Acta Cybernetica **24** (2019) 211–248.

The Logic of Aggregated Data

Tjalling Gelsema[*]

Abstract

A notion of generalization-specialization is introduced that is more expressive than the usual notion from, e.g., the UML or RDF-based languages. This notion is incorporated in a typed formal language for modeling aggregated data. Soundness with respect to a sets-and-functions semantics is shown subsequently. Finally, a notion of congruence is introduced. With it terms in the language that have identical semantics, i.e., synonyms, can be discovered. The resulting formal language is well-suited for capturing faithfully aggregated data in such a way that it can serve as the foundation for corporate metadata management in a statistical office.

Keywords: aggregation, information modeling, semantics

Introduction

This article is a sequel to [8]. The reader is therefore advised to make her- or himself acquainted of the notions and the results of [8]. Also, we inform the reader that this article is written from the perspective of information management in a statistical office. Its results however, we feel, are applicable in any situation where large quantities of aggregated datasets need to be managed faithfully.

One of the distinguishing features of a national statistical institute (NSI) is the large amount of information it harbors, dealing with many different social and economic phenomena. Managing this mass of information, i.e., making sure that the right pieces of information are available at any situation where they are required, is both necessary as well as nontrivial. For various reasons — accuracy, professionalism, transparency and coherence to name a few — an NSI should be aware of all the social and economic concepts it uses to produce statistics, across the entire process from data collection to publication.

While Statistics Netherlands (SN) is very keen on managing these concepts for the statistics that are published on its website, overall office-wide management of information across the statistical production process lags behind. Apart from the obvious, such as low risk of public exposure, there are various reasons why office-wide information management receives less attention and is difficult to achieve.

[*]Statistics Netherlands, Henri Faasdreef 312, 2492 JP Den Haag E-mail: `te.gelsema@cbs.nl`

DOI: 10.14232/actacyb.24.2.2019.4

First, true office-wide information management requires local investments, from the various departments the organization consists of, while the return on the investments is often less apparent from the point of view of these departments. The administration, i.e., the proper naming and describing, of the numerous variables, classifications, datasets, production rules, etcetera, is sometimes considered drudgery and mostly for the benefit of others.

Second, variables, classifications, statistical information models, micro data and aggregated data show dependencies that makes proper management of, say, variables through a variable management system, separately from that of, say, classifications through a classification management system, hard to realize. As an example of such a dependency, consider variables like *turnover generated from the sales of shoes*, *turnover generated from the sales of jeans*, etcetera. These variables are properly managed only if they are seen as one generic variable, *turnover generated from the sales of a good* say, that is 'indexed' by some classification of goods: only then need changes, e.g., in the definition of the variables, be recorded only once instead of for each of the indexed variables separately. For the converse, classifications that depend on variables, examples can be given as well.

In many organizations such as SN there is a need for *corporate metadata management*, which is aimed at integrating these separate initiatives in a meaningful way. However, uncovering dependencies that are meaningful from the perspective of corporate metadata management is harder than is acknowledged by initiatives such as the GSIM [22]. Besides, as a separate discipline, studying structural metadata, which is aimed at discovering these dependencies, has received too little attention in the scientific literature anyway (but see, e.g., [20]). In the sequel we will give some examples that fail to be handled properly by existing metadata frameworks. As we will see, having access to such dependencies can make various tasks much easier, but the systems or frameworks that should record them, we claim, are either lacking or are inadequate. As a result, the administrative tasks mentioned earlier are not properly supported and they therefore require more human effort than necessary.

SN has initiated a program to gain an office-wide overview of all of its datasets that are one 'steady state' behind its output tables in the statistical process: datasets that have reached a certain stability in the sense that they are no longer subject to changes, and are one step away from publishing. This is part of a larger effort to arrive at a complete overview, from data collection to publishing, which is carried out in the opposite direction (i.e., from publishing to data collection). During the years 2013 and 2014, these 'pre-output tables' were named and described both in general terms as well as in terms of the variables they consist of, in a joint effort that involved every department responsible for publishing statistics. The goal was to have office-wide access, by the beginning of 2015, to both the descriptions of these datasets (the so-called dataset designs) and through them authorized access to the datasets themselves. To achieve this, a separate department has been established, the Data Service Center (DSC), which is responsible for offering access to these data for the rest of the organization. The DSC also develops and manages the automated system for storing and retrieving datasets and dataset designs, offers guidelines for naming and describing datasets and variables, and offers overall support to statisticians in applying these guidelines, to name a few of the DSC's responsibilities.

While the DSC has put much effort in what we refer to as the 'non-structural' business rules of dataset design, among which are the naming conventions, the 'structural' business rules have received much less attention. Among these 'structural' business rules are the ones that, given an information model, determine whether a group of variables can be reasonably put together to form a dataset, or that determine, given two variables, whether the one is an aggregated form of the other, as with *total turnover generated by a class of businesses* and *turnover of a business*. The result is that the automated system that the DSC provides is not sufficiently capable of responding to natural queries that arise when datasets are designed, which requires a more global and interdependent view of all information assets than the DSC is able to provide.

One example of such a query is: given an object type (such as one of the types *person*, *household*, or *business*) list all the currently available variables that apply to that type (such as *age* and *income*, *household composition* and *income*, and *turnover* and *profit*, respectively). Another example is: given a family of variables, such as *turnover generated from the sales of shoes* and *turnover generated from the sales of jeans* examined earlier, list the generic variable and the classification that they depend on. The first query is natural, because it stimulates the reuse of existing variables. It is difficult to respond to, because the DSC does not record object subtyping: e.g., the variable *age* (recorded as a variable of type *person*) will be missing from the list when all variables of type *student* are requested. The second query is natural, again for purposes of reuse: the description of the generic variable together with the descriptions of each of the categories in the classification are sufficient to understand each of the variables in the family. The automated system of the DSC cannot respond to the query though, since it does not record this kind of dependency between variables and classification systems. As a result, each variable in the family needs to be described separately, which is an example of the extra human effort that is required because of inadequate support from DSC's information systems.

In order to lay down useful dependencies, we claim it is natural and beneficiary to have a graph-like perception of corporate metadata. Then, we claim, it is equally natural to view the structure of a dataset as a particular subgraph. To illustrate what we mean, consider the dataset below that records the ages and the incomes of two partners in a marriage, as well as the duration of the marriage. (We assume that we list these for all marriages that existed on a particular date, say January 1 2016, in a particular residential area, say Delft. We also assume that *partner 1* and *partner 2* in the marriage, abbreviated by 'pa.1' and 'pa.2' respectively, can be identified according to some criterion.)

The peculiarity with this dataset is that, without further ado, the correct understanding of it depends on the correctness of the labels 'pa.1' and 'pa.2' assigned to the first four columns. These labels are easily switched and switching them probably gets unnoticed on first sight. So the dataset has some internal structure in the

age pa.1	income pa.1	age pa.2	income pa.2	duration
31	20.500	35	22.000	4
62	40.200	57	45.000	31
...	...	...	...	...

Table 1: Data about partners in a marriage

sense that the first and the second column of data need to be treated as a pair, as well as the third and fourth column. Other than assigning labels to columns, the DSC however has no means to formally record such information, and neither has the GSIM.

We propose that these useful dependencies, between columns in a dataset in the example above, can be adequately expressed in the form of the graph below, which should be viewed as an information model of a small part of the 'real world' a statistician might be interested in. To summarize what is depicted in Fig. 1,

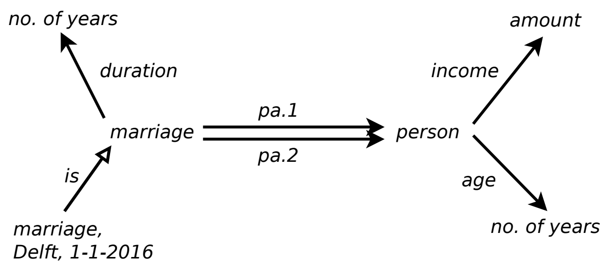

Figure 1: An information model

associated with a *marriage* are two *persons*, which are the partners involved in the marriage. Each person has characteristics *age* and *income*, expressed as a *no. of years* and an *amount*, respectively. A marriage has a characteristic *duration* which is also recorded as a *no. of years*. Each marriage recorded in Delft on January 1 2016 is a *marriage*.

The difference between what we call 'non-structural' and 'structural' business rules, respectively, is that only the latter can be expressed in a form sufficiently precise to be automatically (and correctly) interpreted by a computer program, in order to automatically respond to queries such as the ones above. While we feel that both 'non-structural' and 'structural' business rules are important in the development and use of automated systems that are successful in supporting information management in an NSI, studies of the latter are practically nonexistent in the scientific literature, at least when restricted to statistical information management or statistical metadata. There is however a huge source of literature available about techniques in formal semantics [23, 1], a subfield of theoretical computer science, which, we feel, has valuable applications to statistical information management. This article is an attempt to show that these techniques can be successfully applied to solve the questions raised above.

In order to further delineate the scope of this article, we postulate that within the statistical process, there are two kinds of data transformations: those that change the structure of the data and those that keep the structure intact, but only change the contents of a dataset. Examples of the first kind are aggregation and row selection, examples of the second kind are data editing and imputation. We further postulate that, roughly, transformations of the first kind change or produce structural metadata, while transformations of the second kind change or produce quality metadata. Simply put, the second kind deals with changes of the *estimator*, while the first kind deals with changes of the *estimand*. This article only deals with changes of the *estimand* and hence only deals with changes in structural metadata, keeping quality metadata out of scope.

In [8] formal semantics, initial algebra semantics [9] and some category theory [13, 18, 3] in particular, were taken as a point of departure for developing 'structural' business rules for statistical data and metadata. The main ideas of [8] can be summarized as follows:

(i) The most natural and accurate interpretation of statistical data is the sets-and-functions interpretation. In this interpretation, variables, micro datasets, dimensional datasets, relations between object types (such as between a *marriage* and a *person* in Fig. 1) and also values[1] are seen as functions; object types and value types (such as *amount* in Fig. 1) are seen as sets. In this view for instance, a typical variable v takes an object (a person, a household, a business) from a set of objects p (the interpretation [8] gives to an object type) to a value taken from a set of values x (the interpretation of a value type). Hence we have the function $v : p \to x$;

(ii) To go from, e.g., variables and object type relations to a dataset and from a micro dataset to a dimensional dataset requires *operators* that act on sets and functions. For instance, in [8], column- and row-wise combining, functional composition and aggregation were considered as operators and were defined in the sets-and-functions interpretation;

(iii) Structural metadata should be 'aligned' in a natural way with these operators. This means that, for instance, combining variables column-wise in order to produce a dataset is 'mirrored' in combining descriptions of these variables in order to produce a description of the dataset. The mathematical consequence of this idea, using the initial algebra semantics point of departure, is that structural metadata are *terms*, built from the operators mentioned in (ii), that are mapped onto the sets and functions of (i) by means of a homomorphism.

In this article, terms that represent sets are called *types*; terms that represent functions are called *elements.* Elements have a *domain* and a *codomain*, which are types.

[1]We will see how values are interpreted as functions in the next section.

In the view of [8], the structural metadata for the dataset in Table 1 is the term

$$\langle age \circ pa.1, income \circ pa.1, age \circ pa.2, income \circ pa.2, duration\rangle \circ in,$$

which is an element constructed using the operators column-wise combining $\langle \ldots \rangle$ and functional composition $\circ$, that has the type

$$\textit{marriage, Delft, 1-1-2016}$$

as a domain, and the type

$$no.\ of\ years \times amount \times no.\ of\ years \times amount \times no.\ of\ years$$

as a codomain. Note that the nodes that are incident with the directed edges from the graph in Fig. 1 are types that ensure the correct construction of the element above. For instance, functional composition $\circ$ is defined only for two elements (cf. the directed edges of Fig 1) of which the node incident on the tail (its domain) of the first equals the node incident on the head (its codomain) of the second. This implies that structural metadata expressed as an element, such as the one above, is safer than expressed through labels, which is the method that the DSC and the GSIM must resort to.

In addition, the 'structural' business rules that we need to answer the queries that we saw above require the following idea:

(iv) 'Structural' business rules for metadata are equivalences of terms, which, in the sets-and-functions interpretation of (i) become identities through the homomorphism of (iii). An example of such an identity is the fact that, under certain conditions, a two-stroke aggregation process can be reduced to a one-stroke aggregation step (see [8], Property (6)).

The contribution of this article to the ideas of [8] is twofold. First we extend the set of operators of [8] by a subtype operator and prove properties of it in the context of the other operators. Second, we prove that these properties — the 'structural' business rules — can be used to define a language for expressing structural metadata. For technical reasons explained below, this is more involved than it was for the set of operators of [8]. We explain these contributions briefly.

The idea of subtyping as an instrument of statistical information management, is that object types and their subtypes form a grouping mechanism for variables. A variable such as *age of a person* is recorded only once, viz. at the most generic type to which it applies: the object type *person*. Subtyping, e.g., the notion that any *student* is a *person*, allows access to that variable (e.g., *age of a student*) at more specific levels (*student*). This is the usual concept of inheritance; one of the foundations of the object-oriënted paradigm [15]. The open-headed generalization-specialization arrow of the Unified Modeling Language (UML, see [15]) is the usual way of recording subtyping; note that there is an occurrence of such an arrow in Fig. 1. On the other hand, more specific types (*student*) can give rise to variables (e.g., *year of application*) that do not apply at the generic level. Thus, subtyping is an ordering of types that induces an ordering of groups of variables: the more generic the type, the less variables apply to it.

The notion of subtyping given in this article is more involved however than the usual notion from the UML or other ontology languages. In a(n automated) statistical process we usually need to know the justification for calling one type a subtype of another, in order to decide whether or not an object is a member of the subtype. Thus, we require that a subtype can only be defined — we prefer to say: *constructed* — out of a given type, if some selection criterion is supplied. This selection criterion can be a combination of a variable, applicable at the generic type, and a value in the range of that variable. For instance, we require that the construction of the object type *student* from *person* is supplied with a variable, say *is registered at a university?*, and with the value *yes* applicable to that variable.

The logical consequences of defining a subtype operator in this way are rather great, though. In [8], the universe of types could be defined independently of the universe of elements, which is a requirement for the use of equational logic [16] for defining the language as an initial algebra. With subtyping this is no longer the case, as the construction of a subtype depends on an element (e.g., the variable *is registered at a university?* in the example above). This means that, in a logical sense, there is an extra effort needed to untie the following knot:

a The universe of elements depends on the universe of types, as the domain and codomain of an element are both types;

b The universe of types depends on the universe of elements, by the subtype operator.

The article is organized as follows: in Sections 4 and 5 we define our language for structural metadata, from scratch, i.e., without support of any theory (such as equational logic) other than universal algebra [10]. After the Preliminaries of Section 1, in Sections 2 the subtype operator is introduced and its properties are shown there and in Section 3. To stress that Sections 2 and 3 are dealing with data, i.e., with sets and functions, we refer to the subtype operator there as *subset inclusion.* In Section 6 the semantics of the language is sketched and in Section 7 we give some examples that indicate the expressiveness of the language. We conclude with Section 8.

1 Preliminaries

We recall some of the notions of [8] and introduce some new ones.

For a set p, let Fp be the set of finite subsets of p. For sets x and y, let $x \times y$ be the binary Cartesian product of x and y, i.e., the set of all pairs $\langle d, e\rangle$ with $d \in x$ and $e \in y$. More generally, for any number $n > 1$, the Cartesian product of sets x_i, $i \leq n$, is denoted $x_1 \times \cdots \times x_n$ and consists of all n-tuples with elements taken from $x_1, \ldots, x_n$, respectively.

We let $1 = \{*\}$ be an arbitrary but fixed singleton set. We denote the empty set $\{\}$ by 0. Note that $x_1 \times \cdots \times x_n = 0$ if at least one of the x_i equals 0.

