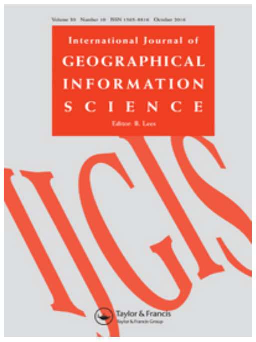


International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: http://www.tandfonline.com/loi/tgis20

Modeling spatiotemporal information generation

Simon Scheider, Benedikt Gräler, Edzer Pebesma & Christoph Stasch

To cite this article: Simon Scheider, Benedikt Gräler, Edzer Pebesma & Christoph Stasch (2016) Modeling spatiotemporal information generation, International Journal of Geographical Information Science, 30:10, 1980-2008, DOI: 10.1080/13658816.2016.1151520

To link to this article: http://dx.doi.org/10.1080/13658816.2016.1151520

Published online: 07 Mar 2016.

INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE, 2016
VOL. 30, NO. 10, 1980–2008
http://dx.doi.org/10.1080/13658816.2016.1151520

Modeling spatiotemporal information generation

Simon Scheider[a,d], Benedikt Gräler[b], Edzer Pebesma[b] and Christoph Stasch[b,c]

[a]Departement Bau, Umwelt und Geomatik, Institut für Kartographie und Geoinformation, ETH Zürich, Zürich, Switzerland; [b]Fachbereich Geowissenschaften, Institute for Geoinformatics, University of Münster, Münster, Germany; [c]52°North Initiative for Geospatial Open Source Software GmbH, Münster, Germany; [d]Human Geography and Spatial Planning, Universiteit Utrecht, Utrecht, The Netherlands

ABSTRACT
Maintaining knowledge about the provenance of datasets, that is, about how they were obtained, is crucial for their further use. Contrary to what the overused metaphors of 'data mining' and 'big data' are implying, it is hardly possible to use data in a meaningful way if information about sources and types of conversions is discarded in the process of data gathering. A generative model of spatiotemporal information could not only help automating the description of derivation processes but also assessing the scope of a dataset's future use by exploring possible transformations. Even though there are technical approaches to document data provenance, models for describing *how* spatiotemporal data are generated are still missing. To fill this gap, we introduce an algebra that models data generation and describes how datasets are derived, in terms of types of reference systems. We illustrate its versatility by applying it to a number of derivation scenarios, ranging from field aggregation to trajectory generation, and discuss its potential for retrieval, analysis support systems, as well as for assessing the space of meaningful computations.

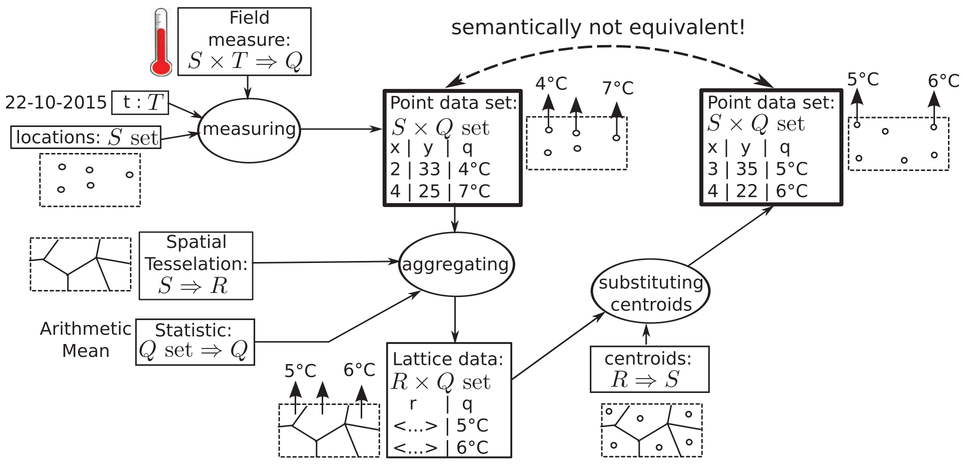
ARTICLE HISTORY
Received 1 September 2015
Accepted 2 February 2016

KEYWORDS
Spatiotemporal data types; data generation; provenance model; algebra

1. Introduction

In order to make effective use of datasets, it is necessary to know how they were generated. Only then they can be turned into meaningful products. Data itself are only sets of organized symbols. Column labels provide some interpretation, but even a comprehensible label like "temperature" might refer to raw temperatures measured at 2 m above ground, to daily averages, or interpolated values. For meaningful interpretation and further analysis, it is essential to know about these origins. For example, a dataset of spatial points with a temperature attribute may be the result of direct field measurement in meteorology, of a statistical aggregation to spatially covering regions represented by their centroids (compare Figure 1), or it may even represent a set of body temperatures measured by mobile devices of people distributed in space. Each origin causes different meanings of points, and each meaning requires a different means of data analysis. However, this meaning is not reflected in the data type. For example, the temperature attributes of the point dataset on the right of Figure 1 are spatially intensive, that is, they apply also to other points inside

CONTACT Simon Scheider simonscheider@web.de



Field measure: $S \times T \Rightarrow Q$
22-10-2015
t : T
locations: S set
measuring
Point data set: $S \times Q$ set
x | y | q
2 | 33 | 4°C
4 | 25 | 7°C
4°C
7°C
semantically not equivalent!
Point data set: $S \times Q$ set
x | y | q
3 | 35 | 5°C
4 | 22 | 6°C
5°C
6°C
Spatial Tesselation: $S \Rightarrow R$
aggregating
Arithmetic Mean
Statistic: Q set $\Rightarrow Q$
Lattice data: $R \times Q$ set
r | q
<...> | 5°C
<...> | 6°C
5°C
6°C
substituting centroids
centroids: $R \Rightarrow S$

Figure 1. How point datasets can be generated from a field in different ways, changing the meaning of points. *S* denotes points in geographic space; *R*, regions; T, time moments; and *Q*, quality values (e.g., temperature). The point dataset on the left is a result of field measurement of temperature at a certain time and a set of locations. The dataset on the right is a result of aggregating measurements to a set of covering regions (tesselation), and of representing regions with their centroid.

their corresponding region, since they denote this region's average. Thus, interpolation is trivial and rather uninformative. However, interpolation of the point dataset on the left is not trivial and can be very useful. In a similar use case, Heuvelink and Pebesma (1999) point out how in a soil mapping study, the final result critically depends on the order in which individual processing steps (running a process model, spatial interpolation of point values, spatial aggregation) are taken. In essence, we argue that in data analysis, we crucially depend on such meta-information which goes beyond the data type.

In this paper, we present a generative model of spatiotemporal information that precisely makes these distinctions, by describing how information is generated, including raw observations as well as derived products. If the who, how, and why of the entire production chain is known, meaningful analysis can be inferred and added on top of an existing data product and datasets can be queried on the basis of how they were derived or which products they could be turned into. This enables *smart data* as opposed to *smart applications*[1] (Janowicz et al. 2015): the required knowledge remains no longer hidden in the procedures of some data analysis software, but forms an essential part of the metadata which can be reused by other applications. For example, if the graph in Figure 1 would form part of the metadata of point sets, then data analysis tools such as R (R Development Core Team 2015) could query for point data with a kind of origin appropriate for kriging.

While a lot of research on spatiotemporal data in the past went into data semantics, that is, descriptions of the *what*, *when*, and *where* of data, ranging from meta-data standards[2] to ontologies (Brodaric and Gahegan 2010), only few efforts have been made so far in describing data pragmatics (Asuncion and Van Sinderen 2010), that is, *who* produced or uses data, *how* and *why* (Gahegan and Adams 2014). From a pragmatic viewpoint, the purpose (the why) of data generation and use is fundamental but has almost been neglected in the past (Couclelis 2009). Information about data producers and users (the who), on the other hand, is rather straightforward to gain, for example, based on extracting textual information on the Web (Gahegan and Adams 2014) which can be linked to datasets (Zhao and Hartig 2012). The current understanding of the data generation process itself (the how), however, still lacks good models.

For example, the PROV-O ontology (Lebo et al. 2013) can be used to maintain knowledge about the activities (times), agents, and entities involved in generating an artifact, and in this way documents provenance in the Web of data (Zhao and Hartig 2012). However, the model cannot tell us how the artifact was generated, that is, based on which procedures. Other provenance models[3] have a comparable blind spot: they mainly serve to link originators with different versions of generated artifacts, not with the way they were made. A statistical model of data generation which learns conditional probabilities between task, community, domain, and data from text descriptions was recently proposed by Gahegan and Adams in terms of a Bayesian network (Gahegan and Adams 2014). However, the model leaves open what its nodes and edges really mean, and it can only reflect what happened in the past, not what may be possible in the future.

In order to describe what can be done with datasets, another option is to use spatiotemporal data models (Worboys 1994, Yuan 1999, Mennis et al. 2005, Miller 2005, Goodchild et al. 2007, Kranstauber et al. 2012). In the past, algebras have proven useful for specifying relational (Codd 1970) and geo-relational databases (Güting 1988), as well as fundamental GIS[4] operations (Tomlin 1990, Frank and Kuhn 1995). Recently, algebraic definitions of spatial data types (Camara et al. 2014, Ferreira et al. 2014) were proposed.

While algebraic approaches allow specifying the operations on spatiotemporal data types independently from a particular data format or platform, available algebras often do not distinguish enough between data types and the concepts these types represent (Kuhn 2012). For example, while Camara et al. (2014) propose a data model for spatial fields, a spatial field (understood as an observable function from continuous space into some attribute, compare Stasch et al. 2014) is not a data type. This can be seen from the fact that the same field, for example, measurable surface temperature, can be represented by different incommensurable data types, such as a point set representing a sample of field measurements, a raster type representing satellite images, a data type of irregular areas representing aggregated values, or a set of isolines. Note also that 'field' data types (such as raster data or isolines) and 'field' concepts have very different mathematical properties[5]. For very similar reasons, Kuhn and Ballatore (2015) have put the distinction of concepts and data types at the heart of their language for spatial computation, providing a conceptual layer on top of existing technology.

