Type Theoretical Databases

Henrik Forssell[1], Håkon Robbestad Gylterud[2](✉), and David I. Spivak[3]

[1] Department of Informatics, University of Oslo, Oslo, Norway
[2] Department of Mathematics, Stockholm University, Stockholm, Sweden
gylterud@math.su.se
[3] Department of Mathematics, MIT, Cambridge, MA, USA

Abstract. We show how the display-map category of finite 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. We indicate how it allows for the posing of queries. A novelty of the type theory is that it has non-trivial context constants.

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. 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

S. Artemov and A. Nerode (Eds.): LFCS 2016, LNCS 9537, pp. 117–129, 2016.
DOI: 10.1007/978-3-319-27683-0_9

the type theory has context constants, mirroring that our choice of "display maps" does not include all maps to the terminal object. But from the database perspective the interesting 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.

This 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 missing information or 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[1]. This requirement means that a relation variable can be identified with the set of its attributes. Together with the first requirement, 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* [7], 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 used to model the natural join operation. Second, it is a model of dependent type theory. We specify a dependent type theory with context constants and a sound interpretation which 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. The context constants are 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

[1] Coming from reasons having to do with simplicity and wanting to stay close to the view of tables as relations, this requirement, and indeed the structure of the schemas we are considering, does mean that a certain care has to be taken with attribute names at the modeling level. For instance whether one should, when faced with two tables with exactly the same attributes, collect these into one table (possibly with an extra column), rename some attributes, or introduce new "dummy" or "relation name" attributes to keep the two tables apart. For reasons of space, we do not discuss these issues here.

allows for the syntactic specification of both schemas and instances, and formally derives the answers to queries; for instance, using the dependent product of the type theory, the elements of the natural join of two instances can be derived in the type theory.

We focus, in the space available here, on the basic presentation of the model and the type theory (Sects. 2 and 3, respectively). In Sect. 4 we then give a few brief indications of the use and further development of the model and the type theory: we give a suggestion of how the dependent structure of the model can be put to use to model e.g. updates; and we sketch the introduction and use of large types such as, and in particular, a *universe*. The universe allows reasoning generically about classes of instances of in the type theory itself, without having to resort to the metalanguage, and provides the basis for e.g. a precise, formal definition and analysis of the notion of *query* in this setting.

2 The Model

2.1 Complexes

We fix the following terminology and notation, adjusting the standard terminology somewhat for our purposes. More details can be found in [4]. A background on simplicial complexes and simplicial sets can be found in e.g. [5,6]. The question of whether vertices or attributes should be ordered is not essential for our presentation here, and is swept under the rug.

A *simplicial complex*, or just *complex*, X consists of the union of a finite set X_0 and a collection $X_{\geq 1}$ of non-empty, non-singleton subsets of X_0, satisfying the condition that if x is a set in $X_{\geq 1}$, then all non-empty, non-singleton subsets of x are in $X_{\geq 1}$. It is convenient to also allow singleton subsets, and identify them with the elements of X_0. The *natural order* on X is then given by subset inclusion. The elements of X_0 are referred to as *vertices*. The elements of $X_{\geq 1}$ as *faces*. For $n \geq 0$, we write X_n for the set of elements of X with size $n+1$, and refer to them as *faces of dimension* n. Accordingly, vertices are seen as having dimension 0. We use square rather than curly brackets for faces, e.g. $[A, B]$ rather than $\{A, B\}$.

A morphism $f : X \longrightarrow Y$ of complexes is a function $f_0 : X_0 \longrightarrow Y_0$ satisfying the condition that for all x in X the image $f_0(x)$ is in Y (again, with a singleton identified with its element). Thus a morphism of complexes can be seen as a morphism of posets by setting $f(x)$ to be the image $f_0(x)$.

A morphism $f : X \longrightarrow Y$ of complexes is said to be a *display map* if $x \in X_n$ implies $f(x) \in Y_n$ for all n. Thus display maps are the maps that "preserve dimension". Display maps are also the "local isomorphisms", in the sense that for $x \in X$ the restriction $f \restriction_{(\downarrow x)} : (\downarrow x) \to (\downarrow f(x))$ is an isomorphism. Note that a display map need not be a injection of vertices.