A commutative monoid is a structure $m = (m; +, 0)$ with m a set, $+$ an associative and commutative binary operation on m, and with $0 \in m$ an identity for $+$ (not to be confused with the empty set). More details can be found in [8]; the reader could consult [5] for full details. We refer to m simply as a monoid, since all monoids we consider here are commutative. For monoids m and $m' = (m'; +', 0')$, a function $h : m \to m'$ is a (monoid) homomorphism, if $h(a + b) = h(a) +' h(b)$ for all $a, b \in m$, and $h(0) = 0'$.

For a function $v : p \to x$, we call p the domain of v and x the codomain of v; the set of functions with domain p and codomain x is denoted by x^p. If v is considered to be a variable, then we say that v is defined on (the population) p and that v is defined for (the value set) x. All functions we consider in this article are total; for v as before this means that it associates with every e in p exactly one d in x, and then this d is denoted by $v(e)$. If, conversely, every d in x is associated with at most one e in p through $v(e) = d$, then v is an injection. The composition of $v : p \to x$ with $w : q \to p$ is the function $v \circ w : q \to x$ defined by $v \circ w(d) = v(w(d))$ for every $d \in q$. We will normally abbreviate $v \circ w$ by vw. The composition of two injections is an injection. The left and right identities for composition are the identity functions: if we let v as before, then $v \circ \mathsf{id}_p = v = \mathsf{id}_x \circ v$, where $\mathsf{id}_p : p \to p$ is defined by $\mathsf{id}_p(e) = e$ for every $e \in p$, and similarly for id_x.

The product (or: 'column-wise' combination) of n functions $v_i : p \to x_i$, $1 \leq i \leq n$ with $n > 1$, is defined in the following way: we let $\langle v_1, \ldots, v_n \rangle : p \to x_1 \times \cdots \times x_n$ be the function u defined by $u(e) = \langle v_1(e), \ldots, v_n(e) \rangle$ (i.e., an n-tuple) for all $e \in p$. Note that the product is defined for functions with a common domain only. Given x_i as above, we let $\pi^n_i : x_1 \times \cdots \times x_n \to x_i$, called the i-th projection, be the function that maps an n-tuple $\langle d_1, \ldots, d_n \rangle$ to the element d_i in x_i. Note that π^n_i is a family of functions: for every combination of $n > 1$ and i with $1 \leq i \leq n$, and every combination of sets $x_1, \ldots, x_n$, we assume that the i-th projection π^n_i applies as defined.

We stress that this article excludes the 'row-wise' combination of functions, an operation that was included in [8].

A function $w : p \to q$ is inverse-finite if $w^{-1}(d) = \{e \in p \mid w(e) = d\}$ is finite for every $d \in q$. The composition of two inverse-finite functions is inverse-finite and every injection is inverse-finite. For an inverse-finite w with domain and codomain as before, let $\delta(w) : q \to Fp$ be the function that maps an element $d \in q$ to the finite set $w^{-1}(d)$. Note that $\delta(w)$ is not an injection in general. For a function $v : p \to m$ with m a monoid, we let $\gamma(v) : Fp \to m$ be the function that maps a finite nonempty set $\{e_1, \ldots, e_k\}$ to $v(e_1) + \cdots + v(e_k)$, and the empty set 0 to the monoid identity 0. The mappings $\delta(w)$ and $\gamma(v)$ are called the dimensional structure induced by w and the elementary class parameter induced by v, respectively. Their composition $\gamma(v)\delta(w)$, called the aggregation of v by w, is denoted by $\alpha(v, w)$. Note that the composition 'factors through' Fp.

Aggregation, as defined by α, captures most of the more common aggregation operators, such as sums, maxima and minima, and (weighted) averages (see [8]). We conjecture however that medians are not covered by α.

For every $d \in x$ there is a unique function $\vec{d} : 1 \to x$ with $\vec{d}(*) = d$, and conversely, every function $1 \to x$ 'picks out' a unique element in x. There is thus a one-to-one correspondence between x and the set of functions with domain 1 and codomain x. It is therefore natural, but not so common, to identify an element d in x with the function $\vec{d}$. Note that, for a function $v : x \to y$, we can then equally identify $v(d)$ with $v\vec{d}$ (since $v(d) = (v\vec{d})(*)$) and even with vd if we omit notational differences. To sum up: in the sequel the phrase "d is an element of x" can mean $d \in x$ or it can mean $d : 1 \to x$; it will always be clear from the context whichever applies.

For each set x there is exactly one function with domain x and codomain 1 (viz. every element in x has $*$ as its image) and we write 1_x to denote this function. Also, for each set x there is a unique function with domain 0 and codomain x and we denote this function by 0_x.

We adopt the following notational conventions: we use the letters p, q, r and $x, x_1, \ldots x_i, \ldots, x_k, y, z$ to denote sets, we use a, b, c, d, e to denote elements from these sets (potentially interpreted as functions with 1 as domain, as explained above), we use u, v, w to denote arbitrary functions, m, m' to denote monoids, and h, g to denote monoid homomorphisms.

The reader is encouraged to interpret a set denoted by p, q or r as a set of 'objects (or entities) of statistical interest'. This gives an informal meaning to the notion of an 'object type', such as the object type *person* or the object type *household*: informally an 'object e is of type p' if $e \in p$. Similarly, any of the sets $x, x_1, \ldots, x_k, y, z$ and m, m' should be interpreted as a set of 'values', i.e., as a rough interpretation of a 'value type'. However, within the context of this article, we make no mathematical distinction between 'values' and 'objects', except perhaps in the case of a 'value type' that supports aggregation: we require that such a type is a monoid. (We also advocate that a classification system is a Boolean algebra — see [7] — but that is outside the scope of this article.) Thus formally we do not make any distinction between a set of 'objects' and a set of 'values'. Hence in particular, the general Cartesian product is also defined for sets of 'objects', or for any combination of sets of 'objects' and sets of 'values', and the finite powerset operator F also applies to sets of 'values'.

Thus, when we write, e.g., $v : p \to x$ then v informally corresponds to the typical notion of a variable: a mapping that takes an object e of object type p to a value $v(e)$ of value type x. However, any mathematical rule to distinguish a variable from an arbitrary function is problematic: should we call a mapping of the form $p \to x \times y$ a variable or not? And what about a mapping of the form $p \times x \to y$? Also: is $p \times q$ an object type when both p and q are designated as object types? See [8] for a more conceptual discussion on these issues.

We proceed formally. Below we list the most important identities involving composition, product and aggregation, discovered and proven in [8]. We assume that h is a monoid homomorphism in Equation 5, that w is inverse-finite in Equations 4, 5 and 6, and that w is an injection in Equation 7. We urge the reader to draw a diagram of the situation for each of the identities, in which proper domains and codomains of the functions involved are depicted as vertices, and the functions are depicted as directed edges, in a way similar to Fig. 2, which sketches the situation

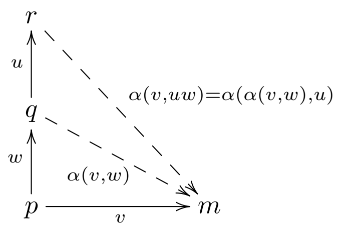
$$\begin{tikzcd}
r & \\
q \arrow[u, "u"] \arrow[dr, dashed, "\alpha(v,w)"'] & \\
p \arrow[u, "w"] \arrow[r, "v"'] & m \arrow[uul, dashed, leftarrow, "\alpha(v,uw)=\alpha(\alpha(v,w),u)"']
\end{tikzcd}$$

Figure 2: Distributivity of composition over aggregation

of Equation 4.

Associativity of composition:

$$(vw)u = v(wu) \tag{1}$$

Definition of projection:

$$\pi_1^2\langle v_1, v_2\rangle = v_1 \text{ and } \pi_2^2\langle v_1, v_2\rangle = v_2 \tag{2}$$

Distributivity of composition over product, right argument:

$$\langle vu, wu\rangle = \langle v, w\rangle \circ u \tag{3}$$

Distributivity of composition over aggregation, right argument:

$$\alpha(v, uw) = \alpha(\alpha(v, w), u) \tag{4}$$

Distributivity of composition over aggregation, left argument :

$$\alpha(hv, w) = h \circ \alpha(v, w) \tag{5}$$

Distributivity of product over aggregation, left argument:

$$\alpha(\langle v_1, v_2\rangle, w) = \langle \alpha(v_1, w), \alpha(v_2, w)\rangle \tag{6}$$

Cancellation law for aggregation:

$$\alpha(v, w)w = v \tag{7}$$

We stress that Equations 2, 3 and 6 can be easily extended to arbitrary products. We therefore also assume the correctness of

$$\pi_i^n\langle v_1, \ldots, v_n\rangle = v_i \tag{2'}$$

and

$$\langle v_1u, \ldots, v_nu\rangle = \langle v_1, \ldots, v_n\rangle \circ u \tag{3'}$$

and

$$\alpha(\langle v_1, \ldots, v_n\rangle, w) = \langle \alpha(v_1, w), \ldots, \alpha(v_n, w)\rangle, \tag{6'}$$

respectively.

2 Subset inclusion

In this section we extend the constructs of the previous section with the mechanisms involved in forming a subset of a given set, given some conditions.

The principle motivation for extending the constructs of [8] with the formation of a subset is given by the observation that the theory developed in [8] lacks the means of relating a variable $u : p \to x$ with the variable $u' : p' \to x$ that is obtained from u by restricting its domain p to a subset $p' \subseteq p$. According to [8], u and u' can only be treated as separate variables, having separate and unrelated properties, which in general is not a desirable feature. To see this, take as an example of u the variable *age class of a person*, i.e., p is the set of persons and x is a set of age classes, such as $[16 - 25]$ and $[26 - 40]$. Then the variable *age class of a female person* is a variable $u' : p' \to x$, which is essentially just u only applied to the subset p' of women. It would be beneficiary if we could describe u' in terms of u, so that u' could inherit some of the properties of u, such as its definition. The first observation is thus that a subset induces variables that are derived from their 'principle form' in the sense that they are essentially the same, only restricted to this subset. It is the objective of this section to describe the mechanisms behind this derivation. As a prelude, note that the inclusion $i : p' \to p$ from p' into p given by $i(e) = e$ gives us the means to satisfactorily define u' as $u \circ i$.

An equally important observation is that the introduction of a subset p' of p gives rise to an asymmetry in the variables that are defined on p and p' respectively, in the sense that a variable $u : p \to x$ is 'equally defined' on p' (viz. through the inclusion i, as we saw above) but the reverse need not be true. Take for instance the variable *number of pregnancies carried to term* defined on the set p' of female persons: this variable makes no sense for the 'full' set p of persons. Thus, subsets, through inclusions, order the availability of variables: through an inclusion more, and more specialized, variables become available.

It is in this sense that the inclusion $i : p' \to p$ can be thought of as the generalization-specialization arrow of the class diagrams of the Unified Modeling Language (UML)[15]. There is however a crucial difference between our treatment of subsets sketched above and the UML generalization-specialization arrow: we only allow the creation of a subset p' from p if it is 'justified' by means of variables that are defined on p. To explain this, note that the set p' of female persons introduced above suggests that we can tell which entity of p is also an entity of p', viz. through the variable *sex*. Thus in the introduction of p' we tacitly used as a selection criterion a variable $v : p \to \{m, f\}$, which assigns a gender (either m or f) to each member of p, and we selected those members e from p for which $v(e) = f$. This suggests that we may write $p' = p_{(v,f)}$ or, more suggestively, $p' = p_{(v=f)}$. It is in this way that we require that each subset is to be constructed from the combination of a set, a variable and a constant, and we will generalize this to the combination of a set and two variables in a minute. Since the inclusion from a subset p' into p is given once p' is defined in this way, we may also write $i = i_{(v,f)}$ (or $i = i_{(v=f)}$).

By identifying an element d from a set x with the constant $d : 1 \to x$, as explained in the Preliminaries, we obtain the general situation for subset construction sketched in Fig. 3. Thus for a variable $v : p \to x$ and an element $d : 1 \to x$, we

$$\begin{tikzcd} & & 1 \arrow[d, "d"] \\ s(v,d) \arrow[r, "i(v,d)"] & p \arrow[r, "v"'] & x \end{tikzcd}$$

Figure 3: The subset induced by v and d

define $s(v,d)$ as the set $\{e \in p \mid v(e) = d(*)\}$. We also let $i(v,d)$ be the function $i : s(v,d) \to p$ defined by $i(e) = e$ for all $e \in s(v,d)$.

For technical reasons mainly, we immediately want to make the situation sketched in Fig. 3 even more general. First we observe that any constant $d : 1 \to x$ can be turned into a constant mapping $d' : p \to x$ (by which we mean that d' maps every element of p to the same element $d \in x$) viz. by composing it with the unique map $1_p : p \to 1$ defined in the Preliminaries. Thus, if we let $d' = d \circ 1_p$ then d' is essentially the same as d: both always yield the element $d \in x$. Note however that now both d' and v are variables defined on p. Thus the subset $s(v,d)$ defined above is identical to the set $\sigma(v,d') = \{e \in p \mid v(e) = d'(e)\}$, since $d'(e) = d(1_p(e)) = d(*)$. The introduction of σ however allows a more general construction in which both arguments are arbitrary variables v and w defined on p and for x, as sketched in Fig. 4. The case $w = d'$ then yields the special case of Fig 3. Thus, in the more

$$\begin{tikzcd} \sigma(v,w) \arrow[r, "\iota(v,w)"] & p \arrow[r, shift left, "v"] \arrow[r, shift right, "w"'] & x \end{tikzcd}$$

Figure 4: The subset induced by v and w

general situation sketched in Fig. 4, we let the *subset induced by v and w*, denoted by $\sigma(v,w)$, be defined as the set $\{e \in p \mid v(e) = w(e)\}$ and, as before, we let the *inclusion induced by v and w*, denoted by $\iota(v,w)$, be the function $\iota : \sigma(v,w) \to p$ defined by $\iota(e) = e$ for all $e \in \sigma(v,w)$. Clearly, $\iota(v,w)$ is an injection.

The definitions of σ and ι are inspired by category theory [3, 18, 13] and the notion of an equalizer in particular, which in a more abstract sense characterizes the subset and the inclusion induced by two functions. We give the definition involved, since it can help prove properties of σ and ι that we state in the next section, but we stress that understanding it is not essential for understanding the development of the theory in this article. Following the terminology of category theory, given objects p and x, and arrows $v, w : p \to x$, an *equalizer of v and w* is an object σ together with an arrow $\iota : \sigma \to p$ for which it holds that $v\iota = w\iota$, and such that for every object σ' and arrow $\iota' : \sigma' \to p$ with $v\iota' = w\iota'$, there is a unique arrow $\mu : \sigma' \to \sigma$ such that $\iota' = \iota\mu$.

$$\begin{tikzcd}
\sigma \arrow[r, "\iota"] & p \arrow[r, shift left, "v"] \arrow[r, shift right, "w"'] & x \\
\sigma' \arrow[u, dashed, "\mu"] \arrow[ur, "\iota'"'] & &
\end{tikzcd}$$

Figure 5: An equalizer of v and w

It can be shown, in the category of sets and functions, that $\sigma(v,w)$ and $\iota(v,w)$ form an equalizer. First, it is easy to see that

$$v\iota(v,w) = w\iota(v,w), \tag{8}$$

by taking an arbitrary element e from $\sigma(v,w)$. We remind the reader that $v\iota(v,w)$ is shorthand for $v \circ \iota(v,w)$ and similarly for $w\iota(v,w)$. Second, assume that $vi' = wi'$ for a function $i' : s' \to p$, i.e., for every $d \in s'$ we have $v(i'(d)) = w(i'(d))$. This means that $i'(d) \in \sigma(v,w)$ for every $d \in s'$. Hence the mapping $u : s' \to \sigma(v,w)$ with $u(d) = i'(d)$ for every $d \in s'$ is well-defined and satisfies $i' = \iota(v,w) \circ u$. Since $\iota(v,w)$ is an inclusion, u is the only such mapping.

Intuitively, a second motivation for introducing the more general subset construction of Fig. 4 by means of an equalizer is that there are situations for which comparing two variables, instead of one variable and one constant, could be useful. Take for instance two variables v and w that both measure the income of a person, but through different means, e.g., through a survey and a register say. Then to investigate the set of persons for which both variables yield the same value requires the equalizer of v and w.