We suggest that spatiotemporal concepts should be described in terms of the generative procedures that underlie datasets, outside and inside a database (Scheider 2012). This idea is captured in a reference system (Kuhn 2003), a system grounded in terms of measurements or other kinds of operations that account for reference. For this reason, we propose an algebraic model that describes possible spatiotemporal data derivations as functions on data generation procedures, where the latter are captured in terms of functions on reference systems.

We explain this idea of a two-level generation in Section 2, before we introduce spatiotemporal information types in terms of reference systems (Section 3) and our algebra in Section 4. To illustrate its applicability, we define a number of important types of derivation (Section 5) and apply corresponding derivation graphs to well-known scenarios. The diversity of scenarios illustrates the power of the algebra as a high-level language to describe data generation. In Section 6, we discuss the scope and potential of the algebra, including:

- to document, for a given dataset, how it was obtained;
- given a dataset, to reason about the possible conversions and transformations;
- to provide an abstract interface to conversion and transformation tools;
- to assess meaningfulness of data analysis.

2. Spatiotemporal data generation

There are many ways how datasets can be generated. If we say that a dataset is derived, we mean that there are other datasets from which it has been derived. This is not the case for all datasets. Some datasets are raw data in the sense that there is no other data collection from which they have been derived. Data derivation is only one way of generating datasets: another one is observation (understood as a generic term for technical sensing and perception) that generates raw datasets. The distinction between derived and raw datasets is important, because it highlights the sources of data provenance in terms of different kinds of data generation. Observation procedures underlying datasets are thereby semantically prior to derivation procedures, in the sense that they ground and give meaning to all datasets which can be derived thereupon (Scheider 2012, Stasch et al. 2014).

Spatiotemporal information is not only a collection of numbers or statements. First of all, it is a way of interpreting these numbers and statements in terms of experience and measurements, based on corresponding reference systems (Kuhn 2003). Furthermore, it is also a way of knowing how new interpretable data can be obtained. In previous work (Stasch et al. 2014), we formalized observation, prediction, and aggregation procedures as functions over reference systems in order to define new notions of meaningful prediction and aggregation. In this paper, we extend this idea and generalize spatiotemporal data generation procedures as typed functions, independently of whether they are based on observation or derivation, as shown in Figure 2.

We start reading the flowchart in Figure 2 bottom up, from inputs to outputs or to other inputs, where everything with an input and output is a function. For example, a 'field' observation has space and time as input and an observed quality (e.g., temperature) as output, whereas an 'object' observation has an object and time as input (e.g., a power plant at time *t*) and an object quality as output (e.g., tons of emitted CO_2)[6]. A simple example for deriving new functions (and thus new procedures) is to concatenate these functions with functions on output types. Take, for example, *z*-standardization. It illustrates that derivations can be considered higher-order functions (functions on functions), where fields or time series and *z*-standardization are inputs (see Figure 2), and *z*-standardized fields or time series are outputs.

Data derivation is specified in terms of algebraic operations on generation procedures, that is, in terms of higher-order functions. Generation procedures (specified as functions on basic types, see Section 3.3) can be transformed by derivation functions (specified as functions on functions, see Section 4) to other generation procedures

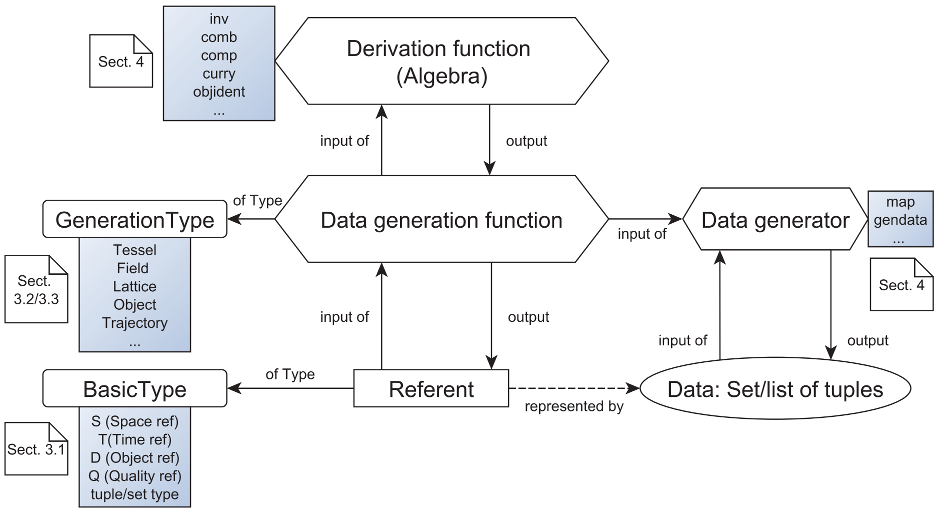


Figure 2. The general approach taken in this paper. Datasets are finite sets/lists of tuples of referents of a certain type of reference system (basic type). Generation procedures are functions over these types, and derivation operations are functions over these functions. Data generators take a generation procedure and a given dataset and generate a new dataset. Examples for all types are in gray boxes (compare references to sections of this article).

(specified as functions on basic types). Generation procedures, in turn, are inputs for data generators (see Section 4.2), that is, functions on datasets or lists (compare Figure 2). This approach has several advantages.

One advantage is that generation procedures and data types can now be explicitly distinguished, the latter being types of (finite) tuple lists or sets. Datasets result from performing generation procedures finitely many times, that is, of applying corresponding functions finitely many times on their domain. This distinguishes the possibility of data generation, which may be infinite, from a certain derivation process and its data product, which is always finite. In general, a function can be defined on an infinity of inputs even though in practice it can be applied only to a finite selection.

Our distinction between the left/middle- and the right-hand side of Figure 2 also reflects another important divide: the distinction between concepts (procedures, referents, and their types) on one hand, and data on the other. Normally, the information about concepts underlying data is thinned out considerably. However, observation and derivation are not only ways to obtain datasets, they also give meaning to them. Therefore, specific kinds of observations and derivations can be used to give meaning to *spatiotemporal* datasets. Thus, our algebra can essentially be used to add spatiotemporal concepts on the left-hand side and the middle of Figure 2 to *spatiotemporal* datasets on the right.

A further advantage of our approach is that the kinds of data generation that form inputs do not need to be known on the level of the algebra. Generation procedures can either directly denote observations, that is, starting points for derivation, or results of derivation. This makes our algebra very flexible in dealing with inputs. Furthermore, using type variables (variables that range over the types), the abstract operations of the algebra can be specified independently from types and reference systems and thus be reused on different kinds of inputs.

We have specified our algebra[7] in terms of the typed higher-order logic (HOL) *Isabelle*[8]. However, we could have used any other strong static polymorphic type language as used in functional programming (Hughes 1989), such as Haskell[9].

3. Types of spatiotemporal information

Our type system captures spatiotemporal information in terms of the domains of reference systems listed under *BasicType* and *GenerationType* of Figure 2. Basic types denote basic domains of spatiotemporal reference, while generation types denote functions that map between these domains ($'a \Rightarrow 'b$), standing for generation procedures. As will become clear in the remainder of this paper, our spatiotemporal type system includes more than just space and time referents[10]. This allows us to treat 'nonspatiotemporal' concepts as special cases of spatiotemporal ones, namely by leaving out some of the observable information.

3.1. *Basic types: space, time, object, and quality*

To begin with, we need types that anchor our datasets in spatiotemporal experience and distinguish domains of reference from each other. While many of these domains are mathematically modeled in terms of real numbers, they are not identical with these numbers (see Table 1). The domain of possible spatial point locations is denoted by the type S. S stands for the set of all possible spatial points $s \in S$. A particular spatial reference system (Iliffe and Lott 2008) uniquely identifies points (coordinate arrays) with locations on the globe. The temporal domain is identified by T and denotes moments in time. As for the spatial case, T denotes the set of all possible points in time $t \in T$, and a temporal reference system, such as *POSIX time*, uniquely identifies such a point with a moment in time. The Cartesian product $S \times T$, along with its corresponding reference systems, uniquely identifies locations in space and time. Note that there are more complex algebraic structures than $S \times T$ to model space and time and that retain its intrinsic properties. CAUSTA (Yuan et al. 2010) is an implementation based on a Clifford algebra that allows elevating metric and

Table 1. Overview of basic reference system types and simple derivations thereof.

Symbol	Type definition	Math Model	Description
S		R^3	Type of possible spatial locations with RS
T		R	Type of possible moments in time with RS
D		N	Type of possible discrete entities with RS
Q		R	Type of possible quality values with RS
R	S set	$P\ (S)$	Type of regions: bounded by polygons, curves, or collections of isolated locations and combinations thereof
I	T set	$P\ (T)$	Type of collections of moments in time: continuous intervals or a set of moments in time or combinations thereof
D set	D set	$P\ (D)$	Type of collections of object identifiers
Q set	Q set	$P\ (Q)$	Type of collections of quality values
bool		$\{T, F\}$	Boolean, also used to express predicates
Extent	$R \times I$	$P\ (S) \times P\ (T)$	Spatiotemporal extent of something
Occurs	$(S \times T)$ set	$P\ (S \times T)$	Set of spatiotemporal points (occurrences of something); footprint, support

Note: Each type needs to go along with its reference system (RS). P denotes the power set (set of all subsets).

geometric operations to space–time in a meaningful way. In this paper, we do not provide geometric data representations, but focus on modeling semantic types and well-defined information generation procedures.

In order to distinguish discrete entities, we use the domain D, which can be imagined as containing references to all the possible objects or events under study (persons, animals, factories, countries, fire outbreaks …). The reference systems of this domain need to provide means to uniquely identify discrete entities in the world, for example, based on human perception or sensors (Scheider 2012).