A poset P that is isomorphic to a complex can clearly be uniquely rewritten to a complex with the same set of vertices as P. For that reason, and as it is occasionally notationally convenient and can yield more intuitive examples, we allow ourselves to extend the notion of complex to any poset that is isomorphic to a complex. We say that it is a *strict* complex if we need to emphasize that it is of the form defined above.

2.2 Schemas and Instances

Schemas. A *simplicial schema* is a complex with the natural order reversed. We consider the resulting poset as a category, in the usual way. If $x \leq y$ in the natural order, we write $\delta^y_x : y \longrightarrow x$ for the corresponding arrow in the simplicial schema. In the context of simplicial schemas, we use "attribute" and "relation variable" synonomously with "vertex" and "face", respectively. Let $\mathbb{S}$ be the category of simplicial schemas and morphisms, and $\mathbb{S}_d$ be the category of simplicial schemas and display maps.

With respect to the traditional notion of schema, a simplicial schema X can be thought of as given in the usual way by a finite set of attributes $X_0 = \{A_0, \ldots, A_{n-1}\}$ and a set of relational variables $X_{\geq 1} = \{R_0, \ldots R_{m-1}\}$, each with a specification of column names in the form of a subset of X_0, but with the restrictions (1) that no two relation variables are over exactly the same attributes; and (2) for any non-empty subset of the attributes of a relation variable there exists a relation variable over (exactly) those attributes. As with "complex", we henceforth mostly drop the word "simplicial" and simply say "schema".

The category $\mathbb{S}_d$ contains in particular the *n-simplices* Δ_n and the *face maps*. Recall that the the n-simplex Δ_n is the complex given by the set $\{0, \ldots, n\}$ as vertices and all non-empty, non-singleton subsets as faces. For $0 \leq i \leq n+1$, the face map $d^n_i : \Delta_n \longrightarrow \Delta_{n+1}$ is the morphism of complexes defined by the vertex function $k \mapsto k$, if $k < i$ and $k \mapsto k+1$ else. These satisfy the simplicial identities $d^{n+1}_i \circ d^n_j = d^{n+1}_{j-1} \circ d^n_i$ if $i < j$. As a schema, Δ_n is the schema of a single relation on $n+1$ attributes named by numbers $0, \ldots, n$ (and all its "generated" part-relations). The face map $d^n_i : \Delta_n \longrightarrow \Delta_{n+1}$ can be seen as the inclusion of the relation variable $[0, \ldots, i-1, i+1, \ldots, n+1]$ in Δ_{n+1}. These schemas and morphisms play a special role in Sect. 3 where they are used to specify general schemas and instances syntactically.

Example 1. Let S be the schema the attributes of which are A, B, C and the relation variables $R : AB$ and $Q : BC$, with indicated column names. From a "simplicial" point of view, S is the category

$$\begin{tikzcd}
 & R \arrow[dl, "\delta^R_A"'] \arrow[dr, "\delta^R_B"] & & Q \arrow[dl, "\delta^Q_B"'] \arrow[dr, "\delta^Q_C"] & \\
A & & B & & C
\end{tikzcd}$$

Replacing R with $[A, B]$ and Q with $[B, C]$ (and inverting the order) yields a strict complex. For another example, the 2-simplex Δ_2 can be seen as a schema on attributes 0,1, and 2, with relation variables $[0, 1, 2]$, and its part-relations. 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. f_0^{-1} does not define a morphism of schemas.