In some situations, it could even be more useful to take the subset of p for which both variables yield a *different* result, which upon first glance is a subset construction that cannot be realized through σ and ι, or through s and i. If however we assume the set $b = \{\mathit{true}, \mathit{false}\}$ of the Boolean values and an inequality function $\neq : x \times x \to b$, then this subset (and the inclusion similarly) can be expressed as $s({\neq}\langle v,w\rangle, \mathit{true})$ where 'true' is a constant $1 \to b$ and ${\neq}\langle v,w\rangle$ is the application of the inequality function to the product of v and w. But then of course, assuming equality $= : x \times x \to b$, a similar construction shows that σ and ι can be expressed in terms of the subset construction of Fig. 3. Hence, with some assumptions, the combination of s and i is equally expressive as σ and ι. Since we made less assumptions when viewing a subset as an equalizer (cf. Fig. 4) we take σ and ι as atomic and consider s and i as useful derivations.

One obvious difference is that for σ and ι the order of their arguments is of no importance, while for s and i it is required that their second argument is a constant. Thus we have

$$\sigma(v,w) = \sigma(w,v) \tag{9}$$

and

$$\iota(v,w) = \iota(w,v). \tag{10}$$

3 More properties of subset inclusion

In this section we continue to investigate properties of the construction of subsets and subset inclusions, especially in the context of the other operators mentioned in the Preliminaries, and explain their relevance for statistical practice.

We first study a number of situations in which expressions containing σ and ι can be simplified. Let p, x, v and w be as before. Consider an injection $u : x \to y$. Then we have

$$\sigma(uv, uw) = \sigma(v, w) \tag{11}$$

and

$$\iota(uv, uw) = \iota(v, w), \tag{12}$$

since $u(v(e)) = u(w(e))$ if and only if $v(e) = w(e)$, for all $e \in p$. Next, consider the situation in which a subset of $\sigma(v, w)$ is induced by v and w, i.e., using the same condition under which $\sigma(v, w)$ is formed. This is the situation in which, e.g., all females from a set of females are selected: this set should of course remain unchanged. More precisely, we have the situation sketched in Fig. 6. It can be

$$\begin{tikzcd}
\sigma(v\iota(v,w), w\iota(v,w)) \arrow[r, "\iota(v\iota(v,w),w\iota(v,w))"] & \sigma(v,w) \arrow[r, "\iota(v,w)"] & p \arrow[r, shift left, "v"] \arrow[r, shift right, "w"'] & x
\end{tikzcd}$$

Figure 6: The subset of a subset, both 'induced' by v and w

shown that

$$\sigma(v\iota(v, w), w\iota(v, w)) = \sigma(v, w) \tag{13}$$

and

$$\iota(v\iota(v, w), w\iota(v, w)) = \mathsf{id}_{\sigma(v,w)}, \tag{14}$$

as expected. The conditions for the third simplification are sketched in Fig. 7. Note that d and e are constants, as defined in the Preliminaries: they are functions

$$\begin{tikzcd}
p \arrow[rr, "v"] \arrow[dr, "1_p"'] & & x \\
& 1 \arrow[ur, shift left, "d"] \arrow[ur, shift right, "e"'] &
\end{tikzcd}$$

Figure 7: Subsets defined by constants d and e

$d, e : 1 \to x$ that are given by members d and e of x. Now suppose that $d \neq e$. Then we have

$$\sigma(v\iota(v, d1_p), e1_p\iota(v, d1_p)) = 0, \tag{15}$$

and

$$\iota(v\iota(v, d1_p), e1_p\iota(v, d1_p)) = 0_{\sigma(v,d1_p)}, \tag{16}$$

indicating that, e.g., selecting all males from a female population results in the empty set. Recall that the right hand side of Equation 16 is the unique map with the empty set as domain and $\sigma(v, d1_p)$ as codomain. Properties 15 and 16 are depicted in Fig. 8. We leave their proofs to the reader. Finally, the only subset of

$$\begin{tikzcd}
\sigma(v\iota(v,d1_p), e1_p\iota(v,d1_p)) \arrow[r, "\iota(v\iota(v,d1_p),e1_p\iota(v,d1_p))"] & \sigma(v,d1_p) \arrow[r, "\iota(v,d1_p)"] & p
\end{tikzcd}$$

Figure 8: The subset of a subset, induced by different constants

the empty set is the empty set itself, so we have

$$\sigma(0_x, 0_x) = 0, \tag{17'}$$

where 0_x is the unique mapping $0 \to x$, and

$$\iota(0_x, 0_x) = 0_0, \tag{18'}$$

with 0_0 the unique mapping $0 \to 0$. More generally, if v is a mapping $p \to q$, then

$$\sigma(v, v) = p, \tag{17}$$

and

$$\iota(v, v) = \mathsf{id}(p). \tag{18}$$

Next consider the situation sketched in Fig. 9, where two subsets are formed using different pairs of variables as conditions. Take for example p as the set of persons, $\sigma(v_1, w_1)$ the subset of women, and $\sigma(v_2, w_2)$ the subset of elderly people (assuming, e.g., suitable conditions on the variables *sex* and *age*, respectively). The question is in what way these subsets are related. Intuitively, the two subsets are

$$\begin{tikzcd}
\sigma(v_1,w_1) \arrow[dr, "\iota(v_1,w_1)"] & & x_1 \\
& p \arrow[ur, shift left, "v_1"] \arrow[ur, shift right, "w_1"'] \arrow[dr, shift left, "v_2"] \arrow[dr, shift right, "w_2"'] & \\
\sigma(v_2,w_2) \arrow[ur, "\iota(v_2,w_2)"'] & & x_2
\end{tikzcd}$$

Figure 9: Two subsets induced by different pairs of variables

related through a third: the subset of elderly women. It should be clear that there are two ways of forming this subset: either by a restriction involving *age* on the subset of woman, or by a restriction involving *sex* on the subset of elderly, the results of which should be equal intuitively. Formally this situation is depicted in

$$\begin{tikzcd}
\sigma(v_2\iota(v_1,w_1), w_2\iota(v_1,w_1)) \arrow[r, "\iota(v_2\iota(v_1,w_1),w_2\iota(v_1,w_1))"] & \sigma(v_1,w_1) \arrow[dr, "\iota(v_1,w_1)"] & \\
& & p \\
\sigma(v_1\iota(v_2,w_2), w_1\iota(v_2,w_2)) \arrow[r, "\iota(v_1\iota(v_2,w_2),w_1\iota(v_2,w_2))"'] & \sigma(v_2,w_2) \arrow[ur, "\iota(v_2,w_2)"'] &
\end{tikzcd}$$

Figure 10: Two more subsets induced by different pairs of variables

Fig. 10. As expected the top-left subset equals the bottom-left, i.e., we have

$$\sigma(v_2\iota(v_1,w_1), w_2\iota(v_1,w_1)) = \sigma(v_1\iota(v_2,w_2), w_1\iota(v_2,w_2)). \tag{19}$$

To see this, note that the left-hand side reduces to

$$\{e \in \sigma(v_1,w_1) \mid v_2\iota(v_1,w_1)(e) = w_2\iota(v_1,w_1)(e)\},$$

which equals

$$\{e \in p \mid v_1(e) = w_1(e) \text{ and } v_2(e) = w_2(e)\},$$

since $e \in \sigma(v_1,w_1)$ if and only if $e \in p$ and $v_1(e) = w_1(e)$, and since $\iota(v_1,w_1)(e) = e$ for all $e \in \sigma(v_1,w_1)$. A similar reduction applies to the right-hand side of Equation 19. It should be equally clear that we then also have

$$\iota(v_1,w_1)\iota(v_2\iota(v_1,w_1), w_2\iota(v_1,w_1)) = \iota(v_2,w_2)\iota(v_1\iota(v_2,w_2), w_1\iota(v_2,w_2)), \tag{20}$$

following the paths of the arrows at the top and at the bottom of Fig. 10, respectively. This can be shown formally using the uniqueness condition of the equalizer construction of Fig. 5, the proof of which we leave to the reader.

The 'commutativity' laws specified in Equations 19 and 20 call for a change of notation: we let $\sigma(v_1{\sim}w_1)$ be an alternate notation for $\sigma(v_1,w_1)$ (and similarly for $\iota(v_1{\sim}v_2)$), we let $\sigma(v_1{\sim}w_1, v_2{\sim}w_2)$ be the left-hand side of Equation 19, and we let $\iota(v_1{\sim}w_1, v_2{\sim}w_2)$ be the left-hand side of Equation 20. This means that Equations 19 and 20 reduce to

$$\sigma(v_1{\sim}w_1, v_2{\sim}w_2) = \sigma(v_2{\sim}w_2, v_1{\sim}w_1) \tag{21}$$

and

$$\iota(v_1{\sim}w_1, v_2{\sim}w_2) = \iota(v_2{\sim}w_2, v_1{\sim}w_1), \tag{22}$$

respectively.

The subsets mentioned in Equation 21 as well as the inclusions mentioned in Equation 22 can also be formed in 'one stroke', viz. through the product operator. More precisely, in the situation of Fig. 9, we have that

$$\sigma(\langle v_1,v_2\rangle{\sim}\langle w_1,w_2\rangle) = \sigma(v_1{\sim}w_1, v_2{\sim}w_2), \tag{23}$$

and

$$\iota(\langle v_1, v_2\rangle\sim\langle w_1, w_2\rangle) = \iota(v_1\sim w_1, v_2\sim w_2), \tag{24}$$

the proof of which we leave to the reader.

The notation introduced just before Equation 21 can be extended to any number of (pairs of) arguments. We give the details by an inductive definition. Assume $v_j, w_j : p \to x_j$ with $1 \le j \le m$. Then we let $\sigma(v_1\sim w_1)$ be $\sigma(v_1, w_1)$ and $\iota(v_1\sim w_1)$ be $\iota(v_1, w_1)$ as before, and for $1 < j \le m$ we let

$$\sigma(v_1\sim w_1, \ldots, v_m\sim w_m) = \sigma(v_m\iota_1^{m-1}, w_m\iota_1^{m-1}), \tag{25}$$

and

$$\iota(v_1\sim w_1, \ldots, v_m\sim w_m) = \iota_1^{m-1}\iota(v_k\iota_1^{m-1}, w_k\iota_1^{m-1}), \tag{26}$$

where ι_1^{m-1} abbreviates $\iota(v_1\sim w_1, \ldots, v_{m-1}\sim w_{m-1})$. Without proof, we claim that Equations 21, 22, 23, and 24 can be extended to any number of arguments. In particular, as far as Equations 21 and 22 are concerned, this means that

$$\sigma(v_1\sim w_1, \ldots, v_m\sim w_m) = \sigma(v_{\phi(1)}\sim w_{\phi(1)}, \ldots, v_{\phi(m)}\sim w_{\phi(m)}), \tag{27}$$

and

$$\iota(v_1\sim w_1, \ldots, v_m\sim w_m) = \iota(v_{\phi(1)}\sim w_{\phi(1)}, \ldots, v_{\phi(m)}\sim w_{\phi(m)}), \tag{28}$$

for any permutation ϕ of $\{1, \ldots, m\}$.

We close this section by the observation that in some circumstances forming a subset in the context of aggregation can be simplified. Consider the situation sketched in Fig. 11, where we assume that x is a monoid and w is inverse-finite. To explain this situation, let p be a set of persons, let q be a set of households,

$$\begin{tikzcd}
 & \sigma(u\sim z) \arrow[r, "\iota(u\sim z)"] & q \arrow[r, shift left, "u"] \arrow[r, shift right, "z"'] \arrow[d, "\alpha(v,w)"] & y \\
\sigma(uw\sim zw) \arrow[r, "\iota(uw\sim zw)"'] & p \arrow[ur, "w"] \arrow[r, "v"'] & x &
\end{tikzcd}$$

Figure 11: Subsets in the context of aggregation I

let w associate a person in p to the household in q he or she is a member of, and let v be the variable *income of a person.* Then, assuming $+$ is the monoid operation of x, $\alpha(v, w)$ is the *income of a household*, summing over the incomes of each member in a household. Now let $\iota(u\sim z)$ select the two-person households in q, where we assume that y is a set of household composition classes and u and z are suitable variables (or a suitable combination of a constant and a variable, as explained earlier). This means that $\iota(uw\sim zw)$ selects all the members of two-person households. The question is: how do $\alpha(v, w)$ and the *income of a two-person household* formed by $\alpha(v\iota(uw\sim zw), w\iota(uw\sim zw))$ relate? Intuitively, they should be equal, as far as two-person households are concerned. We show that indeed we have

$$\alpha(v\iota(uw{\sim}zw), w\iota(uw{\sim}zw))\iota(u{\sim}z) = \alpha(v, w)\iota(u{\sim}z). \tag{29}$$

Let $d \in \sigma(u{\sim}z)$, i.e., we have $u(d) = z(d)$. We show that $\alpha(v, w)$ applied to d equals $\alpha(v\iota(uw{\sim}zw), w\iota(uw{\sim}zw))$ applied to d. Since these expand to

$$\sum_{d=w(e)} v(e) \quad \text{and} \quad \sum_{d=w\iota(uw\sim zw)(e)} v\iota(uw{\sim}zw)(e),$$

respectively, it suffices to show that $e \in \sigma(uw{\sim}zw)$ if and only if $d = w(e)$, since we then have $\iota(uw{\sim}zw)(e) = e$ as required. To show $e \in \sigma(uw{\sim}zw)$ whenever $d = w(e)$ is easy, since $d = w(e)$ implies that $u(w(e)) = z(w(e))$. The other implication is immediate and left to the reader.

It might be tempting to simplify Equation 29 by instead trying to prove the following equation

$$\delta(w\iota(uw{\sim}zw))\iota(u{\sim}z) = \delta(w)\iota(u{\sim}z).$$

This fails however since the codomains of both sides differ.

Finally, a similar but simpler case of forming subsets in the context of aggregation is sketched below (and we assume similar restrictions on x and w). In this

$$\begin{tikzcd}
 & & & q \arrow[dd, "\alpha(v,w)"] \\
 & & 1 \arrow[ur, "d"] & \\
\sigma(w{\sim}d1) \arrow[r, "\iota(w\sim d1)"'] & p \arrow[uurr, "w"] \arrow[ur, "1"'] \arrow[rr, "v"'] & & x
\end{tikzcd}$$

Figure 12: Subsets in the context of aggregation II

case we have

$$\alpha(v, w) \circ d = \alpha(v\iota(w{\sim}d1), w\iota(w{\sim}d1)) \circ d. \tag{30}$$

The proof is left to the reader, but we provide some intuition of the situation above: let p be a set of persons, w be the gender of a person (q is the set of the two sexes) and let v be an arbitrary variable, *income* say. Then the total income of men can be computed by first computing the totals for both men and woman followed by selecting the total for men (the left hand side of Equation 30), or men are selected first from the total popuation p and then their total is computed (the right hand side of Equation 30). Note that while the left hand side of Equation 30 is more concise and easier to understand, its right hand side is probably more computationally efficient.

4 Types and elements

We begin this section by pointing out that there is a crucial difference between the operators recalled in the Preliminaries and the subset operator σ introduced in Section 2. We note that all operators in the Preliminaries that produce a set, viz. the set Fp of finite subsets of p and the general Cartesian product $x_1 \times \cdots \times x_n$ of sets x_i, depend only on sets in their arguments: p and the x_i, respectively. Also, most operators that produce a function, viz. the composition $v \circ w$, the general product $\langle v_1, \ldots, v_n \rangle$, and the constructs $\delta(w)$ and $\gamma(v)$ for defining aggregation, depend only on functions in their arguments — the exceptions are the general projections π_i^n. Nevertheless, in short we have that sets depend on sets only, and functions depend on functions mostly.

The operator $\sigma(v, w)$ that produces a set in contrast relies on functions v and w in its arguments. This means that if we identify a set with a 'type', as we will do in this section, then $\sigma(v, w)$ is a so-called *dependent type* [21, 19]: one that needs additional values, or 'elements' as we will call them, in its construction. In the case of $\sigma(v, w)$, these values v and w are both 'elements' of the functional 'type' x^p, as expressed by the declaration $v, w : p \to x$.