The fourth basic domain consists of quality values Q. Generally, the quality domain might be extended to the d-dimensional case. Every quality value comes with a reference system that uniquely determines its meaning, for example, in terms of SI base units[11]. Q may as well refer to variables with ordinal or nominal measurement scales.

3.2. *Sets, selections, and partitions*

Data can reflect values at spatial points, such as point measurements of elevation, but also values for entire regions, such as population counts or densities. In our algebra, a spatial region is defined as a particular subset $r \subseteq S$, and the type R denotes the set of all possible regions[12] ($R = \{r | r \subseteq S\}$ the power set). Recall that any $s \in S$ is only a single location and for any $r \in R$, $r \subseteq S$. This distinction is important to clearly identify the support of spatiotemporal information. Similarly, $i \subseteq T$ denotes a temporal interval or a set of time stamps. All temporal subsets are contained in $I = \{i | i \subseteq T\}$. In a similar way, we will frequently use D *set* meaning $P(D) = \{U | U \subseteq D\}$ and Q *set* meaning $P(Q) = \{U | U \subseteq Q\}$ to denote collections of objects and quality values, respectively. Note that R and I are only abbreviations of the set types to ease notation and follow common connotation.

Often it is necessary to know the time interval a time stamp is contained in, or the region in which a spatial location is located. For these purposes, we define *partitions* of basic domains: tessellations over space, time, and space–time are disjoint sets whose union is the entire domain. They are understood here as procedures to obtain the region (and/or interval) of a point in space (and/or time). A *selection*, vice versa, picks a representative location (e.g., a centroid) in space or time for a spatial region or temporal interval (see Table 2).

Table 2. Selections and partitions of basic domains, such as selection of intervals/regions that contain a certain point or vice versa points representing an interval/region.

Symbol	Type definition	Description
Select	Extent $\Rightarrow S \times T$	Select the centroid (or alike) of an extent
SSelect	$R \Rightarrow S$	Select the centroid of a region
TSelect	$I \Rightarrow T$	select the centroid of a time interval
Tessel	$S \times T \Rightarrow$ Extent	Map spatiotemporal locations to spatiotemporal extent
STessel	$S \Rightarrow R$	Spatial tessellation: map spatial locations to regions
TTessel	$T \Rightarrow I$	Temporal tessellation: map time stamps to time intervals
QPartition	$Q \Rightarrow Q$ set	Map quality values to ranges of qualities
Qstat	$(Q \Rightarrow$ bool$) \Rightarrow Q$	Summarize quality values (e.g., mean, median)

3.3. *Spatiotemporal data generation types*

3.3.1. *Fields*

Fields (Table 3) are continuous phenomena over space and time, such as air temperature (cf. Galton 2004). The definition of a spatiotemporal field is given by the type $S \times T \Rightarrow Q$. Here, every location in space and time $(s,t) \in S \times T$ is assigned one value from the quality domain $q \in Q$. The reduced case of a spatial field is defined as $S \Rightarrow Q$, and the temporal field, which is equivalent to a time series, is given by $T \Rightarrow Q$.

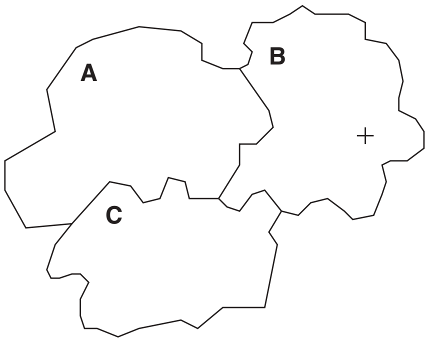
3.3.2. *Inverted fields*

Inverted fields (Table 4), sometimes referred to as coverages, are similar to fields, but rather than giving a Q value for every S and T point they delineate the (S and T) *regions* in which a given Q value (or Q value range) occurs. Typical examples of inverted fields are land cover or land use maps, or contour line maps for elevation. A spatial inverted field maps from the quality domain (e.g., land cover type: forest, water, urban, …) to output regions in which each single point has the given quality value (Figure 3, land cover type). A spatiotemporal inverted field yields volumes over space and time that possess a certain quality.

Note that an inverted field is in general not the mathematical inverse of a field, as fields are typically not bijective, that is, the same value may occur at several locations. Note also the difference to lattices, which are rather summaries over fields (see next section).

3.3.3. *Lattices*

Lattice[13] datasets (Table 5) give aggregated values over given regions (or time intervals). Examples are population counts over an administrative district, the fraction of a country that was deforested over a certain time period, or the intensity of red light averaged over a single LANDSAT pixel area. As opposed to inverted fields, for an arbitrary location within the given region or time interval, the value of the (non-aggregated) variable is not available from lattice data. Figure 3 shows an example of a lattice: the *average elevation* over polygon B does not inform us about the elevation at the location indicated by the + symbol.



	land cover type	average elevation (m)
A	Forest	631
B	Urban	220
C	Water	42
+	Urban	

Figure 3. The difference between inverted field (coverage) and lattice. For three polygons A, B, and C, land cover type and average elevation values are given in the table. At an arbitrary location inside a polygon, for example, the location marked with a +, an inverted field yields the quality for that location, such as its land cover type. For a lattice, such as polygon average elevation, the elevation value at this particular location (+) is not available, as the data refer to the whole polygon.

Values reported as an interval time series ($I \Rightarrow Q$) per region (R) (e.g., monthly averages) may stem from a spatiotemporal lattice:: $R \Rightarrow I \Rightarrow Q$. Lattices may reflect counts or densities of discrete entities (population density), or aggregations of continuous values (average elevation). The region (or time interval) size over which lattices aggregate is called the *support* of the dataset.

3.3.4. *Events*

An event refers to an instantaneous or extended occurrence of a phenomenon at some location in space and time. For this reason, events are sometimes also called occurrences (Galton and Mizoguchi 2009). Events are discrete and thus can be referenced by the identifier set D. Spatiotemporal events are instantaneous in space and time and defined as $D \Rightarrow S \times T$, while we call extended ones block events (cf. Galton 2004). Hence, an *Event* maps a discrete identifier $d \in D$ to a location in space and time $(s,t) \in S \times T$. A *RegionalEvent* maps discrete identifiers $d \in D$ to a region with a single timestamp $(r,t) \in R \times T$. Several derivations and hybrid cases are listed in Table 6. Marked events not only refer to a location in space and time but have also a quality/attribute associated with them.

Table 3. Definitions of types related to fields.

Symbol	Type definition	Description
Field	$S \times T \Rightarrow Q$	Spatiotemporal field
SField	$S \Rightarrow Q$	Spatial field
TField	$T \Rightarrow Q$	Temporal field (time series)

Table 4. Definitions of types related to inverted fields.

Symbol	Type definition	Description
InvField	$Q \Rightarrow Occurs$	Spatiotemporal inverted field
SInvField	$Q \Rightarrow R$	Spatial inverted field
TInvField	$Q \Rightarrow T$	Temporal inverted field

Table 5. Definitions of types related to lattices.

Symbol	Type definition	Description
Lattice	$R \Rightarrow I \Rightarrow Q$	Spatiotemporal lattice
SLattice	$R \Rightarrow Q$	Spatial lattice
TLattice	$I \Rightarrow Q$	Temporal lattice

Table 6. Definitions of types related to events.

Symbol	Type definition	Description
Event	$D \Rightarrow S \times T$	Spatiotemporal events
RegionalEvent	$D \Rightarrow R \times T$	Events affecting a set of locations
IntervalEvent	$D \Rightarrow S \times I$	Events lasting for some time interval
BlockEvent	$D \Rightarrow$ Extent	Events affecting a set of locations and lasting for some time interval
SEvents	$D \Rightarrow S$	Events' locations
TEvents	$D \Rightarrow T$	Events' timestamps
MarkedEvent	$D \Rightarrow S \times T \times Q$	Spatiotemporal marked events

Note: This table is not meant to be a complete listing. Further definitions can be made by adding a quality domain Q; for illustration, one such addition 'MarkedEvent' is shown.

Table 7. Definitions of types related to trajectories.

Symbol	Type definition	Description
Trajectory	$T \Rightarrow S$	Trajectory
RegionalTajectory	$T \Rightarrow R$	Trajectory of regions
IntervalTrajectory	$I \Rightarrow S$	Trajectory over temporal intervals
BlockTajectory	$I \Rightarrow R$	Trajectory over temporal intervals of regions
MarkedTrajectory	$T \Rightarrow S \times Q$	Marked trajectory

Note: This table is not meant to be a complete listing. Further definitions can be made by adding a quality domain Q; for illustration, one such addition 'MarkedTrajectory' is shown.

Table 8. Definitions of types related to objects.

Symbol	Type definition	Description
Objects	$D \Rightarrow T \Rightarrow S$	Objects in time and space
RegionalObjects	$D \Rightarrow T \Rightarrow R$	Objects in time and over regions
IntervalObjects	$D \Rightarrow I \Rightarrow S$	Objects for collections of moments and in space objects for collections of moments and over regions
BlockObjects	$D \Rightarrow I \Rightarrow R$	
ObjectTimeSeries	$D \Rightarrow T \Rightarrow Q$	Time series associated with objects
MarkedObjects	$D \Rightarrow T \Rightarrow S \times Q$	Marked objects in time and space

Note: This table is not meant to be a complete listing. Further definitions can be made by adding a quality domain Q; for illustration, two such additions 'ObjectTimeSeries' and 'MarkedObjects' are shown.

3.3.5. *Objects*

Another type of information is generated when objects (also called endurants or continuants) are observed (Galton 2004). Objects might be moving or stationary, but in contrast to events, they can be individuated in each moment of their existence (i.e., they only have spatial and no temporal parts). Thus, they are able to undergo change (Galton and Mizoguchi 2009).

An object trajectory records changes of an object's location. We define a trajectory (in general) as a function $T \Rightarrow S$ where single time stamps $t \in T$ are mapped to spatial locations $s \in S$. The regional and interval variants map time stamps $t \in T$ to a region $r \in R$ (set of locations) or temporal intervals $i \in I$ onto spatial locations $s \in S$, respectively. Adding marks (attributes) further enriches this structure (see Table 7).