Instances. Let X be a schema, say with attributes $X_0 = \{A_0, \ldots, A_{n-1}\}$. A functor $F : X \longrightarrow \mathbf{FinSet}$ from X to the category of finite sets and functions can be regarded as an instance of the schema X. For $x = [A_{i_0}, \ldots, A_{i_{m-1}}] \in X$, the set $F(x)$ can be regarded as a set of "keys" or "row-names". The "value" $k[A_{i_j}]$ of such a key $k \in F(x)$ at attribute A_{i_j} is then the element $k[A_{i_j}] := F(d^x_{A_{i_j}})(k)$. Accordingly, $F(x)$ maps to the set of tuples $F(A_{i_0}) \times \ldots \times F(A_{i_{m-1}})$ by $k \mapsto \left\langle k[A_{i_0}], \ldots, k[A_{i_{m-1}}]\right\rangle$. For arbitrary F, this function 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, a relational instance is a functor $F : X \longrightarrow \mathbf{FinSet}$ such that for all $x \in X$ the functions $\{\delta^x_A \mid A \in x\}$ are jointly injective. Say that a relational instance is *strict* if the keys are actually tuples and the δ's are the expected projections.

Example 2. Let S be the schema of Example 1. 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

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

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

Let J be the strict instance $J : \varDelta_2 \longrightarrow \mathbf{FinSet}$ given in tabular form by

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

Explicitly, then, J is the functor which e.g. maps $[0,1]$ to $\{\langle a, b\rangle, \langle a', b\rangle\}$ and such that $J(\delta^{[0,1]}_1)(\langle a, b\rangle) = b$.

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

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 *strictification* of the former.

Example 3. Consider the morphism $f : S \longrightarrow \varDelta_2$ of Example 1 and the instances I and J of Example 2. Then $J[f]$ is the strictification of I.

Let $\mathrm{Rel}(X)$ be the category of strict relational instances and natural transformations between them. For convenience and brevity (as with complexes) we often disregard the requirement that instances need to be strict in the sequel. However, 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, both by the connection to the relational model and by the following formal connection between such instances and display maps.

Lemma 1. *Let $f : X \longrightarrow Y$ be a morphism of schemas. Then f is display if and only if for all strict instances J of Y the instance $J[f]$ is also strict.*

The connection between display maps, relational instances and simplicial schemas is given by the following. Let X be a schema and $F : X \longrightarrow \mathbf{FinSet}$ an arbitrary functor. Recall, e.g. from [8], 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. *Let X be a simplicial schema and $F : X \longrightarrow \mathbf{FinSet}$ be a functor. Then F is a relational instance if and only if $\int_X F$ is a simplicial schema and $p : \int_X F \rightarrow X$ is a display morphism.*

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

Example 4. The canonical schema of instance J of Example 2 has 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. $\langle [0, 1, 2], \langle a, b, c\rangle\rangle$ (or, strictly, $[\langle 0, a\rangle, \langle 1, b\rangle, \langle 2, c\rangle]$).

Full Tuples. 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. Accordingly, we define the *terminal instance* $1_X : X \longrightarrow \mathbf{FinSet}$ to be the functor defined by $x \mapsto \{x\}$.[2]

A *full* or *matching tuple* t of an instance I over schema X is a natural transformation $t : 1_X \Rightarrow I$. We write $\mathrm{Trm}_X(I)$ for the set of full tuples (indicating that we see them as terms type-theoretically).

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 5. The instance I of Example 2 has precisely two full tuples. A full tuple can be seen as a tuple over the full attribute set of the schema with the

[2] Strictly speaking, we choose an isomorphic representation which is strict and stable under substitution. For current purposes, however, the current definition is notationally convenient.

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 has precisely one full tuple $\langle a, b, c\rangle$.

2.3 Simplicial Databases

We have a functor $\mathrm{Rel}(-) : \mathbb{S}_d{}^{\mathrm{op}} \longrightarrow \mathit{Cat}$ which maps X to $\mathrm{Rel}(X)$ and $f : X \longrightarrow Y$ to $\mathrm{Rel}(f) = (-)[f] : \mathrm{Rel}(Y) \longrightarrow \mathrm{Rel}(X)$. 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 which we give in Sect. 3. We briefly outline some of the relevant structure available in $\mathfrak{R}$.