In any case, the system of operators that was considered in [8] made life easy: it allowed for a language that could be defined within the framework of equational logic [16] and for which semantics was immediate ("zap", according to [9]). It made use of the fact that 'types' (or rather: sorts, as they are unfortunately called in the context of algebras) could be constructed independently of values or 'elements'. With the introduction of σ, the resulting system of operators does not have that advantage anymore.

Though there are extensions of initial algebra semantics that allow dependent types [14], how we choose to proceed is to introduce 'types' and 'elements' from scratch, i.e., without the use of some underlying theory other than universal algebra [10]. This is in part because our atomic formulae also deal with modalities (introduced in Definition 3 below) that are not treated by such extensions. Second, and more importantly, we think that the dependency between a typing relation and a congruence, as formulated in [14] Definition 3.5, is insufficient for our purposes and requires the inductive approach taken in this section, motivated by Example 1 below and embodied by Definition 6.

In this section we will give a number of elementary definitions that are used in the next section where they will be put together to form our language. First, both types and elements are certain terms, i.e., sequences of symbols formed through syntactic rules, the universes of which we will define below through mutual recursion, i.e., simultaneously. Then we will define type assignment, i.e., a relation between an element and its type(s). We note that types assigned to elements restrict the use of operators that apply to elements, as for instance the product of two elements requires that they have the same type as a domain. In this way we will limit the universes of types and elements to so-called well-formed types and elements. Similar restrictions will give us so-called well-defined types and elements: these will make sure that, for instance, the application of δ is limited to inverse-finite elements only. Finally, we will define the concept of a congruence relation on types and elements and we show how the typing relation can be extended with it in a natural way: if an element is of a type t, then it should be of any type congruent with t. We note that, in turn, this gives rise to an extension of the universes of well-formed and well-defined types and elements.

We start our development with the simultaneous definition of types and elements, informally at first.

We assume a countable set A of *basic type symbols*. We also assume a countable set B of *basic element symbols* disjoint with A. The sets $T(A,B)$ and $E(A,B)$ of type terms (or just: types) and element terms (or just: elements) respectively, are given informally by the following mutually recursive grammars: a term p is a type if it is produced by the following grammar

$$\begin{array}{rcl} \mathsf{p} & ::= & \mathsf{a} \mid \mathsf{0} \mid \mathsf{1} \mid [\mathsf{p} \to \mathsf{q}] \mid \mathsf{p}_1 \times \cdots \times \mathsf{p}_n \mid \\ & & \mathsf{F}(\mathsf{p}) \mid \boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \end{array}$$

with $\mathsf{a} \in A$, $\mathsf{p}, \mathsf{p}_i, \mathsf{q} \in T(A,B)$, $\mathsf{v}_j, \mathsf{w}_j \in E(A,B)$, $n > 1$ and $m > 0$, with $i \leq n$ and $j \leq m$. A term v is an element if it is produced by

$$\begin{array}{rcl} \mathsf{v} & ::= & \mathsf{b} \mid \mathsf{0}(\mathsf{p}) \mid \mathsf{1}(\mathsf{p}) \mid \mathsf{id}(\mathsf{p}) \mid \mathsf{v} \circ \mathsf{w} \mid \langle \mathsf{u}_1, \ldots, \mathsf{u}_n \rangle \mid \\ & & \boldsymbol{\pi}_i(\mathsf{p}_1, \ldots, \mathsf{p}_n) \mid \boldsymbol{\gamma}(\mathsf{v}) \mid \boldsymbol{\delta}(\mathsf{w}) \mid \boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \end{array}$$

with $\mathsf{b} \in B$, $\mathsf{p}, \mathsf{p}_i \in T(A,B)$, $\mathsf{v}, \mathsf{w}, \mathsf{v}_j, \mathsf{w}_j, \mathsf{u}_i \in E(A,B)$, $n > 1$ and $m > 0$, with $i \leq n$ and $j \leq m$.

When A and B are clear from the context, we abbreviate $T(A,B)$ by T and $E(A,B)$ by E.

In addition, we let the set $T(A)$ of elementary types be as follows: a term p is an elementary type if it is produced by

$$\mathsf{p} \quad ::= \quad \mathsf{a} \mid \mathsf{1} \mid [\mathsf{p} \to \mathsf{q}] \mid \mathsf{p}_1 \times \cdots \times \mathsf{p}_n \mid \mathsf{F}(\mathsf{p})$$

with $\mathsf{a} \in A$ and $\mathsf{p}, \mathsf{q}, \mathsf{p}_i \in T(A)$. Note that $\mathsf{0}$ is not an elementary type and that an elementary type does not depend on any elements.

The types $\mathsf{0}$ and $\mathsf{1}$ are called *zero* and *one*, respectively. The type $[\mathsf{p} \to \mathsf{q}]$ is the *function type induced by* p *and* q. The type $\mathsf{p}_1 \times \cdots \times \mathsf{p}_n$ is the *product type induced by* p_i. We stress that the product type is a family of constructs (one for every $n > 1$) and that the ellipsis $(\cdots)$ is not part of the type, but rather part of the metalanguage that is used to define the grammar. Thus $\mathsf{p} \times \mathsf{q}$ and $\mathsf{p} \times \mathsf{q} \times \mathsf{r}$ are types (provided p, q and r are types) and $\mathsf{p} \times \cdots \times \mathsf{q}$ is not a type. The type $\mathsf{F}(\mathsf{p})$ is the *finite power type induced by* p. For elements $\mathsf{v}_j, \mathsf{w}_j \in E$, the type $\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m)$ is the *subtype induced by* v_j *and* w_j. Again, this is a family of constructs (one for each $m > 0$) and the ellipsis is not part of the type.

Note that $\mathsf{0}(\mathsf{p})$ and $\mathsf{1}(\mathsf{p})$ define distinct elements for every $\mathsf{p} \in T$; each is called *zero* and *one*, respectively. When p is clear from the context (and when there is no danger of confusing them with their type counterparts) $\mathsf{0}(\mathsf{p})$ is sometimes abbreviated by $\mathsf{0}$ and $\mathsf{1}(\mathsf{p})$ is sometimes abbreviated by $\mathsf{1}$. The element $\mathsf{id}(\mathsf{p})$ is the *identity on* p and we have such an element for every type p. We sometimes abbreviate $\mathsf{id}(\mathsf{p})$ by id. The names of the rest of the elements follow the names of their functional counterparts defined in Sections 1 and 2. So the element $\mathsf{v} \circ \mathsf{w}$ is called the *composition of* v *and* w and is sometimes abbreviated by vw. The ellipsis in $\langle \mathsf{u}_1, \ldots, \mathsf{u}_n \rangle$ is not part of the element, but part of the metalanguage. So $\langle \mathsf{u}, \mathsf{w} \rangle$ and $\langle \mathsf{u}, \mathsf{w}, \mathsf{v} \rangle$ are elements (provided u, w and v are elements) and $\langle \mathsf{u}, \ldots, \mathsf{w} \rangle$ is not. Also, for every $n > 1$ with $1 \leq i \leq n$ and every $\mathsf{p}_i \in T$, each $\boldsymbol{\pi}_i(\mathsf{p}_1, \ldots, \mathsf{p}_n)$ is a distinct element. When n and p_i are clear from the context, we abbreviate projection by $\boldsymbol{\pi}_i$. We use $\boldsymbol{\alpha}(\mathsf{v}, \mathsf{w})$ as an alternative notation for $\boldsymbol{\gamma}(\mathsf{v}) \circ \boldsymbol{\delta}(\mathsf{w})$. We note that $\boldsymbol{\iota}(\mathsf{v}_1 \sim \mathsf{w}_1, \ldots, \mathsf{v}_m \sim \mathsf{w}_m)$ is a family of constructs, one for each $m > 0$.

Formally now:

Definition 1. *Given a countable set A of basic type symbols and a countable set B of basic element symbols disjoint with A, the sets $T(A, B)$ and $E(A, B)$ of* types *and* elements *respectively are defined as*

$$T(A, B) = \bigcup_{k \geq 0} T_k \text{ and } E(A, B) = \bigcup_{k \geq 0} E_k,$$

where T_k and E_k are defined recursively as

$$\begin{aligned}
T_0 &= A \cup \{\mathsf{0}, \mathsf{1}\}, \\
E_0 &= B, \text{ and} \\
T_k &= T_{k-1} \cup \{[\mathsf{p} \to \mathsf{q}],\ \mathsf{p}_1 \times \cdots \times \mathsf{p}_n, \\
&\qquad \mathsf{F}(\mathsf{p}),\ \boldsymbol{\sigma}(\mathsf{v}_1 \sim \mathsf{w}_1, \ldots, \mathsf{v}_m \sim \mathsf{w}_m) \mid \\
&\qquad \mathsf{p}, \mathsf{q}, \mathsf{p}_i \in T_{k-1},\ \mathsf{v}_j, \mathsf{w}_j \in E_{k-1},\ n > 1 \text{ and } m > 0, \\
&\qquad \text{with } i \leq n \text{ and } j \leq m\}, \text{ and} \\
E_k &= E_{k-1} \cup \{\mathsf{0}(\mathsf{p}),\ \mathsf{1}(\mathsf{p}),\ \mathsf{id}(\mathsf{p}),\ \mathsf{v} \circ \mathsf{w},\ \langle \mathsf{u}_1, \ldots, \mathsf{u}_n \rangle, \\
&\qquad \boldsymbol{\pi}_i(\mathsf{p}_1, \ldots, \mathsf{p}_n),\ \boldsymbol{\gamma}(\mathsf{v}),\ \boldsymbol{\delta}(\mathsf{w}),\ \boldsymbol{\iota}(\mathsf{v}_1 \sim \mathsf{w}_1, \ldots, \mathsf{v}_m \sim \mathsf{w}_m) \mid \\
&\qquad \mathsf{p}, \mathsf{p}_i \in T_{k-1},\ \mathsf{v}, \mathsf{w}, \mathsf{v}_j, \mathsf{w}_j, \mathsf{u}_i \in E_{k-1},\ n > 1 \text{ and } m > 0, \\
&\qquad \text{with } i \leq n \text{ and } j \leq m\},
\end{aligned}$$

for all $k > 0$. We let $TE(A, B)$ be the set of all types and elements, i.e., $TE(A, B) = T(A, B) \cup E(A, B)$.

The set $T(A)$ of elementary types *is defined as*

$$T(A) = \bigcup_{k \geq 0} T'_k,$$

with T'_k defined recursively as

$$\begin{aligned}
T'_0 &= A \cup \{\mathsf{1}\}, \text{ and} \\
T'_k &= T'_{k-1} \cup \{[\mathsf{p} \to \mathsf{q}],\ \mathsf{p}_1 \times \cdots \times \mathsf{p}_n, \\
&\qquad \mathsf{F}(\mathsf{p}) \mid \mathsf{p}, \mathsf{q}, \mathsf{p}_i \in T'_{k-1} \text{ and } n > 1\}
\end{aligned}$$

for all $k > 0$.

Note that Definition 1 makes sense since $T_{k-1} \subseteq T_k$, $E_{k-1} \subseteq E_k$ and $T'_{k-1} \subseteq T'_k$ for all $k > 0$. Note also that $T(A) \subseteq T(A,B)$.

Given a term (an element or a type) $\mathsf{y} \in TE(A,B)$, the notions of a *subterm of* y and a *proper subterm of* y (i.e., a subterm of y not equal to y) are defined in the usual way, i.e., by induction on the structure of y according to Definition 1 above. We denote by $\mathsf{y}' \leq \mathsf{y}$ that y' is a subterm of y, and by $\mathsf{y}' < \mathsf{y}$ that y' is a proper subterm of y. Note that a type can be a subterm of an element and vice versa, due to, e.g., the projection construct and the subtype construct, respectively.

We stress that the elements in Definition 1 are untyped. This means that we do not yet have a relation between an element and a type that prevents constructing elements that make no sense. In other words, Definition 1 introduces elements and types that are intuitively incorrect. An example is the element $\mathsf{0}(\mathsf{1}) \circ \mathsf{1}(\mathsf{1})$: intuitively, its subterms $\mathsf{0}(\mathsf{1})$ and $\mathsf{1}(\mathsf{1})$ represent the (unique) functions of type $[\mathsf{0} \to \mathsf{1}]$ and $[\mathsf{1} \to \mathsf{1}]$ respectively. However, the domain $\mathsf{0}$ of the first is incompatible with the codomain $\mathsf{1}$ of the second, which is required for a correct composition. We will correct this in a minute, when we introduce well-formed types and elements. Also, we note that Definition 1 introduces function types that represent the empty set, such as the type $[\mathsf{p} \to \mathsf{0}]$: if p represents a nonempty set, then no element should have type $[\mathsf{p} \to \mathsf{0}]$. More generally, since $\sigma(v{\sim}w)$ might yield the empty set, we have to be careful about assigning an element to a type of the form $[\mathsf{p} \to \boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{w})]$. The elementary types are 'safe' in this respect: if no basic type represents the empty set, then no elementary type represents the empty set.

The general concept of a typing relation that assigns one or more types to an element is given next. Elements that receive a type in this way are called well-formed, and types that are built from well-formed elements are well-formed. For reasons explained earlier in this section, we base a typing relation on an equivalence relation, such that elements are assigned to equivalent types, and equivalent elements receive identical types. Finally, we assume that basic element symbols are assigned a 'safe' type, through a given mapping.

Definition 2. *Let A and B be a set of basic type symbols and a set of basic element symbols respectively. Let $t : B \to T(A,B)$ be a mapping such that $t(\mathsf{b}) = [\mathsf{p} \to \mathsf{q}]$ with $\mathsf{q} \in T(A)$. Let $\equiv$ be an equivalence relation on types and elements; more specifically, let $\equiv \subseteq T(A,B)^2 \cup E(A,B)^2$. The* typing relation induced by t and $\equiv$, *denoted by $::_{t,\equiv} \subseteq E(A,B) \times T(A,B)$, is defined as the smallest relation such that the following conditions hold:*

(i) *if* $t(\mathsf{b}) = \mathsf{s}$ *and* s *is well-formed, then* $\mathsf{b} :: \mathsf{s}$,

(ii) *if* p *is well-formed, then* $\mathsf{0}(\mathsf{p}) :: [\mathsf{0} \to \mathsf{p}]$, $\mathsf{1}(\mathsf{p}) :: [\mathsf{p} \to \mathsf{1}]$ *and* $\mathsf{id}(\mathsf{p}) :: [\mathsf{p} \to \mathsf{p}]$,

(iii) *if* $\mathsf{v} :: [\mathsf{q} \to \mathsf{r}]$ *and* $\mathsf{w} :: [\mathsf{p} \to \mathsf{q}]$, *then* $\mathsf{v} \circ \mathsf{w} :: [\mathsf{p} \to \mathsf{r}]$,

(iv) *if* $\mathsf{u}_i :: [\mathsf{p} \to \mathsf{p}_i]$, $1 \leq i \leq n$, *then* $\langle \mathsf{u}_1, \ldots, \mathsf{u}_n \rangle :: [\mathsf{p} \to (\mathsf{p}_1 \times \cdots \times \mathsf{p}_n)]$,

(v) *if* p_i *is well-formed for every i with $1 \leq i \leq n$, then* $\boldsymbol{\pi}_i(\mathsf{p}_1, \ldots, \mathsf{p}_n) :: [(\mathsf{p}_1 \times \cdots \times \mathsf{p}_n) \to \mathsf{p}_i]$,

(vi) *if* $\mathsf{v} :: [\mathsf{p} \to \mathsf{q}]$, *then* $\boldsymbol{\gamma}(\mathsf{v}) :: [\mathsf{F}(\mathsf{p}) \to \mathsf{q}]$,

(vii) *if* $\mathsf{w} :: [\mathsf{p} \to \mathsf{r}]$, *then* $\boldsymbol{\delta}(\mathsf{w}) :: [\mathsf{r} \to \mathsf{F}(\mathsf{p})]$,

(viii) *if* $\mathsf{v}_j, \mathsf{w}_j :: [\mathsf{p} \to \mathsf{q}_j]$, $1 \leq j \leq m$,
then $\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) :: [\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \to \mathsf{p}]$,

(ix) *if* $\mathsf{v} :: \mathsf{s}$, $\mathsf{s} \equiv \mathsf{s}'$ *and* s' *is well-formed, then* $\mathsf{v} :: \mathsf{s}'$, *and*

(x) *if* $\mathsf{v} :: \mathsf{s}$, $\mathsf{w} \equiv \mathsf{v}$ *and* w *is well-formed, then* $\mathsf{w} :: \mathsf{s}$,

where in (i), $\mathsf{b} :: \mathsf{s}$ *is a shorthand notation for* $(\mathsf{b}, \mathsf{s}) \in ::_{t,\equiv}$ *and similarly for (ii) — (x), and where the set of* well-formed *types is the smallest set such that*

(xi) *every* $\mathsf{a} \in A$ *is well-formed, and* $\mathsf{0}$ *and* $\mathsf{1}$ *are well-formed,*

(xii) *if* p *and* q *are well-formed, then* $[\mathsf{p} \to \mathsf{q}]$ *is well-formed,*

(xiii) *if* p_i *is well-formed for every* i, $1 \leq i \leq n$, *then* $\mathsf{p}_1 \times \cdots \times \mathsf{p}_n$ *is well-formed,*

(xiv) *if* p *is well-formed, then* $\mathsf{F}(\mathsf{p})$ *is well-formed, and*

(xv) *if* $\mathsf{v}_j, \mathsf{w}_j :: [\mathsf{p} \to \mathsf{q}_j]$, $1 \leq j \leq m$, *then* $\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m)$ *is well-formed.*

An element v *is* well-formed *if* $\mathsf{v} :: \mathsf{s}$ *for some* s. *The sets of well-formed types and well-formed elements are denoted by* $T(A, B, t, \equiv)$ *and* $E(A, B, t, \equiv)$, *respectively. We let* $TE(A, B, t, \equiv) = T(A, B, t, \equiv) \cup E(A, B, t, \equiv)$.