Adding the object identifying domain D generates sets of object-referenced trajectories defined by $D \Rightarrow T \Rightarrow S$. An additional quality domain Q generates marked objects (see Table 8). Hence, the type *Objects* maps a discrete identifier $d \in D$ to a trajectory $T \Rightarrow S$. *RegionalObjects* map discrete identifiers $d \in D$ to regional trajectories. Note that we do not distinguish ontologically between objects and events. Instead, we suggest these categories denote different views on discrete entities, expressed in terms of different generation procedures[14].

4. An abstract algebra of data generation

In this section, we introduce primitive operations of our algebra and illustrate how they can be used to derive operations and generate datasets (compare Figure 2, upper left and bottom right, respectively). First, we give a quick introduction into the syntax of the typed HOL we use. We write x:: $'a$ for saying that expression x is of the type $'a$. Type variables ($'a$, $'b$, …) may represent any spatiotemporal information types as introduced in Section 3. Complex types are either tuples (ordered sequences of the form ('*a* × '*b*)), relations, or functions. Relations (denoted by types of the form ('*a* × '*b*) set) are used to express datasets[15], while functions (denoted by types '*a* ⇒ '*b*) are used to express data generation procedures. The operations of the algebra are *higher order functions*, that is, functions that can have any of these types as input or output. Expressions are either primitive (*x*:: '*t*) or can be obtained by *applying* functions to some other expression of the corresponding input type (*f x*:: '*u*)[16].

4.1. *Primitive operations*

Primitive operations are listed in Table 9 and explained in the following in the same order in which they appear.

4.1.1. *Operations on functions*

First, operations that manipulate functions are needed. The identity function (*id*) maps things to themselves, and inverting a function (*inv*) yields another function that maps from outputs to inputs. Since only bijective functions can be entirely inverted, the inverted function yields a definite value only for those things into which the original function maps exactly once, for all others, it yields an error[17]. Currying (*curry*) means to turn unary functions with tuples as input into *n*-ary functions, and *uncurry* means to 'undo' this[18]. A combination (*comb*) adds two functions with the same input into a single function with a tuple as output (see Table 9). We abbreviate this operator by ⊙ in (left-associative) infix notation (i.e., we can write $f_1 \odot f_2 = f_3$). A composition (*comp*,

Table 9. Primitive operations of the algebra.

Operation	Type	Description
id	:: '*a* ⇒ '*a*	Identity
inv	:: ('*a* ⇒ '*b*) ⇒ ('*b* ⇒ '*a*)	Invert a function
curry	:: ('*a* × '*b* ⇒ '*c*) ⇒ ('*a* ⇒ '*b* ⇒ '*c*)	Curry a function
uncurry	:: ('*a* ⇒ '*b* ⇒ '*c* ⇒ ('*a* × '*b* ⇒ '*c*)	Uncurry a function
comb	:: ('*x* ⇒ '*y*) ⇒ ('*x* ⇒ '*z*) ⇒ ('*x* ⇒ ('*y* × '*z*))	Function combination (**infixl** ⊙)
comp	:: ('*x* ⇒ '*y*) ⇒ ('*x* ⇒ '*z*) ⇒ ('*x* ⇒ '*z*)	Function composition (**infixl** ∘)
objident	:: '*a* set ⇒ *D*	Object identification from a set
qmap	:: *D* ⇒ *Q*	Turn object into a value
()	:: '*a* ⇒'*b* ⇒ '*a* × '*b*	Construct a pair
fst	:: '*a* × '*b* ⇒'*a*	Get the first element of a pair
snd	:: '*a* × '*b* ⇒ '*b*	Get the second element of a pair
sglton	:: '*a* ⇒ '*a* set	Construct a one element set
∩,∪	:: '*a* set ⇒ '*a* set ⇒ '*a* set	Set intersection and union
ptoset	:: ('*a* ⇒ bool) ⇒ '*a* set	Convert predicate to set
settop	:: ('*a* set) ⇒ ('*a* ⇒ bool)	Convert set to predicate
map	:: '*a* set ⇒ ('*a* ⇒ '*b*) ⇒ '*b* set	Map a function over a set
domain	:: ('*a* × '*b*) set ⇒ '*a* set	Get domain of a relation
range	:: ('*a* × '*b*) set ⇒ '*b* set	Get range of a relation
relimage	:: ('*a* × '*b*) set ⇒ '*a* set ⇒ '*b* set	Get image of a relation
prod	:: '*a* set ⇒ '*b* set ⇒ ('*a* × '*b*) set	Construct cross-product of two sets
eqcl	:: ('*a* × '*a*) set ⇒ ('*a* set set)	Get equivalence classes of relation
subfun	:: ('*x* ⇒ '*z*) ⇒ ('*x* ⇒ bool) ⇒ ('*x* ⇒ '*z*)	Get function defined on subdomain
subdom	:: ('*a* ⇒ '*b*) ⇒ '*b* set ⇒ '*a* set	Get subdomain of a function
image	:: ('*a* ⇒ '*b*) ⇒ '*a* set ⇒ '*b* set	Get image of a function

abbreviated by ∘) concatenates one function with another, yielding a function from the input of the first to the output of the second (see Table 9).

4.1.2. *Constructors for basic domains*

Some of the referents of our type system can be treated as a result of data construction. In particular, we assume constructors for objects and qualities. We construct objects from arbitrary sets (*objident*)[19] and qualities from objects (*qmap*). The latter allows us to express objects as quality values (for turning objects into fields), while the former allows us to derive new objects from datasets (e.g., for extracting objects from fields).

4.1.3. *Operations on tuples and sets*

We furthermore use a number of straightforward and well-known functions (see Table 9, lower part) that manipulate tuples, relations, and functions: () constructs a pair, *fst* (and *snd*) yields the first (and second) of a pair. The function *sglton* constructs a singleton set, and ∩ and ∪ are set intersection and union, respectively. *ptoset* and *settop* turn boolean functions (predicates) into corresponding sets and vice versa. The function *map* has a particular relevance for *data generators*. It is used for mapping a function over a set (i.e., for applying a function to all elements of a set, generating a new set). *domain* and *range* denote domains and ranges of relations; *relimage* projects a relation to its image; *prod* generates the Cartesian product of two sets; *eqcl* generates all equivalence classes from a relation, that is, the maximal sets that are symmetrically and transitively connected by that relation[20]; *image* and *subdom* generate images and subdomains of functions; and *subfun* generates a function which is defined only on a subdomain (yielding an error value otherwise).

4.2. *Defined operations*

We show by a couple of examples how we can use this algebra for constructing more complex operations. For a comprehensive list, as used in this article, see Appendix B. For example, inputs and outputs of functions can be collected into tuples and datasets. Based on this, we can define data generators that extend a dataset based on applying some generation procedure. The *out* function writes out the inputs of a function together with its outputs, *switch* and *switchtuple* switch the input of functions and sequences. The *gendata* function is an example for a data generator, as it generates the set of outputs of a function (i.e., the data generation procedure) together with its inputs over sets of inputs.

```
def out:: ('a ⇒ 'c) ⇒ ('a ⇒ ('c × 'a)) where
   out f == f Θ id
def switchtuple:: ('a × 'b) ⇒ ('b × 'a) where
   switchtuple ab = = (snd ab, f st ab)
def switch::('a ⇒ 'b ⇒ 'c) ⇒ ('b ⇒ 'a ⇒ 'c) where
   switch f = = curry(switchtuple ∘ (uncurry f))
def gendata:: 'a set ⇒ ('a ⇒'b) ⇒ ('a × 'b) set where
   gendata as f = = map as ((out f) ∘ switchtuple)
```

As another example, we define aggregation of a binary function over its subdomain (*agg*). For learning how this operation is used, see Section 5. The idea is that we take some binary data generation function (the first input *f*) as well as a subset of its domain (the second input *p*) for which we would like to aggregate *f*'s outputs using some aggregation function (*n*). An intuitive notation (using uncurried functions and sets) of this idea would look like this:

def *agg*:: ('x_0 × 'x_i ⇒ 'y) × (('x_0 × 'x_i) set) × ('y set ⇒ 'y) ⇒ 'y* **where**
  agg f p n = = n (image f p)*

However, in the following, we will often use curried functions and predicates instead of sets, in order to reduce the number of necessary derivation steps. Thus, we define the *agg* operation using a boolean version of the image operation (*imageb*):

def *imageb:: ('a ⇒ 'b) ⇒ ('a ⇒ bool) ⇒ ('b ⇒ bool)* **where**
  imageb f b = = settop (image f (ptoset b))
def *agg:: ('x_0 ⇒ 'x_i ⇒ 'y) ⇒ ('x_0 ⇒ 'x_i ⇒ bool) ⇒ (('y ⇒ bool) ⇒ 'y) ⇒ 'y* **where**
  agg f p n = = n (imageb (uncurry f) (uncurry p))

5. Spatiotemporal data derivation graphs by example

In this section, we illustrate how abstract operations can be chained to obtain spatio-temporal *derivation graphs*. Derivation graphs describe possible ways through the flowchart depicted in Figure 2, ending in a single product (i.e., they are trees rooted in this product). As we demonstrate in this section, they are also precise models of concrete derivation scenarios well known from spatiotemporal information. Thus, they are compact, precise, and flexible descriptions of given derivation methods.

If we speak of an operation, then we mean a function (from the algebra or some spatiotemporal generation function) which is applied to inputs for derivation purposes. Note that functions do not have to be applied in this sense, they can also be inputs or outputs themselves. In derivation graphs, we link operations (orange nodes) with their typed inputs (white nodes with type labels) via white arrows and with their typed outputs via black arrows. Operations of the algebra are rhombic, while spatiotemporal generation procedures are oval (Figure 4). Note that in order to keep figures to a manageable size, operation nodes are allowed to have more than one input, so that the true order of inputs is sometimes lost.