Definition 1. *For $f : X \longrightarrow Y$ in $\mathbb{S}_d$ and $J \in \mathrm{Rel}(Y)$ and $t : 1_Y \Rightarrow J$ in $\mathrm{Trm}_Y(J)$:*

1. *Define $t[f] \in \mathrm{Trm}_X(J[f])$ by $x \mapsto t(f(x)) \in J[f](x)$. Note that for $g : Z \longrightarrow X$ we have $t[f][g] = t[f \circ g]$.*
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 $\langle y, a\rangle \mapsto a$. (This term is needed for the type theory. 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 3. *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 $\tilde{p} \circ \hat{v} = \mathrm{Id}_{X.I}$.*

The following instance-forming operations exist and commute with substitution.

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

Dependent Sum: Let $X \in \mathbb{S}_d$, $J \in \mathrm{Rel}(X)$, and $G \in \mathrm{Rel}(X.J)$. We define the instance $\Sigma_J G : X \longrightarrow \mathbf{FinSet}$ up to strictification by

$x \mapsto \{\langle a, b\rangle \mid a \in J(x),\ b \in G(x, a)\}$. For δ^x_y in X, let $\Sigma_J G(\delta^x_y)(a, b) = \left\langle \delta^x_y(a), \delta^{x,a}_{y,\delta^x_y(a)}(b) \right\rangle$.

Identity: Given $X \in \mathbb{S}_d$ and $J \in \mathrm{Rel}(X)$ the *Identity instance* $\mathrm{Id}_J \in \mathrm{Rel}(X.J.J[p])$ is defined, up to strictification, by $\langle\langle x, a\rangle, b\rangle \mapsto \star$ if $a = b$ and $\langle\langle x, a\rangle, b\rangle \mapsto \emptyset$ else.

$\star$ being e.g. the empty tuple. The full tuple refl $\in \mathrm{Trm}_{(X.J)}(\mathrm{Id}_J[\hat{v}])$ is defined by $\langle x, a\rangle \mapsto \star$.

Disjoint Union: Given $X \in \mathbb{S}_d$ and $I, J \in \mathrm{Rel}(X)$, the instance $I + J \in \mathrm{Rel}(X)$ is defined up to strictification by
$x \mapsto \{\langle n, a\rangle \mid (n = 0 \wedge a \in I(x)) \vee (n = 1 \wedge a \in J(x))\}$. 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 $\langle x, a\rangle \mapsto \langle 1, a\rangle$.

Dependent Product: Let $X \in \mathbb{S}_d$, $J \in \mathrm{Rel}(X)$, and $G \in \mathrm{Rel}(X.J)$. We define the instance $\Pi_J G : X \longrightarrow \mathbf{FinSet}$ as strictification of the right Kan-extension (in the sense of e.g. [8]) of G along p. See [4] for an explicit construction. There are operations Ap and λ which for any full tuple $t \in \mathrm{Trm}_{X.J}(G)$ yields a full tuple $\lambda t \in \mathrm{Trm}_X(\Pi_J G)$, and for any full tuple $s \in \mathrm{Trm}_X(\Pi_J G)$ yields a full tuple $\mathrm{Ap}_s \in \mathrm{Trm}_{X.J}(G)$. Moreover, $\mathrm{Ap}_{\lambda t} = t$. We further indicate the relationship between the dependent product and full tuples, and the way in which the dependent product models the natural join operation, with the following example.

Example 6. Consider the schema S and instance I of Examples 1 and 2. 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 1 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 (again omitting single column tables),

	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

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 (see [4]).

3 The Type Theory

We introduce a Martin-Löf style type theory [9], with explicit substitutions (in the style of [3]), extended with context and substitution constants representing simplices and face maps. The type theory contains familiar constructs such as Σ- and Π-types. 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. Further details can be found in [4].

3.1 The Type Theory $\mathcal{T}$

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

Judgement	Interpretation
$? : \mathtt{Context}$	$[\![?]\!]$ is a schema
$? : \mathtt{Type}(\Gamma)$	$[\![?]\!]$ is an instance of the schema Γ
$? : \mathtt{Elem}(A)$	$[\![?]\!]$ 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
$A \equiv B : \mathtt{Type}(\Gamma)$	$[\![A]\!]$ and $[\![B]\!]$ are equal instances of $[\![\Gamma]\!]$
$t \equiv u : \mathtt{Elem}(A)$	$[\![t]\!]$ 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 Figs. 1 and 2. The interpretation of these are given by the constructions in the previous section, and summarised in Fig. 3.