It follows from Definition 2 that if $\mathsf{v} :: \mathsf{s}$, then s is well-formed. Moreover, if $[\mathsf{p} \to \mathsf{q}] \equiv \mathsf{s}$ implies that $\mathsf{s} = [\mathsf{p}' \to \mathsf{q}']$, then we have that $\mathsf{v} :: \mathsf{s}$ implies that $\mathsf{s} = [\mathsf{p} \to \mathsf{q}]$ for some well-formed types p and q. Note that all elementary types are well-formed, i.e., we have $T(A) \subseteq T(A, B, t, \equiv)$. Also note that if $\equiv_1 \subseteq \equiv_2$, then $T(A, B, t, \equiv_1) \subseteq T(A, B, t, \equiv_2)$ and $E(A, B, t, \equiv_1) \subseteq E(A, B, t, \equiv_2)$. Moreover, we have the following properties, which we will need in the next section.

Proposition 1. *For* $k \geq 0$, *let* $\equiv_k$ *be equivalence relations with* $\equiv_k \subseteq \equiv_{k+1}$. *Then*

$$T(A, B, t, \bigcup_{k \geq 0} \equiv_k) = \bigcup_{k \geq 0} T(A, B, t, \equiv_k)$$

and

$$E(A, B, t, \bigcup_{k \geq 0} \equiv_k) = \bigcup_{k \geq 0} E(A, B, t, \equiv_k).$$

Proof. We only show the second; the first is analogous. To see that

$$\bigcup_{k \geq 0} E(A, B, t, \equiv_k) \subseteq E(A, B, t, \bigcup_{k \geq 0} \equiv_k)$$

let $\mathsf{v} \in \bigcup_{k \geq 0} E(A, B, t, \equiv_k)$. Then $\mathsf{v} \in E(A, B, t, \equiv_k)$ for some $k \geq 0$. Hence $\mathsf{v} \in E(A, B, t, \bigcup_{k \geq 0} \equiv_k)$ since $\equiv_k \subseteq \bigcup_{k \geq 0} \equiv_k$. To show the reverse, let $\equiv$ denote $\bigcup_{k \geq 0} \equiv_k$. Let $\mathsf{v} \in E(A, B, t, \equiv)$, i.e., let $\mathsf{v} :: \mathsf{s}$. To derive $\mathsf{v} :: \mathsf{s}$ from the cases (i) — (x), it is clear that (ix) and (x) cannot be used an infinite number of times, i.e., there is a finite list of equations $\mathsf{w}_i \equiv \mathsf{v}_i$ and a finite list of equations $\mathsf{s}_j \equiv \mathsf{s}'_j$ from which we can conclude that $\mathsf{v} :: \mathsf{s}$. But then, since $\equiv_k \subseteq \equiv_{k+1}$, there is a k such that $\mathsf{w}_i \equiv_k \mathsf{v}_i$ and $\mathsf{s}_j \equiv_k \mathsf{s}'_j$. That means that $\mathsf{v} \in E(A, B, t, \equiv_k) \subseteq \bigcup_{k \geq 0} E(A, B, t, \equiv_k)$. □

We stress that Definition 2 can be rewritten into the more constructive form of Definition 1 by indexing both the sets of well-formed types T_k as well as the typing relation; the latter by letting $:: = \bigcup_{k \geq 0} ::_k$, $::_0 = \emptyset$, and $::_k$ be defined cf. conditions (i) — (x). We think, however, that Definition 2 is more readable as it is.

The intuitive meaning of a well-formed term is that all of the operators it consists of are correctly applied with respect to the typing relation, in particular composition, product and the formation of subtypes and inclusions. Note that every type $\mathsf{a} \in A$ is well-formed, that $\mathsf{0}$ and $\mathsf{1}$ are well-formed, that an element $\mathsf{b} \in B$ is well-formed if $\mathsf{b} :: [\mathsf{p} \to \mathsf{q}]$ with well-formed p and elementary type q, and that $\mathsf{0}(\mathsf{p})$, $\mathsf{1}(\mathsf{p})$ and $\boldsymbol{\pi}_i(\mathsf{p}_1, \ldots, \mathsf{p}_n)$ are well-formed whenever p and p_i are well-formed, respectively.

Finally, note that Definition 2 excludes type assignments such as $\mathsf{u} :: [\boldsymbol{\sigma}(\mathsf{u}{\sim}\mathsf{u}) \to \mathsf{p}]$, at least if $[\boldsymbol{\sigma}(\mathsf{u}{\sim}\mathsf{u}) \to \mathsf{p}]$ is the only type that is associated with u. In general, type assignments such as $\mathsf{1}(\mathsf{0}) :: [\boldsymbol{\sigma}(\mathsf{1}(\mathsf{0}){\sim}\mathsf{1}(\mathsf{0})) \to \mathsf{1}]$ might be allowed though, viz. through condition (ix), provided that $[\mathsf{0} \to \mathsf{1}] \equiv [\boldsymbol{\sigma}(\mathsf{1}(\mathsf{0}){\sim}\mathsf{1}(\mathsf{0})) \to \mathsf{1}]$, since we already have $\mathsf{1}(\mathsf{0}) :: [\mathsf{0} \to \mathsf{1}]$ by condition (ii).

To understand the dynamics of Definition 2, in particular with respect to the given equivalence relation, consider the following example.

Example 1. Let $A = \{\mathsf{a}, \mathsf{a}'\}$ and let $B = \{\mathsf{b}\}$. Let $t(\mathsf{b}) = [\mathsf{a} \to \mathsf{a}']$. Consider the lists of types and elements below.

$\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b})$	$\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b})$
$\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}))$	$\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}))$
$\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b})))$	$\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b})))$
$\cdots$	$\cdots$

Note that every type is a member of $T(A,B)$ and every element is a member of $E(A,B)$. Also note that every element in the right column is a subterm of the type one row lower in the left column. To help the intuition of the reader, part of the situation is depicted below.

$$\begin{tikzcd} \boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}) \arrow[r, "\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b})"'] & \mathsf{a} \arrow[r, "\mathsf{b}"'] & \mathsf{a}' \end{tikzcd}$$

Now let $\equiv_I$ be the identity on $TE(A,B)$. According to conditions (xi) and (xii) of Definition 2 respectively, a, a' and $[\mathsf{a} \to \mathsf{a}']$ are well-formed, and so $\mathsf{b} :: [\mathsf{a} \to \mathsf{a}']$ by condition (i). Hence by condition (xv), $\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b})$ is well-formed, and $\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}) :: [\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}) \to \mathsf{a}]$ by condition (viii). Thus we have that $\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}) :: [\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}) \to \mathsf{a}']$ by condition (iii). However, the prerequisites of (xv) and (viii) respectively do not apply to the type $\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}))$ and the element $\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}\boldsymbol{\iota}(\mathsf{b}{\sim}\mathsf{b}))$ in the second row of the table above: note that it requires that the types $[\mathsf{a} \to \mathsf{a}']$ and $[\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}) \to \mathsf{a}']$ be equivalent. However, conditions (ix) and (x) are impotent in this respect. This means that only the first row above constitutes of a well-formed type (i.e., a member of $T(A,B,t,\equiv_I)$) and a well-formed element (a member of $E(A,B,t,\equiv_I)$). Note that if we let $\equiv\, = \,\equiv_I \cup \{(\boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}), \mathsf{a}), (\mathsf{a}, \boldsymbol{\sigma}(\mathsf{b}{\sim}\mathsf{b}))\}$, then the types and elements in both the first and the second row are well-formed with respect to $T(A, B, t, \equiv)$ and $E(A, B, t, \equiv)$ respectively, but the type and element in the third row are not.

We need to further restrict well-formed elements and types, in particular because the dimensional structure $\boldsymbol{\delta}(\mathsf{w})$ induced by w is meaningful only if w can be interpreted as a inverse-finite function (see [8], or the Preliminaries). Also, for a class parameter $\boldsymbol{\gamma}(\mathsf{v})$, both $\boldsymbol{\gamma}(\mathsf{v})$ and v should be interpreted as functions that have a monoid as a codomain (again, see [8], or the Preliminaries). Well-formed elements and types thus formed will be called well-defined. We give the definitions.

Definition 3. *Let A and B be a set of basic type symbols and a set of basic element symbols respectively. Let t and $\equiv$ be a mapping and an equivalence relation respectively, cf. Definition 2. Let M be the set $\{\mathsf{mon}, \mathsf{invfin}, \mathsf{inj}, \mathsf{hom}\}$ of the* modalities *named* monoid, inverse-finite, injection *and* homomorphism*, respectively. Let $u \subseteq (A \times \{\mathsf{mon}\}) \cup (B \times \{\mathsf{invfin}, \mathsf{inj}, \mathsf{hom}\})$ be a relation. The* modality denomination *induced by u, denoted by $\rhd_u \subseteq TE(A, B) \times M$, is defined as the smallest relation such that the following conditions hold:*

(i) *if* $(\mathsf{a}, \mathsf{mon}) \in u$, $\mathsf{a} \in A$, *then* $\mathsf{a} \rhd \mathsf{mon}$,

(ii) *if* $(\mathsf{b}, \mathsf{m}) \in u$, $\mathsf{b} \in B$, *then* $\mathsf{b} \rhd \mathsf{m}$,

(iii) $\mathsf{0}(\mathsf{p}) \rhd \mathsf{inj}$ *and* $\mathsf{id}(\mathsf{p}) \rhd \mathsf{inj}$,

(iv) *if* $\mathsf{v} \rhd \mathsf{inj}$, *then* $\mathsf{v} \rhd \mathsf{invfin}$,

(v) *if* $\mathsf{p}_i \rhd \mathsf{mon}$, $1 \leq i \leq n$, *then* $\mathsf{p}_1 \times \cdots \times \mathsf{p}_n \rhd \mathsf{mon}$,

(vi) *if* $\mathsf{v}, \mathsf{w} \rhd \mathsf{m}$, *then* $\mathsf{v} \circ \mathsf{w} \rhd \mathsf{m}$,

(vii) *if* $\mathsf{u}_i \rhd \mathsf{m}$, $1 \leq i \leq n$, *then* $\langle \mathsf{u}_1, \ldots, \mathsf{u}_n \rangle \rhd \mathsf{m}$,

(viii) $\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \rhd \mathsf{inj}$,

(ix) *if* $\mathsf{u} :: [1 \to \mathsf{p}]$ *then* $\mathsf{u} \rhd \mathsf{inj}$, *and*

(x) *if* $\mathsf{y} \rhd \mathsf{m}$ *and* $\mathsf{y} \equiv \mathsf{y}'$, *then* $\mathsf{y}' \rhd \mathsf{m}$,

where in (i), $\mathsf{a} \rhd \mathsf{mon}$ *is shorthand for* $(\mathsf{a}, \mathsf{mon}) \in \rhd_{t,\equiv,u}$ *and similarly for (ii) — (x); where in (ii), (vi) and (vii),* $\mathsf{m} \in \{\mathsf{invfin}, \mathsf{inj}, \mathsf{hom}\}$*; and where in (x),* $\mathsf{m} \in \{\mathsf{mon}, \mathsf{invfin}, \mathsf{inj}, \mathsf{hom}\}$.

A type p *if* well-defined*, if* p *is well-formed and each proper subterm of* p *is well-defined. An element* u *is well-defined, if* u *is well-formed, each proper subterm of* u *is well-defined, and the following conditions hold:*

(xi) *if* $\mathsf{u} \rhd \mathsf{hom}$ *and* $\mathsf{u} :: [\mathsf{p} \to \mathsf{q}]$, *then* $\mathsf{p} \rhd \mathsf{mon}$ *and* $\mathsf{q} \rhd \mathsf{mon}$,

(xii) *if* $\mathsf{u} = \boldsymbol{\delta}(\mathsf{w})$, *then* $\mathsf{w} \rhd \mathsf{invfin}$, *and*

(xiii) *if* $\mathsf{u} = \boldsymbol{\gamma}(\mathsf{v})$ *and* $\mathsf{v} :: [\mathsf{p} \to \mathsf{q}]$, *then* $\mathsf{q} \rhd \mathsf{mon}$.

The set of well-defined types is denoted $T(A, B, t, \equiv, u)$ and the set of well-defined elements is denoted $E(A, B, t, \equiv, u)$; their union is $TE(A, B, t, \equiv, u)$.

Definition 3 syntactically embodies some knowledge we have about monoids, inverse-finite functions, injections and homomorphisms, respectively. For instance, condition (iv) expresses that every injection is inverse-finite, and condition (v) expresses that monoids are closed under products of types. Conditions (vi) and (vii) express that injections, inverse-finite functions and homomorphisms are closed under compositions and products, respectively.

Note that if $\boldsymbol{\gamma}(\mathsf{v})$ is well-defined, then we can, e.g., conclude that $\boldsymbol{\gamma}(\mathsf{v}) :: [\mathsf{F}(\mathsf{p}) \to \mathsf{q}]$ with $\mathsf{q} \rhd \mathsf{mon}$, as desired.

It is possible to extend the modalities of Definition 3 with notions that may have other significance to statistics, or to statistical metadata in particular. For instance, we could introduce modalities corresponding to the notion of an *object type* [8] and an *object type relation* [8], and add the condition to Definition 3 that an object type relation v (i.e., any element v that is denominated as an object type relation) with type $[\mathsf{p} \to \mathsf{q}]$ is well-defined if both p and q are denominated as an object type. Also, given a meaningful denomination corresponding to the notion of a statistical variable, we could add the condition to Definition 3 that a type $\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{w})$ is an object type if v and w are variables. Though these additional notions could prove useful (and we have introduced the mechanisms for formalizing them in Definition 3) we stress that they play no significant role in the theory developed in this article. That changes if we added the notion of a classification system as a type modality, since we then would have to add the rules of a Boolean algebra (see [7, 8]) to our notion of a congruence of elements and types (to be defined later in the article).