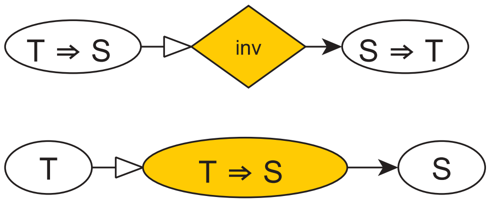


Figure 4. Derivation graphs. The upper one uses an operation from the algebra to convert a spatiotemporal function, the lower one a spatiotemporal function to generate datasets.

In the following examples, we generate datasets from fields, aggregations from fields, construct objects from fields, trajectories from object occurrences, and events from trajectories. Starting from spatiotemporal fields, we first illustrate how a dataset denoting a time series is derived by measuring this field, a procedure that applies equivalently to all generation procedures discussed in this article. Then we describe how to generate various kinds of aggregations as well as object constructions from the field. Each type of derivation is illustrated with an application example.

5.1. *Generating Mauna Loa time series data from a field*

Suppose there is a measurement procedure for measuring CO_2 concentration in arbitrary locations in space and at arbitrary times. This procedure constitutes a field, say f_{CO2}:: *Field*. If we position our sensor at a certain location, say, $l_{MaunaLoa}$:: *S*, then we fix the location and thus derive a procedure to generate a time series (*TField*) for that location, that is (*curry* f_{CO2}) $l_{MaunaLoa}$:: *TField*. If we furthermore execute this procedure (using the operation *gendata*) over a finite selection of time points *times*:: *T set* (the measurement times), then we get a time series dataset (for which monthly averages are shown in Figure 5).

$$timeseries_{MaunaLoa} :: (T \times Q)set == gendata\ times((curryf_{CO2})\, l_{MaunaLoa})$$

Note that in the following, we omit the process of data generation if it is done in an equivalent way.

5.2. *Generating lattices: averaging summer temperatures over a city*

Suppose that we want to aggregate a field variable such as temperature over a single spatial region *r*:: *R*, such as a city, using a statistic *qs*:: *Qstat* such as the sample mean (compare Figure 6(a)). This procedure can be defined as (see Appendix B for definition of *aggl*):

$$\textbf{def}\ spatialAgginR :: Field \Rightarrow R \Rightarrow Qstat \Rightarrow S\ set \Rightarrow TField\ \textbf{where}$$
$$spatialAgginR\ f\ r\ qs\ d == agg_l\ (curry\ f)\ (settop\ (r \cap d))\ qs$$

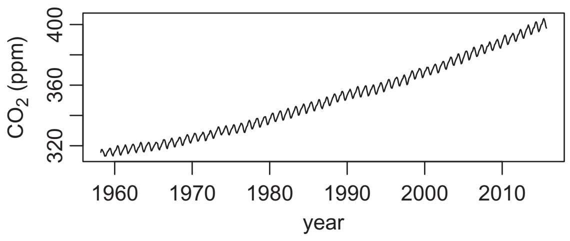


Figure 5. Monthly average atmospheric CO_2 concentration, measured at Mauna Loa, taken from columns 3 (decimal time) and 4 (average) of ftp://aftp.cmdl.noaa.gov/products/trends/ co2/co2_ mm_mlo.txt, downloaded on 10/28/2015.

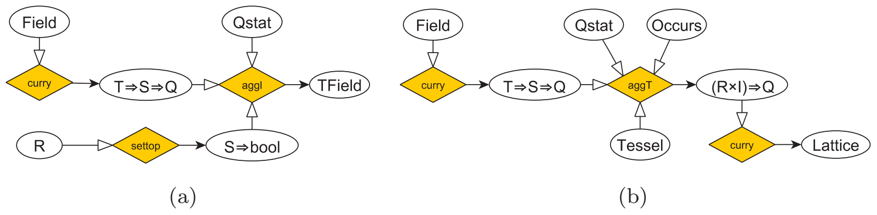


Figure 6. Aggregating a field into a time series and into a lattice. (a) The derivation graph for generating a time series from a field (compare def. *spatialAgginR*). (b) The derivation graph for aggregating a field into a spatiotemporal lattice (compare def. *sptAgg*).

Note that in this definition, a set (*d*:: *S set*) is required as input which denotes those locations where measurements were taken. For example, suppose we have a procedure for measuring temperature in arbitrary locations in space and at arbitrary times, f_{temp}:: *Field*. We would like to know average temperatures for a city whose region is *city*:: *R*. This amounts to averaging[21] the image of our field over the set of measured *locations*:: *S set* that lie in this region, for any given time. This yields a new time series procedure, a time series of averaged temperatures of this city with fixed locations of measurement.

$avgtemp_{city}$:: *TField* = = *spatialAgginR* f_{temp} *city mean locations*

Similarly, we also aggregate fields over spatiotemporal tessellations. In this case, we aggregate into a covering set of regions. The latter may cover space, time, or space–time (compare Figure 6(b), and see Appendix B for definition of *aggT*):

def *sptAgg:: Field ⇒ Tessel ⇒ Qstat ⇒ Occurs ⇒ Lattice* **where**

sptAgg field tes qs d = = *curry* (*aggT* (*curry field*) *tes qs* (*curry* (*settop d*)))

For example, suppose one is interested in average temperatures of countries over years, and we have a tessellation of space into *countries*:: *STessel* and a tessellation of time into *years*:: *TTessel*. We know the set of space–time points (*measures*:: *Occurs*) where and when measurements were taken. Then we can generate a corresponding spatiotemporal 'temperature' lattice by averaging this temperature field at the measured space–time points into this tessellation (compare also Figure 6(b), and see Appendix B for definition of $comp_2$.), and a corresponding spatial lattice by constraining this lattice to a particular year *2014*:: *I*:

templattice:: Lattice = = *sptAgg* f_{temp} ($comp_2$ *countries years* ()) *mean measures*

tempSlattice:: SLattice = = (*switch templattice*) *2014*

We can now obtain a 'pseudo' spatial point dataset (compare our introductory example in Section 1) by generating lattice data on the lattice's regions *lregions*:: *R set*, and by substituting these regions with their *centroid*:: *SSelect*:

pseudopointdata:: $(S \times Q)set$ = = *map lregions* ((*tempSlattice* ⊙ *centroid*) ∘ *switchtuple*)

5.3. *Constructing fire occurrences from a temperature field*

Suppose we know a continuous field of surface temperatures, varying over space and time, how can we identify forest fires? We will need to threshold temperature, and identify regions exceeding it. Objects are often derived from field datasets based on defining particular logical subsets of these fields, which we call here *occurrences* (($S \times T$) set), because they denote those space–time locations where that particular object occurs. For this purpose, objects need to be constructed based on identifying underlying sets. There are many possibilities to do this[22]; however, a very useful (and often used) option is to identify sets by equivalence relations, that is, relations which say whether a state of affairs 'continues' to occur over elements of a set or not. The operator *eqcl* generates equivalence classes based on such a relation, which are then transformed to objects. *genObjfromrel* constructs a set of objects in this way starting from any given relation *R*:

```
def objcons:: ('a set) set ⇒ D set where
    objcons s = = map s objident
def gen Objfromrel:: ('a × 'a) set ⇒ D set where
    genObjfromrel R = = objcons(eqcl R)
```

If we have a way to define such an equivalence relation on a field using an operator (*Field* ⇒ (($S \times T$) ⇒ ($S \times T$) ⇒ *bool*)), for example, based on comparing values in this field, then we can construct objects directly from fields:

```
def consObjFromField:: Field ⇒ (Field ⇒ ((S × T) ⇒ (S × T) ⇒ bool)) ⇒ D set
where consObjFromField f torel = = genObjfromrel(ptoset(uncurry(torel f)))
```

For example, suppose we want to construct fire objects from a spatiotemporal temperature field f_{temp}:: *Field*. We have an operation for detecting whether a fire object continues in space and time, for example, based on a temperature threshold and spatio-temporal neighborhood in this field (such that neighboring points that exceed this threshold belong to a single fire object): *fireequivalent*:: *Field* ⇒ (($S \times T$) ⇒ ($S \times T$) ⇒ bool) . A set of fire objects is then constructed from our field as follows (compare Figure 7):

fires:: *D set* = = *consObjFromField* f_{temp} *fireequivalent*

Once we have objects, how can we know about their occurrences? As discussed above (Section 4.1.2), we identify and construct objects (*objident*) by identifying underlying sets, and inverting this constructor yields a way of grounding (Scheider 2012) objects in these sets. Based on groundings, we identify object occurrences simply as the space–time

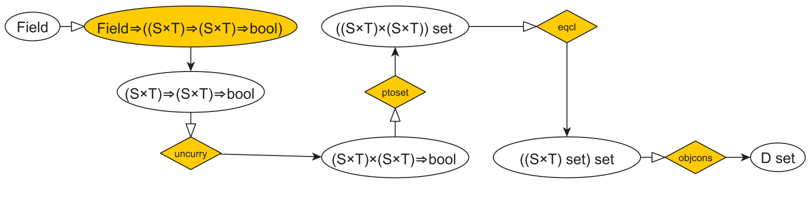


Figure 7. The derivation graph for constructing objects from fields based on an equivalence relation ((S × T) ⇒ (*S* × *T*) ⇒ *bool*) on that field (compare def. *consObjFromField*).

region in which it is grounded. In the simplest case, an object is directly grounded in space–time:

> ***def*** *getOccsfromSTObj:: D ⇒* (*S* × T*) set* **where**
> *getOccsfromSTObj* = = *inv objident*

5.4. *Deriving fire trajectories and qualities from fire occurrences*

Given that we identified fire occurrences from a field, how can we derive their trajectory, and dynamic qualities associated with them, such as their size? Based on its occurrence, we can derive data about some object, such as its quality or trajectory. Furthermore, we can derive events from these occurrences. Trajectories and qualities are formally similar: they can be constructed based on an object's temporal occurrence (*gettime*), as well as some temporal object characteristic (of type *D* ⇒ *T* ⇒ *'a*). In the case of a trajectory, this characteristic is simply its location. We generate corresponding datasets ((*T* × *'a*) set)) about the object (*getObjData*) by mapping the characteristic over those times in which the object occurs (see Figure 8):