$\sigma : \Gamma \longrightarrow \Delta,\ \tau : \Delta \longrightarrow \Theta$	$\vdash \tau \circ \sigma : \Gamma \longrightarrow \Theta$
$\Gamma : \mathtt{Context}$	$\vdash \mathrm{id}_\Gamma : \Gamma \longrightarrow \Gamma$
$A : \mathtt{Type}(\Gamma),\ \sigma : \Delta \longrightarrow \Gamma$	$\vdash A[\sigma] : \mathtt{Type}(\Gamma)$
$a : \mathtt{Elem}(A),\ \sigma : \Delta \longrightarrow \Gamma$	$\vdash a[\sigma] : \mathtt{Elem}(A[\sigma])$
$A : \mathtt{Type}(\Gamma)$	$\vdash \Gamma.A : \mathtt{Context}$
$A : \mathtt{Type}(\Gamma)$	$\vdash {\downarrow_A} : \Gamma.A \longrightarrow \Gamma$
$A : \mathtt{Type}(\Gamma)$	$\vdash v : \mathtt{Elem}(A[\downarrow_A])$
$a : \mathtt{Elem}(A)$	$\vdash a{\uparrow} : \Gamma \longrightarrow \Gamma.A$
$A : \mathtt{Type}(\Gamma),\ \sigma : \Delta \longrightarrow \Gamma$	$\vdash \sigma.A : \Delta.A[\sigma] \longrightarrow \Gamma.A$
	$\vdash \Delta_n : \mathtt{Context}$
	$\vdash d_i^n : \Delta_n \longrightarrow \Delta_{n+1}$

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

Fig. 1. Rules of the type theory: contexts and substitution

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

$A : \mathtt{Type}(\Gamma),\ B : \mathtt{Type}(\Gamma.A)$	$\vdash \Pi_A B : \mathtt{Type}(\Gamma)$
$b : \mathtt{Elem}(B)$	$\vdash \lambda b : \mathtt{Elem}(\Pi_A B)$
$f : \mathtt{Elem}(\Pi_A B)$	$\vdash \mathrm{apply}_f : \mathtt{Elem}(B)$
$A : \mathtt{Type}(\Gamma),\ B : \mathtt{Type}(\Gamma.A)$	$\vdash \Sigma_A B : \mathtt{Type}(\Gamma)$
$A : \mathtt{Type}(\Gamma),\ B : \mathtt{Type}(\Gamma.A)$	$\vdash \mathrm{pair} : \mathtt{Elem}(\Sigma_A B[\downarrow_A][\downarrow_B])$
$C : \mathtt{Type}(\Gamma.\Sigma_A B),$	
$c_0 : \mathtt{Elem}(C[(\downarrow_A \circ \downarrow_B).\Sigma_A B][\mathrm{pair}\uparrow]$	$\vdash \mathrm{rec}_\Sigma\ c_0 : \mathtt{Elem}(C)$
$A : \mathtt{Type}(\Gamma)$	$\vdash \mathrm{Id}_A : \mathtt{Type}(\Gamma.A.A[\downarrow_A])$
$A : \mathtt{Type}(\Gamma)$	$\vdash \mathrm{refl} : \mathtt{Elem}(\mathrm{Id}_A[v\uparrow])$
$C : \mathtt{Type}(\Gamma.A.A[\downarrow_A].\mathrm{Id}_A),$	
$c_0 : \mathtt{Elem}(C[(v\uparrow).\mathrm{Id}_A][\mathrm{refl}\uparrow])$	$\vdash \mathrm{rec}_{\mathrm{Id}}\ c_0 : \mathtt{Elem}(C)$
$A : \mathtt{Type}(\Gamma)\, B : \mathtt{Type}(\Gamma)$	$\vdash A + B : \mathtt{Type}(\Gamma)$
$A : \mathtt{Type}(\Gamma)\, B : \mathtt{Type}(\Gamma)$	$\vdash l_{A,B} : \mathtt{Elem}((A+B)[\downarrow_A])$
$A : \mathtt{Type}(\Gamma)\, B : \mathtt{Type}(\Gamma)$	$\vdash r_{A,B} : \mathtt{Elem}((A+B)[\downarrow_B])$
$C : \mathtt{Type}(\Gamma.(A+B)),$	
$c_0 : \mathtt{Elem}(C[(\downarrow).(A+B)][l_{A,B}])$	
$c_1 : \mathtt{Elem}(C[(\downarrow).(A+B)][r_{A,B}])$	$\vdash \mathrm{rec}_+\ c_0\, c_1 : \mathtt{Type}(C)$
$\Gamma : \mathtt{Context}$	$\vdash 0 : \mathtt{Type}(\Gamma)$
$A : \mathtt{Type}(\Gamma.0)$	$\vdash \mathrm{rec}_0 : \mathtt{Elem}(A)$
$\Gamma : \mathtt{Context}$	$\vdash 1 : \mathtt{Type}(\Gamma)$
$\Gamma : \mathtt{Context}$	$\vdash * : \mathtt{Elem}(1)$
$A : \mathtt{Type}(\Gamma.1), a : A[*\uparrow]$	$\vdash \mathrm{rec}_1\ a : \mathtt{Elem}(A)$