Also, it may seem odd that we rather superficially introduced the notion of a monoid, since we left out the corresponding monoid operation and unit. Again, these could be added to our grammar of elements, and then we could add the monoid laws to our notion of congruence (defined later). However, as far as structural metadata is concerned, that would become significant once we also added the notion of *sum* (called *row-wise combination* in [8]) because only the congruence laws involving sum need the monoid operation. We stress again however that the mechanisms to formally add the notion of a monoid have been introduced here (or will be introduced shortly), but we decided to leave them them out of the theory for the sake of simplicity and brevity.

With respect to well-defined types and elements, we have a corollary similar to Proposition 1.

Corollary 1. *Let, for $k \geq 0$, $\equiv_k$ be equivalence relations with $\equiv_k \subseteq \equiv_{k+1}$. Then*

$$T(A, B, t, \bigcup_{k\geq 0} \equiv_k, u) = \bigcup_{k\geq 0} T(A, B, t, \equiv_k, u)$$

and

$$E(A, B, t, \bigcup_{k\geq 0} \equiv_k, u) = \bigcup_{k\geq 0} E(A, B, t, \equiv_k, u).$$

Proof. Analogous to the proof of Proposition 1. □

The last notion that we define in this section is the general notion of a congruence on well-defined types and elements, given a typing relation and a denomination. Intuitively, a congruence relates terms that should be considered equal; in the next section we give the (business) rules that motivate on what grounds terms are equal and we define one particular congruence relation with them.

Definition 4. *Given sets of basic type and basic element symbols A and B; a mapping and an equivalence relation t and $\equiv$ respectively, cf. Definition 2; and a relation u cf. Definition 3; a relation $\cong\ \subseteq T(A,B,t,\equiv,u)^2 \cup E(A,B,t,\equiv,u)^2$ is a* congruence, *if $\cong$ is an equivalence relation that is closed by the constructs of Definition 1, i.e., for all well-defined terms x, y and z we have*

(i) $\mathsf{x} \cong \mathsf{x}$,

(ii) if $\mathsf{x} \cong \mathsf{y}$, *then* $\mathsf{y} \cong \mathsf{x}$,

(iii) if $\mathsf{x} \cong \mathsf{y}$ *and* $\mathsf{y} \cong \mathsf{z}$, *then* $\mathsf{x} \cong \mathsf{z}$,

and we have

(iv) if $\mathsf{p} \cong \mathsf{p}'$ *and* $\mathsf{q} \cong \mathsf{q}'$, *then* $[\mathsf{p} \to \mathsf{q}] \cong [\mathsf{p}' \to \mathsf{q}']$,

(v) if $\mathsf{p}_i \cong \mathsf{p}'_i$ *for all* $1 \le i \le n$, *then* $\mathsf{p}_1 \times \cdots \times \mathsf{p}_n \cong \mathsf{p}'_1 \times \cdots \times \mathsf{p}'_n$,

(vi) if $\mathsf{p} \cong \mathsf{p}'$, *then* $\mathsf{F}(\mathsf{p}) \cong \mathsf{F}(\mathsf{p}')$,

(vii) if $\mathsf{v}_j \cong \mathsf{v}'_j$ *and* $\mathsf{w}_j \cong \mathsf{w}'_j$ *for all* $1 \le j \le m$, *then*
$\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \cong \boldsymbol{\sigma}(\mathsf{v}'_1{\sim}\mathsf{w}'_1, \ldots, \mathsf{v}'_m{\sim}\mathsf{w}'_m)$,

(viii) if $\mathsf{p} \cong \mathsf{p}'$, *then* $\mathsf{0}(\mathsf{p}) \cong \mathsf{0}(\mathsf{p}')$, $\mathsf{1}(\mathsf{p}) \cong \mathsf{1}(\mathsf{p}')$ *and* $\mathsf{id}(\mathsf{p}) \cong \mathsf{id}(\mathsf{p}')$,

(ix) if $\mathsf{v} \cong \mathsf{v}'$ *and* $\mathsf{w} \cong \mathsf{w}'$, *then* $\mathsf{v} \circ \mathsf{w} \cong \mathsf{v}' \circ \mathsf{w}'$,

(x) if $\mathsf{u}_i \cong \mathsf{u}'_i$ *for all* $1 \le i \le n$, *then* $\langle \mathsf{u}_1, \ldots, \mathsf{u}_n \rangle \cong \langle \mathsf{u}'_1, \ldots, \mathsf{u}'_n \rangle$,

(xi) if $\mathsf{p}_i \cong \mathsf{p}'_i$ *for all* $1 \le i \le n$, *then* $\boldsymbol{\pi}_i(\mathsf{p}_1, \ldots, \mathsf{p}_n) \cong \boldsymbol{\pi}_i(\mathsf{p}'_1, \ldots, \mathsf{p}'_n)$,

(xii) if $\mathsf{w} \cong \mathsf{w}'$, *then* $\boldsymbol{\delta}(\mathsf{w}) \cong \boldsymbol{\delta}(\mathsf{w}')$,

(xiii) if $\mathsf{v} \cong \mathsf{v}'$, *then* $\boldsymbol{\gamma}(\mathsf{v}) \cong \boldsymbol{\gamma}(\mathsf{v}')$, *and*

(xiv) if $\mathsf{v}_j \cong \mathsf{v}'_j$ *and* $\mathsf{w}_j \cong \mathsf{w}'_j$ *for all* $1 \le j \le m$, *then*
$\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \cong \boldsymbol{\iota}(\mathsf{v}'_1{\sim}\mathsf{w}'_1, \ldots, \mathsf{v}'_m{\sim}\mathsf{w}'_m)$,

where it is understood that the left and the right hand sides of equations (iv) – (xiv) yield well-defined terms. We say that $\cong$ is a congruence on $T(A,B,t,\equiv,u)^2 \cup E(A,B,t,\equiv,u)^2$.

Note that we thus require that the left and the right hand sides of two terms that are congruent are well-defined. Also note that we don't require that the left and the right hand sides of a congruence, in the case both are elements, have a common type according to the typing relation $::_{t,\equiv}$. In fact, the congruences we will establish in the next section will not have this property in general. Instead, they will satisfy the following: if $\mathsf{v} \cong \mathsf{w}$, $\mathsf{v} :: \mathsf{s}$ and $\mathsf{w} :: \mathsf{s}'$, then $\mathsf{s} \cong \mathsf{s}'$.

In the following section, Definition 4 is used to construct a particular congruence, equating pairs of terms that are presumed equivalent, cf. the equations of Sections 1, 2 and 3. For instance, we want to include into the congruence we have in mind, all well-defined terms of the form $\mathsf{id}\mathsf{v} \cong \mathsf{v}$ and $\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{v}) \cong \mathsf{id}$.

5 A formal language for structural metadata

In this section we incorporate into our language the rules discovered in Sections 2 and 3. We define a family of congruences, based on a family of notions of well-definedness, and give a closure property to define a final congruence and a final notion of well-definedness. We show that these final notions satisfy natural and desired properties.

Definition 5. *Let A, B, t, $\equiv$, and u be given, cf. Definition 4. The relation* $\mathrm{Con}(t, \equiv, u) \subseteq T(A, B, t, \equiv, u)^2 \cup E(A, B, t, \equiv, u)^2$*, called the* congruence generated by t, $\equiv$ and u*, is defined as the smallest congruence $\cong$ that contains the pairs below:*

$$\mathsf{p}_1 \times \cdots \times \mathsf{0} \times \cdots \times \mathsf{p}_n \cong \mathsf{0} \tag{0a}$$

$$\mathsf{v} \cong \mathsf{0}(\mathsf{q}), \textit{ provided } \mathsf{v} :: [\mathsf{0} \to \mathsf{q}] \tag{0b}$$

$$\mathsf{v} \cong \mathsf{1}(\mathsf{p}), \textit{ provided } \mathsf{v} :: [\mathsf{p} \to \mathsf{1}] \tag{0c}$$

$$\mathsf{idv} \cong \mathsf{v} \tag{0d}$$

$$\mathsf{vid} \cong \mathsf{v} \tag{0e}$$

$$\mathsf{u}(\mathsf{vw}) \cong (\mathsf{uv})\mathsf{w} \tag{1}$$

$$\boldsymbol{\pi}_i\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle \cong \mathsf{v}_i \tag{2'}$$

$$\langle \mathsf{v}_1\mathsf{w}, \ldots, \mathsf{v}_n\mathsf{w} \rangle \cong \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle \mathsf{w} \tag{3'}$$

$$\boldsymbol{\alpha}(\boldsymbol{\alpha}(\mathsf{v}, \mathsf{w}), \mathsf{u}) \cong \boldsymbol{\alpha}(\mathsf{v}, \mathsf{uw}) \tag{4}$$

$$\mathsf{u}\boldsymbol{\alpha}(\mathsf{v}, \mathsf{w}) \cong \boldsymbol{\alpha}(\mathsf{uv}, \mathsf{w}), \textit{ provided } \mathsf{u} \rhd \mathsf{hom} \tag{5}$$

$$\langle \boldsymbol{\alpha}(\mathsf{v}_1, \mathsf{w}), \ldots, \boldsymbol{\alpha}(\mathsf{v}_n, \mathsf{w}) \rangle \cong \boldsymbol{\alpha}(\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle, \mathsf{w}) \tag{6'}$$

$$\boldsymbol{\alpha}(\mathsf{v}, \mathsf{w})\mathsf{w} \cong \mathsf{v}, \textit{ provided } \mathsf{w} \rhd \mathsf{inj} \tag{7}$$

$$\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}) \cong \mathsf{w}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}) \tag{8}$$

$$\boldsymbol{\sigma}(\mathsf{uv}, \mathsf{uw}) \cong \boldsymbol{\sigma}(\mathsf{v}, \mathsf{w}), \textit{ provided } \mathsf{u} \rhd \mathsf{inj} \tag{11}$$

$$\boldsymbol{\iota}(\mathsf{uv}{\sim}\mathsf{uw}) \cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}), \textit{ provided } \mathsf{u} \rhd \mathsf{inj} \tag{12}$$

$$\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{w}) \cong \boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{w}) \tag{13}$$

$$\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{w}) \cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}) \tag{14}$$

$$\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{d1}, \mathsf{v}{\sim}\mathsf{e1}) \cong \mathsf{0}, \textit{ provided } \mathsf{d}, \mathsf{e} \in B \textit{ with } \mathsf{d} \neq \mathsf{e} \tag{15}$$

$$\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{d1}, \mathsf{v}{\sim}\mathsf{e1}) \cong \mathsf{0}, \textit{ provided } \mathsf{d}, \mathsf{e} \in B \textit{ with } \mathsf{d} \neq \mathsf{e} \tag{16}$$

$$\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{v}) \cong \mathsf{p}, \textit{ provided } \mathsf{v} :: [\mathsf{p} \to \mathsf{q}] \tag{17}$$

$$\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{v}) \cong \mathsf{id}(\mathsf{p}), \textit{ provided } \mathsf{v} :: [\mathsf{p} \to \mathsf{q}] \tag{18}$$

$$\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \cong \boldsymbol{\sigma}(\langle \mathsf{v}_1, \ldots, \mathsf{v}_m \rangle {\sim} \langle \mathsf{w}_1, \ldots, \mathsf{w}_m \rangle) \tag{23}$$

$$\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \cong \boldsymbol{\iota}(\langle \mathsf{v}_1, \ldots, \mathsf{v}_m \rangle {\sim} \langle \mathsf{w}_1, \ldots, \mathsf{w}_m \rangle) \tag{24}$$

$$\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_{m+1}{\sim}\mathsf{w}_{m+1}) \cong \boldsymbol{\sigma}(\mathsf{v}_{m+1}\boldsymbol{\iota}_1^m{\sim}\mathsf{w}_{m+1}\boldsymbol{\iota}_1^m) \tag{25}$$

$$\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_{m+1}{\sim}\mathsf{w}_{m+1}) \cong \boldsymbol{\iota}_1^m\boldsymbol{\iota}(\mathsf{v}_{m+1}\boldsymbol{\iota}_1^m{\sim}\mathsf{w}_{m+1}\boldsymbol{\iota}_1^m) \tag{26}$$

$$\boldsymbol{\sigma}(\mathsf{v}_1{\sim}\mathsf{w}_1, \ldots, \mathsf{v}_m{\sim}\mathsf{w}_m) \cong \boldsymbol{\sigma}(\mathsf{v}_{\phi(1)}{\sim}\mathsf{w}_{\phi(1)}, \ldots, \mathsf{v}_{\phi(m)}{\sim}\mathsf{w}_{\phi(m)}) \tag{27}$$

$$\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1,\ldots,\mathsf{v}_m{\sim}\mathsf{w}_m) \cong \boldsymbol{\iota}(\mathsf{v}_{\phi(1)}{\sim}\mathsf{w}_{\phi(1)},\ldots,\mathsf{v}_{\phi(m)}{\sim}\mathsf{w}_{\phi(m)}) \tag{28}$$

$$\boldsymbol{\alpha}(\mathsf{v},\mathsf{w})\boldsymbol{\iota}(\mathsf{u}{\sim}\mathsf{z}) \cong \boldsymbol{\alpha}(\mathsf{v}\boldsymbol{\iota}(\mathsf{uw}{\sim}\mathsf{zw}),\mathsf{w}\boldsymbol{\iota}(\mathsf{uw}{\sim}\mathsf{zw}))\boldsymbol{\iota}(\mathsf{u}{\sim}\mathsf{z}) \tag{29}$$

$$\boldsymbol{\alpha}(\mathsf{v},\mathsf{w}) \circ \mathsf{d} \cong \boldsymbol{\alpha}(\mathsf{v}\boldsymbol{\iota}(w{\sim}\mathsf{d1}),\mathsf{w}\boldsymbol{\iota}(\mathsf{w}{\sim}\mathsf{d1})) \circ \mathsf{d}, \tag{30}$$

where ϕ is any permutation of $\{1,\ldots,m\}$ and $\boldsymbol{\iota}(\mathsf{v}_1{\sim}\mathsf{w}_1,\ldots,\mathsf{v}_m{\sim}\mathsf{w}_m)$ is abbreviated by $\boldsymbol{\iota}_1^m$.

When t, $\equiv$ and u are clear from the context, we let $\mathsf{v} \cong \mathsf{w}$ be an abbreviation of $(\mathsf{v},\mathsf{w}) \in \mathrm{Con}(t,\equiv,u)$.

In Definition 5, the intention of laws (0a) — (0e) should be clear from the Preliminaries. Observe that laws (0b) and (0c) comprise many different situations: $\mathsf{0}(1) \cong \mathsf{1}(0)$, $\mathsf{id}(1) \cong \mathsf{1}(1)$ and $\mathsf{1}(\mathsf{q})\mathsf{w} \cong \mathsf{1}(\mathsf{p})$, provided $\mathsf{w} :: [\mathsf{p} \to \mathsf{q}]$, are some instances of them. Note that laws (0d) and (0e) correspond to the left and right identities for composition, as mentioned in the Preliminaries. Laws (1) — (7) correspond to Equations 1 to 7 at the end of the Preliminaries. Laws (8) — (30) correspond to equations with identical numbers from Sections 2 and 3.

Note that, by Definition 4, for Definition 5 to make sense, it is required that both the left and the right hand side of each equation yield well-defined terms. Also note that, for reasons of brevity, shorthand notation is used in some of the laws, leaving out, e.g., arguments of the id, $\mathsf{0}$ and $\boldsymbol{\pi}_i$ elements, as for instance in laws (0d), (0e), (16) and law (2'). Finally note that each equation that involves elements either has identical types on the left and the right hand side, or it can be deduced from $\cong$ that they have identical types. So, for instance, in law (16), the left hand side has type $[\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{d1},\mathsf{v}{\sim}\mathsf{e1}) \to \mathsf{p}]$ (provided, e.g., $\mathsf{v} :: [\mathsf{p} \to \mathsf{q}]$) which reduces to $[\mathsf{0} \to \mathsf{p}]$ by law (15).