> **def** *gettime*:: *D* ⇒ *T set* **where**
> *gettime* = = (*switch*(*getOccsfromSTObj* ∘ *map*) *snd*)
> ***def*** *getObjData:: (D* ⇒ *T* ⇒ *'*a*)* ⇒ *D* ⇒ *(*T × *'*a*) set*) **where**
> *getObjData f d* = = ((*gettime* ∘ *map*)) *d* (*out*(*f d*) ∘ *switchtuple*)

Different temporal object characteristics account for object trajectories, object qualities, and events. A trajectory is a function from time to space. In order to derive an object's trajectory from its occurrence, we need to project the space–time relation which represents its occurrence from time to space. We then obtain a function which maps from times to regions in space (compare Figure 9). Then we can use this function to generate a trajectory dataset for a given object:

> ***def*** *regTrajfromOccs::* (S × *T) set* ⇒ *RegionalTrajectory* **where**
> *regTrajfromOccs R b* = = *relimage* (*map R switchtuple*) (*sglton b*)

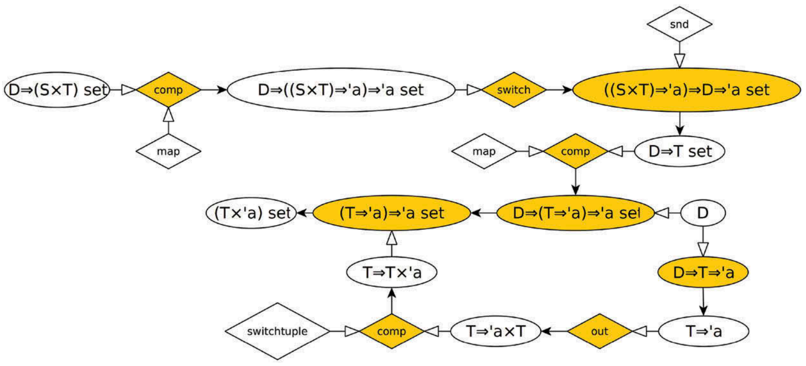


Figure 8. The derivation graph for generating temporal objects ((*T* × 'a) set) from object occurrences (*D* ⇒ (*S* × *T*) set) and some temporal object characteristic (*D* ⇒ *T* ⇒ 'a) (compare def. *getObjData*).

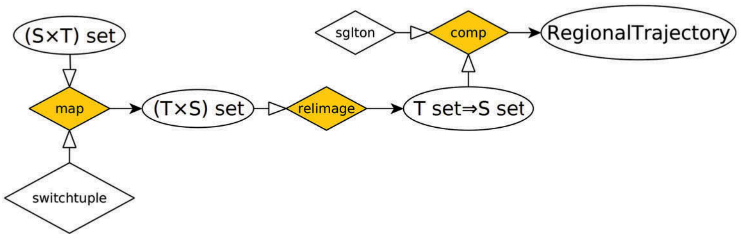


Figure 9. The derivation graph for constructing regional trajectories from occurrences by projecting them from time to space (compare def. *regTrajfromOccs*).

$$\textbf{def}\ gettrajdata :: D \Rightarrow (T \times R)\ set\ \textbf{where}$$
$$gettrajdata == getObjData\ (getOccsfromSTObj \circ regTrajfromOccs)$$

For example, in order to obtain the trajectory dataset of the fire objects from the last section (which is a moving region as depicted in Figure 10), we just have to apply the above mechanism to these objects:

$$firetrajectories :: (D \times ((T \times R)\ set))\ set == gendata\ fires\ gettrajdata$$

In a similar way, we could also generate object qualities from fields, for example, a spatial mean temperature of a fire object based on its occurrence. Furthermore, the mere occurrence of an object (e.g., a fire object) can be conceived as an event. The Rim Fire event in Yosemite in 2013 is the lifetime event of the corresponding fire object, which can be constructed by the extent of the sum of its occurrences:

$$\textbf{def}\ extentfromOcc :: (S \times T)\ set \Rightarrow Extent\ \textbf{where}$$
$$extentfromOcc\ oe == (domain\ oe, range\ oe)$$
$$\textbf{def}\ consEvfromTSObj :: BlockEvents\ \textbf{where}$$
$$consEvfromTSObj == getOccsfromTSObj \circ extentfromOcc$$

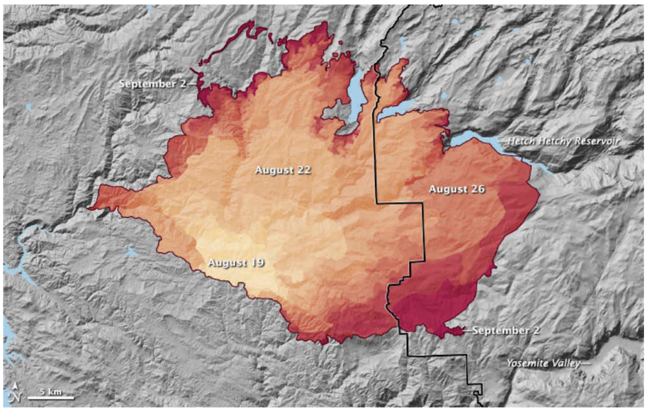


Figure 10. Spread of the Rim Fire in Yosemite from 19 August to 2 September 2013. NASA Earth Observatory image (http://earthobservatory.nasa.gov/IOTD/view.php?id=81971) using Landsat 8 data from the USGS Earth Explorer and fire extent data from InciWeb. By kind permission of NASA.

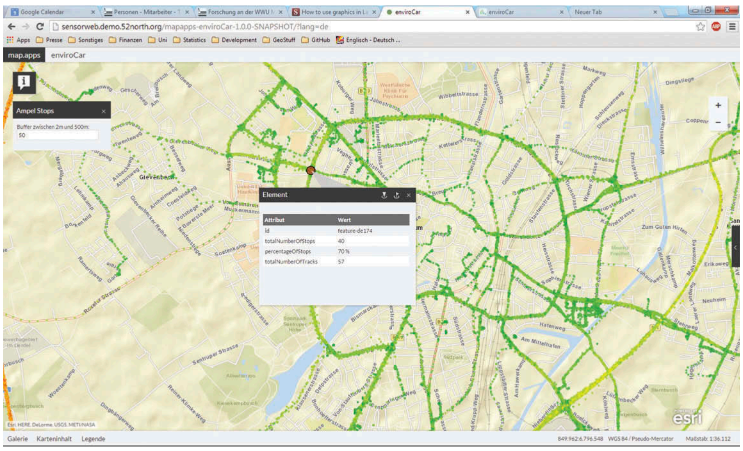
5.5. *Deriving stop events from car trajectories*

In this section, we illustrate what can be done with an object trajectory or an object time series. A very important task is to generate events from trajectories, for example, stop events from a dataset of car trajectories (compare, e.g., Palma et al. 2008). As an example, the enviroCar platform is an open platform for sustainable mobility where users are able to connect their mobile phones to sensors in a car, to collect the sensor data while driving, and to share the collected datasets with other users (Broering et al. 2015). Several analysis tools are currently being developed in this context. As a basic analysis tool for traffic planners, a simple algorithm extracts stops near a certain point of interest, for example, a street junction with traffic lights, from all tracks. Stops are defined as a set of consecutive speed measurements below 5 km/h within a track. Figure 11 depicts an overview of tracks and the analysis tool showing the number of stops at a point of interest[23].

In order to measure a car's track (a trajectory dataset), the generation function used to observe its trajectory (*cartrajectory*:: *Trajectory*) needs to be executed a finite number of times (*fixes*:: *T* set):

```
def measuretrack:: Trajectory ⇒ T set ⇒ (T × S) set where
measuretrack traj ts = = map ts ((out traj) ∘ switchtuple)
    def cartrack:: (T × S) set where
        cartrack = = measuretrack cartrajectory fixes
```

In order to derive stop events from a track, we first need to generate corresponding stop objects based on some equivalence relation (*withinstop*:: $(T \times S) \Rightarrow (T \times S) \Rightarrow bool$) which expresses that pairs of space–time points are below a maximum speed. This speed relation needs to be used as a filter on the track points of our trajectory, which is done by intersecting *withinstop* with the Cartesian product of the track points. We then derive an equivalence class from each (maximal) set of track points connected by the relation. Once we generated stops in this way, we can generate corresponding events based on their spatiotemporal extent, as above (compare Figure 12):



Ampel Stops

Buffer zwischen 2m und 500m: 50

Element

Attribut	Wert
id	feature-de174
totalNumberOfStops	40
percentageOfStops	70 %
totalNumberOfTracks	57

Figure 11. Example enviroCar: Single Stop events extracted from tracks at point of interest.