Fig. 2. Rules of the type theory: Types

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

Fig. 3. Interpretation of the type theory

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 Fig. 4.

$$\begin{array}{rcc}
(\upsilon \circ \tau) \circ \sigma \equiv \upsilon \circ (\tau \circ \sigma) & id_\Gamma \circ \sigma \equiv \sigma & \sigma \circ id_\Gamma \equiv \sigma \\
a[\sigma \circ \tau] \equiv a[\sigma][\tau] & a[id_\Gamma] \equiv a & \downarrow_A \circ (a\uparrow) \equiv id_\Gamma \\
v[a\uparrow] \equiv a & \downarrow_A \circ (\sigma.A) \equiv \sigma \circ \downarrow_{A[\sigma]} & v[f.A] \equiv v \\
(\sigma.A) \circ (a[\sigma]\uparrow) \equiv a\uparrow \circ \sigma & (\sigma.A) \circ (\tau.A[\sigma]) \equiv (\sigma \circ \tau).A & \downarrow .A \circ v\uparrow \equiv Id_{\Gamma.A} \\
\Pi_A B[\sigma] \equiv \Pi_{A[\Sigma]} B[\sigma.A] & \text{apply}(\lambda b) \equiv b &
\end{array}$$

Fig. 4. Definitional equalities in the type theory

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 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 $A : \mathtt{type}(\Gamma)$ is that A is an instance of the schema Γ. However, context extension allows us to view every instance as a schema in its own right; for every instance $A : \mathtt{type}(\Gamma)$, we get a schema $\Gamma.A$. 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 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 7. We construct the rules of the schema S presented as an instance of Δ_2 as in Example 6. The introduction rules tells us the names of tables and attributes in S.

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

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.

An instance of a schema is a type depending in the context of the schema. Therefore instance specification is completely analoguous to schema specification. See [4] for an example. In [4] one can also find a derivation of the terms in $\mathcal{T}$ corresponding to the full tuples of the natural join in Example 6.