We stress that many of the laws of Definition 5 are families of laws, for instance law (0a) for each n and for each position of $\mathsf{0}$ in the left hand side, and law (2') for each n and i, and for each $\boldsymbol{\pi}_i(\mathsf{p}_1,\ldots,\mathsf{p}_n)$, i.e., for each combination of $\mathsf{p}_1,\ldots,\mathsf{p}_n$, and for each suitable combination of $\mathsf{v}_1,\ldots,\mathsf{v}_n$. Note that there is no need to extend law 29 to a family of laws, i.e., one that contains $\boldsymbol{\iota}(\mathsf{u}_1{\sim}\mathsf{z}_1,\ldots,\mathsf{u}_m{\sim}\mathsf{z}_m)$ and $\boldsymbol{\iota}(\mathsf{u}_1\mathsf{w}{\sim}\mathsf{z}_1\mathsf{w},\ldots,\mathsf{u}_m\mathsf{w}{\sim}\mathsf{z}_m\mathsf{w})$ instead of $\boldsymbol{\iota}(\mathsf{u}{\sim}\mathsf{z})$ and $\boldsymbol{\iota}(\mathsf{uw}{\sim}\mathsf{zw})$, because of laws (24) and (3').

Observe that laws (25) and (26) are formulated in a slightly different, but equivalent, way compared to the versions formulated in Section 3. Also observe that law (14) is formulated differently, viz. using law (26) applied to $\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w},\mathsf{v}{\sim}\mathsf{w})$, together with (0e).

Note the conditions of laws (15) and (16): by $\mathsf{d} \neq \mathsf{e}$ we mean that d and e are unequal as terms, i.e., d and e are different basic element symbols. It might be tempting to extend these conditions by dropping the requirement $\mathsf{d},\mathsf{e} \in B$, i.e., by requiring that d and e be any suitable well-defined terms.[2] This would yield a problematic semantics though: take for instance $\mathsf{d} = \mathsf{vd}'$ and $\mathsf{e} = \mathsf{ve}'$ with $\mathsf{d}',\mathsf{e}' \in B$ with $\mathsf{d}' \neq \mathsf{e}'$. Now even if we adopt as a requirement that different basic element symbols of type $[1 \to \mathsf{p}]$ represent different values from the set associated with p

[2]One might be equally tempted to require that $\mathsf{d} \ncong \mathsf{e}$, instead of $\mathsf{d} \neq \mathsf{e}$. This would be technically challenging, and equally wrong.

(which we will do in Section 6), then we cannot conclude that vd' and ve' represent different values, in much the same way that from $x \neq y$ we cannot conclude that $f(x) \neq f(y)$.

Finally, we deduce that laws (13), (14), (17) and (18), in combination with laws (27) and (28), and laws (0d) and (0e), express that arguments of $\boldsymbol{\iota}$ and $\boldsymbol{\sigma}$ form a set of pairs $\mathsf{v}{\sim}\mathsf{w}$ from which pairs of the form $\mathsf{v}{\sim}\mathsf{v}$ can be excluded. To see that, we consider the following expansion:

$$\begin{aligned}
\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{v}, \mathsf{v}{\sim}\mathsf{w}) &\cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{v})\boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{v}){\sim}\mathsf{w}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{v}))\\
&\cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}), \mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})) \circ\\
&\quad\; \boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}), \mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})),\\
&\quad\;\; \mathsf{w}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}), \mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})))\\
&\cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\mathsf{id}\boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\mathsf{id}, \mathsf{w}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\mathsf{id})\\
&\cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w})\boldsymbol{\iota}(\mathsf{v}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}), \mathsf{w}\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}))\\
&\cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}, \mathsf{v}{\sim}\mathsf{w})\\
&\cong \boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{w}),
\end{aligned}$$

where we used, respectively, laws (26) from left to right twice, law (18), law (0e), law (26) from right to left, and finally law (14).

Note that we have that $\equiv_1 \subseteq \equiv_2$ implies that $\mathrm{Con}(t, \equiv_1, u) \subseteq \mathrm{Con}(t, \equiv_2, u)$. This is used in Proposition 2 below.

Next we define the closure of a family of congruences that is built up using Definitions 2 and 3, together with Definition 5.

Definition 6. *Let A, B, t, $\equiv$, and u be given, cf. Definition 5. Let $\equiv_I$ be the identity on $TE(A,B)$. The relation $\cong \subseteq T(A,B)^2 \cup E(A,B)^2$ is defined as*

$$\cong\; = \bigcup_{k \geq 0} \cong_k$$

with $\cong_0 = \emptyset$, and

$$\cong_k = \mathrm{Con}(t, \equiv_I \cup \cong_{k-1}, u)$$

for $k > 0$.

The closure construction of Definition 6 gives us the final notions of congruence and of well-definedness. This is stated below.

Proposition 2. *Let $\cong_k$ and $\cong$ be as in Definition 6. Then we have*

(i) $\cong_k \subseteq \cong_{k+1}$ *for all $k \geq 0$, and*

(ii) $\cong\; = \mathrm{Con}(t, \equiv_I \cup \cong, u)$,

i.e., $\cong$ is the smallest congruence on $T(A, B, t, \equiv_I \cup \cong, u)^2 \cup E(A, B, t, \equiv_I \cup \cong, u)^2$ that satisfies the laws of Definition 5.

Proof. Property (i) is shown by induction on k, using the remark just above Definition 6. To prove inclusion of (ii) from right to left, it suffices to show that $\cong$ is a congruence on $T(A, B, t, \equiv_I \cup \cong, u)^2 \cup E(A, B, t, \equiv_I \cup \cong, u)^2$ that satisfies the laws of Definition 5. Since $\mathrm{Con}(t, \equiv_I \cup \cong, u)$ is the smallest such congruence, we have $\mathrm{Con}(t, \equiv_I \cup \cong, u) \subseteq \cong$. By Corollary 1 and (i) we have that $\cong$ is a relation on $T(A, B, t, \equiv_I \cup \cong, u)^2 \cup E(A, B, t, \equiv_I \cup \cong, u)^2$. To prove that $\cong$ is a congruence, we need to show properties (i) — (xiv) of Definition 4. All are proven similarly: to show (iii) for instance, let $\mathsf{x} \cong \mathsf{y}$ and $\mathsf{y} \cong \mathsf{z}$. Then $\mathsf{x} \cong_i \mathsf{y}$ and $\mathsf{y} \cong_j \mathsf{z}$ for some $i, j \geq 0$. Hence $\mathsf{x} \cong_k \mathsf{y}$ and $\mathsf{y} \cong_k \mathsf{z}$ with $k = \max(i, j)$. Since $\cong_k$ is a congruence, we have $\mathsf{x} \cong_k \mathsf{z}$ and hence $\mathsf{x} \cong \mathsf{z}$. To show that $\cong$ satisfies the laws of Definition 5, let $\mathsf{x}, \mathsf{y} \in TE(A, B, t, \equiv_I \cup \cong, u)$ be a pair of types or elements that satisfy one of the laws. By Corollary 1 and (i), $\mathsf{x}, \mathsf{y} \in TE(A, B, t, \equiv_I \cup \cong_k, u)$ for some k. Hence $\mathsf{x} \cong_{k+1} \mathsf{y}$ and hence $\mathsf{x} \cong \mathsf{y}$. The inclusion of (ii) in the converse direction is immediate by the remark just above Definition 6. □

The following properties show the soundness of the construction of Definition 6 and earlier definitions. They show that the typing relation behaves as expected: that elements are all assigned function types, that the typing relation of Definition 2 and the modality denomination of Definition 3 are closed by congruence in a natural way.

Proposition 3. *Let* $\mathsf{r}, \mathsf{p}, \mathsf{q}, \mathsf{s}, \mathsf{s}', \mathsf{x}, \mathsf{y}, \mathsf{v}, \mathsf{w} \in TE(A, B, t, \equiv_I \cup \cong, u)$*. Then we have*

(i) If $\mathsf{r} \cong [\mathsf{p} \to \mathsf{q}]$*, then* $\mathsf{r} = [\mathsf{p}' \to \mathsf{q}']$ *with* $\mathsf{p} \cong \mathsf{p}'$ *and* $\mathsf{q} \cong \mathsf{q}'$*,*

(ii) If $\mathsf{v} :: \mathsf{s}$*, then* $\mathsf{s} = [\mathsf{p} \to \mathsf{q}]$ *for some* p *and* q*,*

(iii) If $\mathsf{v} :: \mathsf{s}$*, then* $\mathsf{v} :: \mathsf{s}'$ *if and only if* $\mathsf{s} \cong \mathsf{s}'$*,*

(iv) If $\mathsf{v} :: \mathsf{s}$ *and* $\mathsf{w} \cong \mathsf{v}$*, then* $\mathsf{w} :: \mathsf{s}$*,*

(v) If $\mathsf{y} \rhd \mathsf{m}$ *and* $\mathsf{y} \cong \mathsf{y}'$*, then* $\mathsf{y}' \rhd \mathsf{m}$*.*

Proof. The proofs of (iii) — (v) are immediate by Proposition 2(ii) and by substitution of $\equiv_I \cup \cong$ for $\equiv$ in Definition 2(ix) and (x), and in Definition 3(x), respectively. The proof of (ii) is immediate by (i) and the fact that in (i) — (viii) of Definition 2, a type assignment of the form $\mathsf{v} :: [\mathsf{p} \to \mathsf{q}]$ is concluded. To prove (i), note that the only laws from Definitions 4 and 5 from which $\mathsf{r} \cong [\mathsf{p} \to \mathsf{q}]$ can be concluded, are laws (i) — (iv) of Definition 4. Induction to the length of the derivation of $\mathsf{r} \cong [\mathsf{p} \to \mathsf{q}]$ shows property (i). □

6 A sketch of semantics

In this section we give a sketch of the sets-and-functions semantics of the language developed in the previous two sections, relating the material developed there with that of Sections 2 and 3.

We assume possibly empty, countable and disjoint sets A and B of basic type and basic element symbols, a mapping $t : B \to T(A, B)$ cf. Definition 2, and a relation $u \subseteq (A \times \{\mathsf{mon}\}) \cup (B \times \{\mathsf{invfin}, \mathsf{inj}, \mathsf{hom}\})$ cf. Definition 3.

Let, for every $\mathsf{a} \in A$, $\mathcal{I}(\mathsf{a})$ be a nonempty set, let $\mathcal{I}(\mathsf{0}) = \emptyset$, the empty set, and let $\mathcal{I}(\mathsf{1}) = \{*\}$, an arbitrary but fixed one-element set.[3] If $(\mathsf{a}, \mathsf{mon}) \in u$, then we assume that $\mathcal{I}(\mathsf{a})$ comes with a proper associative and commutative binary operation $+$ and with a proper identity 0, i.e., we then treat $\mathcal{I}(\mathsf{a})$ as a commutative monoid, cf. [8].

Let the set $\mathcal{D}_0$ be the collection of all $\mathcal{I}(\mathsf{a})$ together with $\mathcal{I}(\mathsf{0})$ and $\mathcal{I}(\mathsf{1})$, i.e., $\mathcal{D}_0 = \{\mathcal{I}(\mathsf{a}) \mid \mathsf{a} \in A\} \cup \{\emptyset\} \cup \{\{*\}\}$.[4] Let $\mathcal{D}'$ be the smallest set that contains $\mathcal{D}_0$ and that is closed by the formation of arbitrary Cartesian products, finite power sets, subsets, and set exponentiation, i.e., $\mathcal{D}_0 \subseteq \mathcal{D}'$, and

(i) if $x_1, \ldots, x_n \in \mathcal{D}'$, then $x_1 \times \cdots \times x_n \in \mathcal{D}'$,

(ii) if $x \in \mathcal{D}'$, then $Fx \in \mathcal{D}'$,

(iii) if $x \in \mathcal{D}'$ and $y \subseteq x$, then $y \in \mathcal{D}'$, and

(iv) if $x, y \in \mathcal{D}'$, then $x^y \in \mathcal{D}'$.

Finally, let $\mathcal{D}'' = \bigcup \mathcal{D}'$ and let $\mathcal{D} = \mathcal{D}' \cup \mathcal{D}''$.[5] We note that $\mathcal{I}$ is a mapping $A \cup \{\mathsf{0}, \mathsf{1}\} \to \mathcal{D}_0$. We extend $\mathcal{I}$ to a mapping $TE(A, B, t, \equiv_I \cup \cong, u) \to \mathcal{D}$ next.

Following the grammar of the informal definition of types and elements prior to Definition 1, we let $\mathcal{I}(\mathsf{p})$ and $\mathcal{I}(\mathsf{v})$, with p and v well-defined, be compositional in the way expected:

$$\begin{aligned} \mathcal{I}(\mathsf{p}) \quad ::= \quad & \mathcal{I}(\mathsf{a}) \mid \mathcal{I}(\mathsf{0}) \mid \mathcal{I}(\mathsf{1}) \mid \mathcal{I}(\mathsf{q})^{\mathcal{I}(\mathsf{p})} \mid \mathcal{I}(\mathsf{p}_1) \times \cdots \times \mathcal{I}(\mathsf{p}_n) \mid \\ & F\mathcal{I}(\mathsf{p}) \mid \sigma(\mathcal{I}(\mathsf{v}_1) {\sim} \mathcal{I}(\mathsf{w}_1), \ldots, \mathcal{I}(\mathsf{v}_m) {\sim} \mathcal{I}(\mathsf{w}_m)) \end{aligned}$$

and

$$\begin{aligned} \mathcal{I}(\mathsf{v}) \quad ::= \quad & \mathcal{I}(\mathsf{b}) \mid \mathcal{I}(\mathsf{0}(\mathsf{p})) \mid \mathcal{I}(\mathsf{1}(\mathsf{p})) \mid \mathcal{I}(\mathsf{id}(\mathsf{p})) \mid \mathcal{I}(\mathsf{v}) \circ \mathcal{I}(\mathsf{w}) \mid \\ & \langle \mathcal{I}(\mathsf{u}_1), \ldots, \mathcal{I}(\mathsf{u}_n) \rangle \mid \pi_i^n(\mathcal{I}(\mathsf{p}_1), \ldots, \mathcal{I}(\mathsf{p}_n)) \mid \\ & \gamma(\mathcal{I}(\mathsf{v})) \mid \delta(\mathcal{I}(\mathsf{w})) \mid \iota(\mathcal{I}(\mathsf{v}_1) {\sim} \mathcal{I}(\mathsf{w}_1), \ldots, \mathcal{I}(\mathsf{v}_m) {\sim} \mathcal{I}(\mathsf{w}_m)), \end{aligned}$$

where $\mathcal{I}(\mathsf{0}(\mathsf{p}))$ is $0_{\mathcal{I}(\mathsf{p})}$, $\mathcal{I}(\mathsf{1}(\mathsf{p}))$ is $1_{\mathcal{I}(\mathsf{p})}$, $\mathcal{I}(\mathsf{id}(\mathsf{p}))$ is the identity on $\mathcal{I}(\mathsf{p})$, and $\mathcal{I}(\mathsf{b})$ is a function that respects the elementary typing mapping t and the relation u, i.e., $\mathcal{I}(\mathsf{b})$ is an element of $\mathcal{I}(\mathsf{q})^{\mathcal{I}(\mathsf{p})}$ if $t(\mathsf{b}) = [\mathsf{p} \to \mathsf{q}]$, that is inverse-finite, an injection or a homomorphism, whenever $(\mathsf{b}, \mathsf{invfin}) \in u$, $(\mathsf{b}, \mathsf{inj}) \in u$, or $(\mathsf{b}, \mathsf{hom}) \in u$, respectively. Note that we thus assume that $\mathcal{I}$, t and u 'work together' well; for instance, if $(\mathsf{b}, \mathsf{inj}) \in u$, then the cardinalities of $\mathcal{I}(\mathsf{p})$ and $\mathcal{I}(\mathsf{q})$ must be such that an injection $\mathcal{I}(\mathsf{p}) \to \mathcal{I}(\mathsf{q})$ indeed exists. Also, if b, b' are different elementary type symbols with $t(\mathsf{b}) = t(\mathsf{b}') = [1 \to \mathsf{q}]$, then we assume that $\mathcal{I}(\mathsf{b}) \neq \mathcal{I}(\mathsf{b}')$.