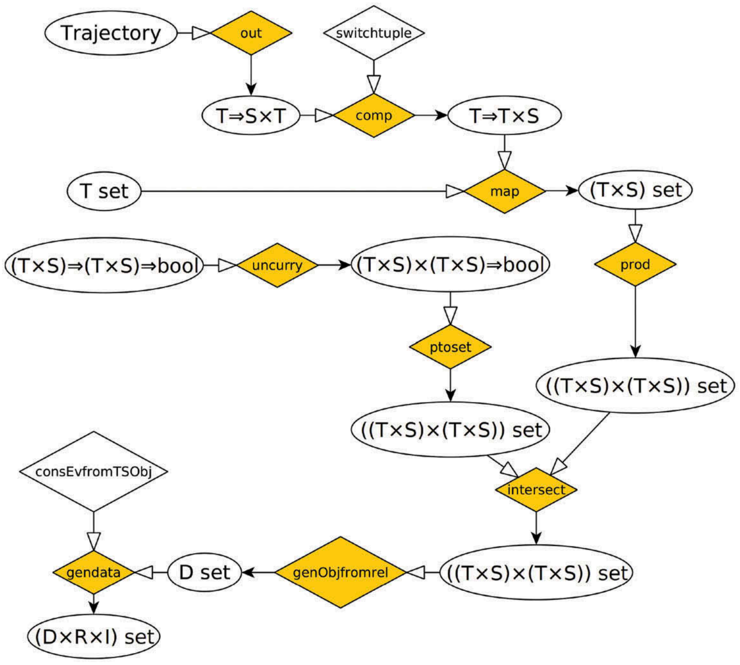


Figure 12. How to extract events from a trajectory based on a speed equivalence relation $(T \times S) \Rightarrow (T \times S) \Rightarrow bool$.

def *trackrel*:: $((T \times S) \times (T \times S))$ set **where**
  trackrel = = (*prod cartrack cartrack*) ∩ (*ptoset*(*uncurry*(*withinstop*)))
def *stops*:: *D set* **where**
  stops = = *genObjfromrel trackrel*
def *stopevents*:: $(D \times R \times I)$ set **where**
  stopevents = = *gendata stops consEvfromTSObj*

Similar to constructing events from object trajectories, we can construct events from time series. In this case, the equivalence relation needs to be defined on $(T \times Q)$ pairs. For example, the quality could be car emission or fuel consumption, and the equivalence holds between time points if the quality is above a certain threshold at these points and in between. Since this approach is analogous to the example above, we skip the details.

6. Discussion

The algebra presented in this paper provides an abstract (general) and formal (precise) model for the procedures involved in generating spatiotemporal datasets. It describes *how* datasets are derived, and it can be used for documenting data provenance as well as assessing the space of meaningful computations. We do not regard our algebra as a means of performing such computations[24]. The basic idea behind our approach is that we model core concepts (Kuhn 2012) (location, field, object, event) in terms of their generation procedures (denoted by functions), and clearly distinguish the latter from data types (denoted by tuple sets and lists).

This allows us to draw a number of important conceptual distinctions. First of all, we avoid the longstanding field-oriented bias in spatiotemporal modeling (cf. Tomlin 1990) which suggests that spatiotemporal information could be ultimately reduced to fields (cf. Camara et al. 2014, Goodchild et al. 2007). In our opinion, this view is responsible for a lot of confusion, since it tends to blur fundamental differences between fields and nonfield data types[25]. It is only in a field-oriented paradigm that nonfield concepts such as object trajectories appear as a challenge. In our algebra, basic domains are described in terms of reference systems, including not only time and space but also discrete entities such as objects, places, and events. This means that all referents are in principle on a par, as they can be the result of observation as well as data construction (cf. Scheider 2012, chapter 4). Certain kinds of objects, such as fire or weather objects, may have straightforward data constructors, while others, such as persons or places, may require the whole human conceptual apparatus. Our approach is flexible enough to allow for both kinds of provenance.

Second, since our model captures concepts as functions, it is able to distinguish a number of concepts relevant in spatial data analysis. For example, while fields are normally represented as functions from space–time to quality, in particular situations they may also be represented as inverted fields and lattices. In a similar vein, satellite data or digital elevation models are usually stored as arrays (raster data), and not as contour lines. However, sometimes, efficient queries may involve inversion of fields to answer questions like *where is the elevation higher than 200 m above sea level?* In our model, we can consistently represent fields in terms of (pseudo) inverses ($Q \Rightarrow R$) by converting one into the other, such that contour lines do not intersect and polygons do not overlap. In addition, as we argue in Pebesma et al. (2014), systems or file formats currently used for processing spatiotemporal datasets, including *R* (Bivand et al. 2013), most geographic information systems, and OpenGIS Simple Features (Herring 2011), do not distinguish lattice datasets from inverted field datasets. This means that it remains unclear whether properties (Q values) of sets of points (R: multipoints, lines, polygons, or raster cells) actually refer to qualities of particular points in that region, or rather to a summary value for the region as a whole (compare Figure 3). For the integration of datasets with other datasets (Bierkens et al. 2000) or for the statistical analysis of such datasets, this distinction matters (Besag 1974, Cressie and Wikle 2011). Our formalism can therefore help avoid wrong or meaningless calculations resulting from this ambiguity.

What is the formal expressivity and the scope of our algebra? While we demonstrated in this article that our algebra models a wide range of relevant spatiotemporal data generation procedures on an abstract level, it remains an open issue whether it can be considered complete. As for now, the algebra does not include lambda abstraction, relational algebra (Codd 1970), geometrical algebra (Yuan et al. 2010), or any kind of numerical computations. Also, we did not consider the spatial core concepts: network and neighborhood (Kuhn 2012). We therefore believe our algebra is noncomplete. Formally proving completeness, however, would first require knowledge about the expressivity needed to cover the practice of spatiotemporal data derivation, and this is considered future work. Concerning the uniqueness of the algebra regarding the number of possible paths between two concepts, we can say that our algebra is non-unique, as it must be capable of modeling different ways of obtaining the same data type (compare our introductory example). The data generation procedures sketched in this paper are intentionally simple, and cover basic but very common operations of GIS practice. In doing so we choose an abstract concept, such as a field, to represent something in the real world, and by that make an assumption explicit that goes unnoticed in practice. More elaborate data analysis procedures such as statistical inference, prediction, and simulation add complexity by making more complex assumptions, such as random variables to represent the field or a sampling process, probability distributions, or stationarity. A future challenge is to extend the algebra presented here with statistical concepts.

The algebra can be used in a variety of ways. First of all, it can be used to publish and document spatiotemporal information processes. For this purpose, a dataset together with its typed derivation graph can be published on the Web, for instance by using *linked data* principles (Zhao and Hartig 2012). This allows querying for datasets based on their data source or based on how they were generated. Furthermore, linking abstract procedures to concrete tools allows querying for different tools that can handle a given type of information, as well as querying for different information that can be handled by a given tool. This can serve as an instruction to regenerate datasets, for example, in the context of e-Science and reproducibility of computational research (Bechhofer et al. 2010). One challenge is to translate the type system of our algebra into classes of a Web ontology. Another challenge is to map tools to generation types, and to annotate datasets with their derivation graph. The latter should be automatized as much as possible in order to avoid work for data publishers. For example, a tool such as R (R Development Core Team 2015) could generate a derivation graph automatically in the background as data producers generate datasets.

Second, since our algebra is generative, it can also serve as a way to *explore* and *reason* about the space of possible derivations. Computing derivation graphs would enable *analysis support systems* that suggest users how to arrive at a certain kind of information. Such a system could furthermore be used to expand data queries to include datasets that do not directly live up to one's purpose but can be made to do so. For example, a field dataset could be retrieved based on the knowledge that it can be turned into a required lattice using that lattice's regions. Computing derivation graphs is a topic for future research. For this purpose (compare Appendix A), we can make use of type inference in functional programming (cf. Damas and Milner 1982), making sure that every type safe function application corresponds to a possible derivation. Computing every possible derivation in our algebra does not terminate, because it includes operations forming infinite derivation loops. Termination can, however, be enforced by restricting the application of functions. As shown in Appendix A, this restricted problem clearly is tractable ($O(n^2)$).

7. Conclusion

We propose an algebra as a model for spatiotemporal information generation which describes a variety of well-known derivation processes in terms of derivation graphs. Such graphs start from spatiotemporal datasets and available generation procedures (e.g., observation), and end in derived data products. They cover major practices of spatiotemporal information handling, such as derivations from and to fields, the construction of objects, events, object occurrences, and trajectories, and include various forms of aggregation over space, time, entity, or quality.

Data generation procedures are expressed as functions on basic types which stand for reference systems of space, time, quality, and discrete entities (Figure 1). Types restrict the function application possibilities. Possible derivations can be expressed as chains of function applications, where each function is either an operation of the algebra or a spatiotemporal data generation procedure. In this way, we define types of data generation such as tessellations, fields, coverages, lattices, events, objects, trajectories, and illustrate how they can be converted into each other. In contrast to existing spatiotemporal algebras, we conceptually distinguish procedures from data types, because only the former allows us to capture provenance, regardless of whether they are based on observation or derivation. Furthermore, the distinction is also relevant for assessing meaningfulness of an operation.

In contrast to existing provenance models, our algebra can be used for modeling *how* spatiotemporal information is generated. In particular, our algebra makes explicit the 'support' of datasets, that is, whether values refer to aggregated values or constant values over regions or time periods. Querying derivation graphs enables establishing whether two or more different datasets have comparable origins or may serve to derive new products. Furthermore, the algebra serves as a way of generating meaningful derivations of a dataset, to be used in data analysis support systems and improved data retrieval. For this purpose, we have discussed the tractability of the problem of generating derivation graphs.

Future challenges include the extension of the algebra to networks and statistical inference, the encoding of derivation graphs as linked data, the automated annotation of data products in analysis tools, and the computation of derivation graphs on top of which discovery, retrieval, and analysis support systems can be built. Also, formal assessments of expressivity and completeness are still missing.

Notes

1. People tend to put a lot of their understanding of a domain into how their software works, that is, into 'smart applications'. However, in this form, knowledge can hardly be reused by others.
2. http:/ /mrdata.usgs.gov/validation/
3. Such as the Provenance Vocabulary or the Open Provenance Model Vocabulary (OPMV) (Moreau et al. 2008).
4. Geographic information system.
5. The measurable function is spatially continuous and dense (as a measurement between two others may be repeated ad infinitum), whereas data as result of measurement are bound to be finite and thus always discrete (cf. Scheider and Kuhn 2011).
6. Compare the ideas about the concepts of field and object outlined in Galton (2004). In fact, many ideas in our article can be traced back to Galton (2004).
7. Code available at https://github.com/simonscheider/Algebra-of-spatio-temporal-information-generation
8. https://isabelle.in.tum.de/
9. https://www.haskell.org/, with type inference of the Damas and Milner (1982) kind. As a matter of fact, Isabelle is implemented in Haskell.
10. In distinction to classical GIS systems, where objects and events and other discrete entities are often reduced to their spatial regions or to spatial fields.
11. http://www.bipm.org/en/measurement-units/
12. Usually these include sets of locations bounded by polygon(s), curves, or sets of isolated locations.