4 Dependent Structure, Large Types, and Queries

Most of this paper has been devoted to explaining the basic structure of the display map category of finite simplicial complexes seen as encoding systems of tables, and to stating the type theory which it models. In the space that remains, we briefly indicate some approaches of ongoing and future work, in particular emphasizing the definitions and roles of large types and universes. Before introducing additional types, however, we point to the use that can be made of the dependent structure itself. It is a cornerstone of the model that an instance over a schema can itself, by context extension, be seen as a schema over which new instances can be constructed. Thus context extension provides, for a given instance, the built in possibility to enter data related to the instance into tables formed by its rows. One immediate suggestion for the potential use of this feature is for updates; an update I' of an instance I over Γ is the instance over $\Gamma.I$ 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 $\Gamma.I+1$ instead, as $I+1$ has a copy of Γ over which new additions can be entered. (Multiple copies of Γ, and indeed of I, can be added if need be; notice that polynomial expressions over I such as $2I+3$ yield meaningful instances over Γ). In this way a current update occurs in a context formed by a string of previous updates. Applying the dependent product operation gives an instance over the original schema Γ, if desired.

Returning to large types, Example 6 gives a glimpse of a "type-theoretic" operation and its relation to one of the standard queries of relational databases. A formal investigation of queries (and dependencies and views) in the setting of the type theory and model of the previous sections requires a formal understanding of what constitutes a query in this setting. For (partly) this purpose, we introduce a *universe*. This is a large type, corresponding to an infinite instance which encodes all finite instances over a fixed domain. Thus, given a schema X, the universe U_X of finite instances of X is an infinite instance of X where the full tuples encode the finite instances of X (over the fixed domain). The universe comes equipped with a *decoding instance* T_X over $X.U_X$ such that given a full tuple $t \in \mathrm{Trm}_X(U_X)$, the instance it encodes is decoded by $T_X[\widehat{t}]$,

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

The universe and decoding instance are stable under substitution, and are closed under the other type constructions, such as Π- and Σ-types. We omit the details of the constructions.

The universe, U_Γ along with $T_\Gamma : \texttt{Type}(\Gamma.U_\Gamma)$, 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 $t : \texttt{Elem}(\Omega_\Gamma)$, the subschema it encodes is $\Gamma.\mathcal{O}[t\uparrow]$.

A query can then be seen as an operation which takes an instance of a schema to another instance of a related schema. Given codes for a source subschema $t : \texttt{Elem}(\Omega)$ and a target subschema $u : \texttt{Elem}(\Omega)$, the type of queries from t to u is thus $(\mathcal{O}[t\uparrow] \to U) \to (\mathcal{O}[u\uparrow] \to U)$. 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 8. In the spririt of Example 6, let $a : \texttt{Elem}(\Omega)$ 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

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

Further new types of interest and use are the type Λ_Γ of "tables", or faces, of a schema and the (large) type $\mathbb{N}_\Gamma$ of "finite" instances, or instances of the form $1 + 1 + \ldots + 1$. The former can be constructed so as to be both small and stable (whereas Ω_Γ seems to have to be large in order to be stable), and is sufficient for schema and instance specification (cf. Example 7). The latter is of relevance e.g. with respect to instances determined by their full tuples (or "with no nulls").

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Cartmell, J.: Formalising the network and hierarchical data models — an application of categorical logic. In: Pitt, D., Abramsky, S., Poigné, A., Rydeheard, D. (eds.) Category Theory and Computer Programming. LNCS, vol. 240, pp. 466–492. Springer, Heidelberg (1986)
3. Dybjer, P.: Internal type theory. In: Berardi, S., Coppo, M. (eds.) TYPES 1995. LNCS, vol. 1158. Springer, Heidelberg (1996)
4. Forssell, H., Gylterud, H.R., Spivak, D.I.: Type theoretical databases (2014). http://arxiv.org/abs/1406.6268
5. Friedman, G.: Survey article: an elementary illustrated introduction to simplicial sets. Rocky Mt. J. Math. **42**(2), 353–423 (2012)
6. Gabriel, P., Zisman, M.: Calculus of Fractions and Homotopy Theory. Springer, Heidelberg (1967)
7. Jacobs, B.: Categorical Logic and Type Theory. Elsevier, Amsterdam (1999)
8. Mac Lane, S.: Categories for the Working Mathematician. Springer, Heidelberg (1998)
9. Martin-Löf, P.: Intuitionistic Type Theory. Studies in Proof Theory, vol. 1. Bibliopolis, Naples (1984)