We claim that $\mathcal{I}$ is well-defined and has the expected properties.

Proposition 4. *The mapping $\mathcal{I}$ is well-defined and sound, i.e., we have*

(i) $\mathcal{I} : TE(A, B, t, \equiv_I \cup \cong, u) \to \mathcal{D}$,

[3] We use $\mathcal{I}$ to indicate *interpretation.*

[4] We use $\mathcal{D}$ to indicate *data.*

[5] $\mathcal{D}'' = \bigcup \mathcal{D}'$ means that $\mathcal{D}'' = \{d \mid d \in x \text{ and } x \in \mathcal{D}'\}$

and for all well-defined types p *and* q *and all well-defined elements* v *and* w*, we have*

(ii) *if* $\mathsf{v} :: [\mathsf{p} \to \mathsf{q}]$*, then* $\mathcal{I}(\mathsf{q})^{\mathcal{I}(\mathsf{p})}$ *is nonempty and contains* $\mathcal{I}(\mathsf{v})$*,*

(iii) *if* $\mathsf{p} \rhd \mathsf{mon}$*, then* $\mathcal{I}(\mathsf{p})$ *is a commutative monoid,*

(iv) *if* $\mathsf{v} \rhd \mathsf{invfin}$*, then* $\mathcal{I}(\mathsf{v})$ *is inverse-finite, and similarly for the cases* $\mathsf{v} \rhd \mathsf{inj}$ *and* $\mathsf{v} \rhd \mathsf{hom}$*, and*

(v) *if* $\mathsf{v} \cong \mathsf{w}$*, then* $\mathcal{I}(\mathsf{v}) = \mathcal{I}(\mathsf{w})$*, and if* $\mathsf{p} \cong \mathsf{q}$*, then* $\mathcal{I}(\mathsf{p}) = \mathcal{I}(\mathsf{q})$*.*

The proof of Proposition 4 is left to the reader.

7 Examples

In this section we give some examples that show the expressiveness of the language defined in the previous sections. We start with showing how to incorporate families of variables, indexed by a list of categories, into the language.

Example 2. In some statistics within a statistical office families of likewise variables are used, e.g., to record the answers to questions in a questionnaire. In the DSC there is no possibility to record the meaning of these variables in one stroke. Instead one must give definitions for each of the variables individually, even if these definitions show minimal differences. Thus the administrative burden is increased, as well as the risk of errors and inconsistencies. Especially in the case in which a family of variables is indexed by a list of categories, ideally it should suffice to give just one definition, in which any particular category can be substituted. We show how ths can be achieved in our language.

Let x be a list of product categories, like *shoes, pants, shirts,* etcetera. Let, for each category $d \in x$, v_d be the variable *turnover generated by the sales of d.* Thus we consider the variables *turnover generated by the sales of shoes*, *turnover generated by the sales of pants*, etcetera. Since each of these variables is assumed to be measured on a business, and to record a quantity of money, we let p be the object type *business*, q be the value type *quantity of money* and we thus assume that $\mathsf{v}_d :: [\mathsf{p} \to \mathsf{q}]$ for each d.

Now consider the type r of product categories (i.e., we assume that $\mathcal{I}(\mathsf{r}) = x$). We let each product category be a constant w_d of type $[1 \to \mathsf{r}]$. Finally, we let v be an element of type $[\mathsf{r} \to [\mathsf{p} \to \mathsf{q}]]$. The intuitive meaning of v is: given a value of type r, it returns an element of type $[\mathsf{p} \to \mathsf{q}]$. Hence it behaves as an r-indexed family of variables of type $[\mathsf{p} \to \mathsf{q}]$. Now let the intuitive meaning of v be *turnover generated by the sales of [...]*, where *[...]* indicates the substitution point of a particular product category. It is natural now to let $\mathsf{v}_d = \mathsf{v} \circ \mathsf{w}_d$. Note that the composition makes sense and is well-formed. Note also however that $\mathsf{v} \circ \mathsf{w}_d :: [1 \to [\mathsf{p} \to \mathsf{q}]] \neq [\mathsf{p} \to \mathsf{q}]$. This can be solved by adding to our congruence in Definition 5 the law $[1 \to [\mathsf{p} \to \mathsf{q}]] \cong [\mathsf{p} \to \mathsf{q}]$ (or even: $[1 \to \mathsf{p}] \cong \mathsf{p}$) which makes sense because the sets both sides of the equation represent are in a one-to-one correspondence, i.e., they are isomorphic. Note however that Proposition 4(v) no longer holds in that case, but we claim that a weaker version involving such an isomorphism does.

Next, we study the way subset inclusion can be used to organize variables according to the object types they best apply to.

Example 3. Let p be the object type *business*, let q be the value type of *economic activities*, like *agriculture, mining, construction*, etcetera, and let $\mathsf{v} : [\mathsf{p} \to \mathsf{q}]$ be the variable *main economic activity*. We assume that each economic activity is reflected as a constant $[1 \to \mathsf{q}]$, so we have, e.g., $\mathsf{agr}, \mathsf{min}, \mathsf{con} :: [1 \to \mathsf{q}]$. The object type *farm* can now be defined as to contain those businesses whose main activity is agriculture, i.e., formally as $\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{agr}1(\mathsf{p}))$. The fact that each farm is a business is reflected by $\boldsymbol{\iota}(\mathsf{v}{\sim}\mathsf{agr}1(\mathsf{p})) :: [\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{agr}1(\mathsf{p})) \to \mathsf{p}]$. The variable w of *number of livestock* applies to farms and not so much to the full object type of *business*, so it is natural to treat it as a variable of type $[\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{agr}1(\mathsf{p})) \to \mathsf{r}]$ where we let r be a value type corresponding to categories including [0..99], [100..999] and [1000..4999], say. Note that we now are in a position to define the object type *small farm* based on the number of livestock, as, e.g., $\boldsymbol{\sigma}(\mathsf{w}{\sim}[0..99]1(\boldsymbol{\sigma}(\mathsf{v}{\sim}\mathsf{agr}1(\mathsf{p}))))$, where $[0..99] :: [1 \to \mathsf{r}]$. Note that in the formal expression of *small farm*, all the necessary components to understand the object type are present: from right to left it reads that a *small farm* is a *business*, whose *main activity* is *agriculture* and whose *number of livestock* is in *[0..99]*. Next we can define additional variables on the object type *small farm* and give further specializations. We leave this to the reader.

Finally, we show how inclusion can play a role in combining two datasets that both have a variable suitable for matching.

Example 4. Let d_1 be a dataset containing two variables: one is *age of a person*, denoted by $\mathsf{v} :: [\mathsf{p} \to \mathsf{r}]$, and the other is *income of a person in 2015*, denoted by $\mathsf{w}_1 :: [\mathsf{p} \to \mathsf{q}]$. A second dataset d_2 contains *gender of a person*, denoted by $\mathsf{u} :: [\mathsf{p} \to \mathsf{o}]$, and *income of a person in 2016*, denoted by $\mathsf{w}_2 :: [\mathsf{p} \to \mathsf{q}]$. So formally, we let $\mathsf{d}_1 = \langle \mathsf{v}, \mathsf{w}_1 \rangle$ and $\mathsf{d}_2 = \langle \mathsf{u}, \mathsf{w}_2 \rangle$; note that both expressions make sense. Suppose we want to construct a third dataset with variables *age*, *gender* and *income* for those persons whose income in 2015 equals that of 2016. In terms of the given datasets, the expression for the required dataset would read

$$\langle \boldsymbol{\pi}_1 \mathsf{d}_1, \boldsymbol{\pi}_1 \mathsf{d}_2, \boldsymbol{\pi}_2 \mathsf{d}_2 \rangle \boldsymbol{\iota}(\boldsymbol{\pi}_2 \mathsf{d}_1 {\sim} \boldsymbol{\pi}_2 \mathsf{d}_2),$$

which, by law (2') is congruent to

$$\langle \mathsf{v}, \mathsf{u}, \mathsf{w}_2 \rangle \boldsymbol{\iota}(\mathsf{w}_1 {\sim} \mathsf{w}_2),$$

which, in turn, is congruent to

$$\langle \mathsf{v}, \mathsf{u}, \mathsf{w}_1 \rangle \boldsymbol{\iota}(\mathsf{w}_1 {\sim} \mathsf{w}_2),$$

by laws (3') and (8).

8 Conclusion

In this article we have defined a typed formal language for structurally modeling statistical data. The language includes a natural congruence relation, which provides a mechanism for identifying models of statistical data that are synonymous. We have given the language a sound compositional sets-and-functions semantics and we have proven some natural and desired properties of the language.

Technically, the main contribution of the article is the construction of a congruence relation in the scope of a typed language, in which types depend on 'values', or on elements as they are called here. Incorporating dependent types in a language has as a consequence that semantic techniques such as equational logic [16] don't work anymore. To make it work still, a particular closure operation needs to be constructed.

From a statistical perspective, the main contribution is the introduction of a notion of subtyping that is constructive, in contrast to similar notions from the UML with its generalization-specialization arrow, or from the Resource Description Framework (RDF) [11] based languages such as the Web Ontology Language (OWL) [17] or RDF Schema [4], with its notion of *subClassOf*. We mean by constructive that our notion of subset inclusion incorporates the conditions for the inclusion, in contrast to the other notions. This means that with UML, OWL or any such language we know of, while we can express that a *man* is a *person*, we cannot express that this is the case because of a property called *gender*. We feel that this is an important and natural addition for use within the statistical process, because of the relationship between categorical variables (such as *gender*) and the subclasses they define (viz. *men* and *women*). Moreover, the conditions for subclasses give us the mechanisms for deciding, e.g., given an arbitrary *person*, whether or not he (she) is a *man*.

In a theoretical sense we think of subset inclusion as an instance of the so-called axiom of comprehension from set theory [6]. In its most general form it states that *for any condition on x there exists a set which contains exactly those elements x which fulfill this condition* (see [6, p. 31]). The fact that the axiom of comprehension[6] is indeed a basic axiom from set theory, and is independent from the other axioms, strengthens our belief that subset inclusion is also basic and expressive. Thus we view our construct as the proper translation of the axiom of comprehension to the vocabulary of variables and data sets used in statistics: variables and data sets give us the means to express the proper conditions meant above.

A question left untouched in this article is whether or not it is decidable, given two terms v and w, whether or not $\mathsf{v} \cong \mathsf{w}$. It is crucial that decidability is established, for instance because the typing relation depends on it, cf. Proposition 3(iii): if, for instance, we want to compose two elements, in general this means that we need to make sure that the domain of the one is congruent with the codomain of the

[6]or rather: the axiom schema of subsets, which according to [6], is left of the general axiom of comprehension within Zermelo-Fraenkel set theory.

other. At this moment, we don't know whether $\cong$ is decidable, but we conjecture that it is.

One of the usual means for establishing decidability of a congruence defined on terms (i.e., to decide so-called word problems in an algebra), is to try to define a term rewriting system [2, 12], in which terms are rewritten according to equations (such as the ones defined in Definition 5) that are given a rewriting direction: either left-to-right, or right-to-left. Terms that cannot be rewritten any further are called normal forms; the rewriting mechanism thus turns the problem of deciding $\mathsf{v} \cong \mathsf{w}$ into checking whether or not the normal forms corresponding to v and w are equal or not. For this to work, the rewriting system must be (strongly) normalizing and confluent: every sequence of rewrite steps must eventually terminate with a normal form (i.e., infinite such sequences are not allowed), and, loosely speaking, the application of one rewrite rule does not block the application of another. Usually, *completion* is needed to gain both, in which rewrite rules are added to a system of already established rewrite rules; a procedure that may finish successfuly or unsuccessfully, or run forever (i.e., completion is semidecidable). At the moment we are investigating whether a proper rewriting system can be formulated. Special care must be taken to take into account the conditions on some of the congruence laws of Definition 5 involving $\triangleright$ and $::$, and the fact that some of the laws are in fact families of laws. This means that we will have an infinite actual number of rewrite rules, but we claim that this system can be reformulated into an equivalent one with a finite number of rules. Finally, because of the typing relation, a suitable nonstandard notion of a typed term rewriting system must be formulated.

During the formulation of the laws of Definition 5, we were surprised to learn that the 'interaction' between subset inclusion and aggregation is confined to two (one of which rather obscure) laws, viz. laws (29) and (30). This means that in general, aggregation and inclusion are hard to interchange: if we select some rows from a dataset and then aggregate, then in general we cannot arrive at the same result doing it the other way around in most cases. This fact, we feel, is crucial in the formulation of metadata models (or normal forms in the sense above for that matter) that claim to incorporate both inclusion and aggregation.

Acknowledgements

The author is grateful to Sander Scholtus for his careful reading of an earlier version of this article, and for his many useful comments.

References

[1] Abramsky, S., Gabbay, Dov M., and Maibaum, T. S. E., editors. *Handbook of Logic in Computer Science (Vol. 4): Semantic Modelling.* Oxford University Press, Inc., New York, NY, USA, 1995.

[2] Baader, Franz and Nipkow, Tobias. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

[3] Barr, Michael and Wells, Charles. *Category Theory for Computing Science.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[4] Brickley, D. and Guha, R.V. RDF schema 1.1 (W3C recommendation), 2014.

[5] Davey, Brian A. and Priestley, Hilary A. *Introduction to lattices and order.* Cambridge University Press, Cambridge, 1990.

[6] Fraenkel, A.A., Bar-Hillel, Y., and Levy, A. *Foundations of Set Theory.* Elsevier Science, 1973.

[7] Gelsema, Tjalling. General requirements for the soundness of metadata models. In *Joint UNECE/Eurostat/OECD work session on statistical metadata (METIS)*, 2008.

[8] Gelsema, Tjalling. The organization of information in a statistical office. *Journal of Official Statistics*, 28(3):413–440, 2012.

[9] Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1):68–95, 1977. DOI: `10.1145/321992.321997`.

[10] Grätzer, G. *Universal Algebra.* D. Van Nostrand Company, Princeton New Jersey, 1968.

[11] Hayes, P.J. and Patel-Schneider, P.F. RDF 1.1 semantics (W3C recommendation), 2014.

[12] Klop, J.W. and de Vrijer, R.C. *Term Rewriting Systems.* Cambridge University Press, 2003.

[13] Lane, S. M. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1998.

[14] Manca, V., Salibra, A., and Scollo, G. Equational type logic. *Theoretical Compututer Science*, 77(1–2):131–159, 1990. DOI: `10.1016/0304-3975(90)90118-2`.

[15] Martin, J. and Odell, J.J. *Object-Oriented Methods: A Foundation; UML Edition.* Prentice-Hall, Upper Saddle River, New Jersey, 1998.

[16] Meinke, K. and Tucker, J.V. Universal algebra. In Abramsky, S., Gabbay, M., and Maibaum, T., editors, *Handbook of Logic in Computer Science, Vol. I: Background; Mathematical Structures.* Oxford Science Publications, 1992.

[17] Motik, B., Patel-Schneider, P.F., and Grau, B. Cuenca. OWL 2 web ontology language direct semantics (second edition, W3C recommendation), 2012.

[18] Pierce, B.C. *Basic Category Theory for Computer Scientist.* The MIT Press, Cambridge Massachusetts, 1991.

[19] Pierce, B.C. *Types and Programming Languages.* The MIT Press, Cambridge Massachusetts, 2002.

[20] Signore, M., Scanu, M., and Brancato, G. Statistical metadata: a unified approach to management and dissimination. *Journal of Official Statistics*, 31(2):325–347, 2015. DOI: `10.1515/jos-2015-0020`.

[21] Thomson, S. *Type Theory and Functional Programming.* Addison-Wesley, 1994.

[22] United Nations Economic Commission for Europe (UNECE). Generic statistical information model (GSIM): Specification, 2013.

[23] van Leeuwen, J. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics.* The MIT Press, Cambridge Massachusetts, 1994.

Received 10th September 2018