
James	Moriarty	222	Criminology
⋆	⋆	⋆	⋆

An update that corrects Kirk's first name, deletes Moriarty and inserts two new rows can be written as an instance $P.(M+1) \vdash N$ (we abbreviate some of the strings involved):

FN: Jim T.	LN: Kirk	SSN: 333	Dep: Astro
James Tiberius	Kirk	333	Astrophysics

FN: James	LN: Moriarty	SSN: 222	Dep: Crime

FN:⋆	LN:⋆	SSN:⋆	Dep:⋆
Sherlock	Holmes	111	Criminology
James	Watson	555	Medicin

The dependent product $P \vdash \Sigma_{M+1}N$, collecting this over the original schema, becomes:

First name(s)	Last name	SSN	Department
Jim T.: James Tiberius	Kirk: Kirk	333: 333	Astro: Astro
⋆ : Sherlock	⋆ : Holmes	⋆ : 111	⋆ : Crime
⋆ : James	⋆ : Watson	⋆ : 555	⋆ : Med

The one condition on simplicial schemas that can hinder a straightforward translation from a relational schema is the condition that there can be at most one relation variable over a given set of attributes. In cases where there is a conflict and it is unnatural to rename attributes, this will have to be worked around, for instance by adding extra attributes to keep tables distinct.

EXAMPLE 5.5
Faculty and staff tables, both with intended attributes First name, Last name, SSN and Department. Separated with table name attribute with dummy value:

Staff	First name(s)	Last name	SSN	Department
⋆	Jim T.	Kirk	333	Astrophysics
⋆	James	Moriarty	222	Criminology

Faculty	First name(s)	Last name	SSN	Department
⋆	Khan N.	Sing	666	Astrophysics
⋆	Sherlock	Holmes	111	Criminology

Separated with primary key columns:

Staff_Id	First name(s)	Last name	SSN	Department
1	Jim T.	Kirk	333	Astrophysics
2	James	Moriarty	222	Criminology

Faculty_Id	First name(s)	Last name	SSN	Department
1	Khan N.	Sing	666	Astrophysics
2	Sherlock	Holmes	111	Criminology

In one table:

Staff/Faculty	First name(s)	Last name	SSN	Department
Staff	Jim T.	Kirk	333	Astrophysics
Staff	James	Moriarty	222	Criminology
Faculty	Khan N.	Sing	666	Astrophysics
Faculty	Sherlock	Holmes	111	Criminology

Finally, we illustrate the suggestion of using part tables to record missing information.

EXAMPLE 5.6
Holmes and Watson have unknown Social Security Numbers. Watson has not been assigned a department.

First name(s)	Last name	SSN	Dep
Jim T.	Kirk	333	Astroph

First name(s)	Last name	Dep
Jim T.	Kirk	Astroph
Sherlock	Holmes	Crimin

First name(s)	Last name
Jim T.	Kirk
Sherlock	Holmes
James	Watson

all other part tables are projections.

Funding

Supported by Norwegian Research Council grant no. 230525.

References

[1] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading MA, 1995.

[2] J. Cartmell. Formalising the network and hierarchical data models—an application of categorical logic. In David Pitt, Samson Abramsky, Axel Poigné and David Rydeheard, eds. In *Category Theory and Computer Programming, vol. 240 of LNCS*, pp. 466–492. Springer, Heidelberg, 1986.

[3] J. Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, **32**, 209–243, 1986.

[4] P. Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, eds. In *Types for Proofs and Programs, vol. 1158 of LNCS*, pp. 120–134. Springer, Heidelberg, 1996.

[5] G. Friedman. Survey article: an elementary illustrated introduction to simplicial sets. *The Rocky Mountain Journal of Mathematics*, **42**, 353–423, 2012.

[6] P. Gabriel and M. Zisman. *Calculus of Fractions and Homotopy Theory*. Springer, Heidelberg, 1967.

[7] M. Grandis. Finite sets and symmetric simplicial sets. *Theory and Applications of Categories*, **8**, 244–252, 2001.

[8] B. Jacobs. *Categorical Logic and Type Theory*. Elsevier, Amsterdam, 1999.

[9] S. M. Lane. *Categories for the Working Mathematician*. Springer, Heidelberg, 1998.

[10] P. Martin-Löf. *Intuitionistic type theory, vol. 1 of Studies in Proof Theory*. Bibliopolis, Naples, 1984.

[11] D. Spivak. *Simplicial Databases*. 2009. http://arxiv.org/abs/0904.2012.

Received 1 March 2016