13. Note that this notion does not refer to the mathematical concept, but is adopted from spatial statistics (Cressie and Wikle 2011).
14. Compare the arguments of Galton and Mizoguchi (2009) for regarding objects and events as different interfaces to processes, as well as the difficulties of systematically distinguishing them in Galton (2004).
15. Using sets instead of lists for our purpose is a simplification, since we cannot express redundancy and repetition of records and values. This simplification was not essential for the scenarios in our paper; however, it can become relevant when describing sampling. In that case, corresponding operations should be expressed using lists.
16. Where $f{::}\ 't \Rightarrow 'u$ is a function and $x{::}\ 't$ is a thing of the required input type. Note that binary functions $f{::}\ 't \Rightarrow 'u \Rightarrow 'v$, when applied to a single input, simply yield a unary function $f{::}\ 'u \Rightarrow v'$ and that the function $f{::}\ 't \times 'u \Rightarrow 'v$ is a unary function with a pair (tuple) as input.
17. In Isabelle, this operator is defined based on Hilbert's choice operator, compare isabelle.in.tum.de/doc/tutorial.pdf.
18. The latter is actually not a primitive, because it can be defined with inversion.
19. In Isabelle, this can be done based on a parametrized type (datatype) definition, for example, datatype '*a* Ob = obj' a set.
20. Generating first the symmetric transitive closure of the relation, and then taking the union of all equivalence classes, that is, eqcl $R = = \cup\ (x{::}\ 'a).\ \{R''\{x\}\}$.
21. Using some function mean:: $(Q \Rightarrow \text{bool}) \Rightarrow Q$, which takes a predicate over quality values (≈ a set) and returns *a single quality value*.
22. Compare the discussion in Scheider (2012), chapter 4.
23. More information about the enviroCar project and the different enviroCar analysis tools can be found at http://www.envirocar.org.
24. Even though functional programming could in fact be used for this purpose (Hughes 1989, Frank and Kuhn 1995).
25. cf. Galton (2004). Worboys (1994), Peuquet (1994), and Yuan (1999) can be regarded as attempts to integrate fields with nonfield concepts.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Asuncion, C.H. and Van Sinderen, M.J., 2010. Pragmatic interoperability: a systematic review of published definitions. *In*: P. Bernus, G. Doumeingts, and M. Fox, eds. *Enterprise architecture, integration and interoperability - EAI2N 2010. Proceedings*.. Vol. 326. IFIP Advances in Information and Communication Technology. Berlin: Springer, 164–175.

Bechhofer, S., et al. 2010. Why linked data is not enough for scientists. *In*: P. Roe, J. Hogan, and D. Abramso, eds. *Proceedings of the 2010 IEEE sixth international conference on e-Science, ESCIENCE '10*, Washington, DC, USA: IEEE Computer Society, 300–307.

Besag, J., 1974. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological*), 36 (2), 192–236.

Bierkens, M., Finke, P., and De Willigen, P., 2000. *Upscaling and downscaling methods for environmental research*. Dordrecht: Kluwer Academic.

Bivand, R.S., Pebesma, E.J., and Gomez-Rubio, V., 2013. *Applied spatial data analysis with R*. 2nd ed. New York: Springer-Verlag.

Brodaric, B. and Gahegan, M., 2010. Ontology use for semantic e-Science. *Semantic Web*, 1 (1–2), 149–153.

Broering, A., et al. 2015. enviroCar: a citizen science platform for analyzing and mapping crowdsourced car sensor data. *Transactions in GIS*, 19 (3), 362–376.

Camara, G., et al. 2014. Fields as a generic data type for big spatial data. *In*: M. Duck-Ham, et al., eds. *Geographic information science*. Vol. 8728. Lecture Notes in Computer Science. Cham: Springer International Publishing, 159–172.

Codd, E.F., 1970. A relational model of data for large shared data banks. *Communications of the ACM*, 13 (6), 377–387. doi:10.1145/362384.362685

Couclelis, H., 2009. The abduction of geographic information science: transporting spatial reasoning to the realm of purpose and design. *In*: K.S. Hornsby, et al., eds. *Spatial information theory, 9th international conference, COSIT 2009*. Aber Wrac'h, France: Proceedings Springer, 342–356. September 21-25, 2009.

Cressie, N. and Wikle, C., 2011. *Statistics for spatio-temporal data*. New York: John Wiley & Sons.

Damas, L. and Milner, R., 1982. Principal type-schemes for functional programs. *In*: R.A. DeMillo, ed.. *Conference record of the ninth annual ACM symposium on principles of programming languages*. Albuquerque, New Mexico, USA: ACM Press, 207–212. January 1982.

Ferreira, K.R., Camara, G., and Monteiro, A.M.V., 2014. An algebra for spatiotemporal data: from observations to events. *Transactions in GIS*, 18 (2), 253–269. doi:10.1111/tgis.2014.18.issue-2

Frank, A.U. and Kuhn, W., 1995. Specifying open GIS with functional languages. *In*: M.J. Egenhofer and J.R. Herring, eds. *Advances in spatial databases*. Vol. 951. Lecture Notes in Computer Science. Berlin: Springer, 184–195.

Gahegan, M. and Adams, B., 2014. Re-envisioning data description using peirces pragmatics. *In*: M. Duckham, et al., eds. *Geographic information science*. Vol. 8728. Lecture Notes in Computer Science. Cham: Springer International Publishing, 142–158.

Galton, A., 2004. Fields and objects in space, time, and space-time. *Spatial Cognition & Computation*, 4 (1), 39–68. doi:10.1207/s15427633scc0401_4

Galton, A. and Mizoguchi, R., 2009. The water falls but the waterfall does not fall: new perspectives on objects, processes and events. *Applied Ontology*, 4 (2), 71–107.

Goodchild, M., Yuan, M., and Cova, T., 2007. Towards a general theory of geographic representation in GIS. *International Journal of Geographical Information Science*, 21 (3), 239–260. doi:10.1080/13658810600965271

Güting, R.H., 1988. Geo-relational algebra: a model and query language for geometric database systems. *In*: J. W. Schmidt, S. Ceri, and M. Missikoff, eds. *Proceedings of the international conference on extending database technology: advances in database technology, EDBT '88*. London, UK: Springer-Verlag, 506–527.

Herring, J., 2011. OpenGIS implementation standard for geographic information-simple feature access–part 1: common architecture. *OGC Document 06-103r4*, 06 (103r4), 1–93.

Heuvelink, G.B. and Pebesma, E.J., 1999. Spatial aggregation and soil process modelling. *Geoderma*, 89 (1–2), 47–65. doi:10.1016/S0016-7061(98)00077-9

Hughes, J., 1989. Why functional programming matters. *Computation Journal*, 32 (2), 98–107.

Iliffe, J. and Lott, R., 2008. *Datums and map projections for remote sensing, GIS, and surveying*.. Scotland, UK: CRC Press.

Janowicz, K., et al. 2015. Why the data train needs semantic rails. *AI Magazine*, 36 (1), 5–14.

Kranstauber, B., et al., 2012. A dynamic Brownian bridge movement model to estimate utilization distributions for heterogeneous animal movement. *Journal of Animal Ecology*, 81 (4), 738–746. doi:10.1111/jane.2012.81.issue-4

Kuhn, W., 2003. Semantic reference systems. *International Journal of Geographical Information Science*, 17 (5), 405–409. doi:10.1080/1365881031000114116

Kuhn, W., 2012. Core concepts of spatial information for transdisciplinary research. *International Journal of Geographical Information Science*, 26 (12), 2267–2276. doi:10.1080/13658816.2012.722637

Kuhn, W. and Ballatore, A., 2015. Designing a language for spatial computing. Lecture notes in geoinformation and cartography. *In*: F. Bacao, M.Y. Santos, and M. Painho, eds. *AGILE 2015 - geographic information science as an enabler of smarter cities and communities*. Lisboa, Portugal: Springer, 309–326. 9-12 June 2015.

Lebo, T., et al. 2013. *Prov-o: the prov ontology*. W3C Recommendation. 30th April. https://www.w3.org/TR/prov-o/

Mennis, J., Viger, R., and Tomlin, C.D., 2005. Cubic map algebra functions for spatiotemporal analysis. *Cartography and Geographic Information Science*, 32 (1), 17–32. doi:10.1559/1523040053270765

Miller, H.J., 2005. A measurement theory for time geography. *Geographical Analysis*, 37 (1), 17–45. doi:10.1111/gean.2005.37.issue-1

Moreau, L., et al. 2008. The open provenance model: an overview. In J. Freire and D. Koop, eds. *Provenance and annotation of data and processes*. Berlin: Springer. 323–326.

Palma, A.T., et al. 2008. A clustering-based approach for discovering interesting places in trajectories. *In*: R. L. Wainwright, and H. M. Haddad, eds. *Proceedings of the 2008 ACM symposium on applied computing*, New York, NY, USA: ACM, 863–868.

Pebesma, E., et al. 2014. Meaningfully integrating big earth science data. *In*: D.-D. Rousseau, ed. *AGU fall meeting*. San Francisco, CA: American Geophysical Union, IN33A–3757. Dec 15-19, 2014.

Peuquet, D.J., 1994. It's about time: a conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of American Geographers*, 84 (3), 441–461.

R Development Core Team, 2015. *R: a language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing.

Scheider, S., 2012. *Grounding geographic information in perceptual operations. Frontiers in artificial intelligence and applications*. Vol. 244. Amsterdam: IOS Press.

Scheider, S. and Kuhn, W., 2011. Finite relativist geometry grounded in perceptual operations. *In*: M.J. Egenhofer, et al., eds. *Spatial information theory - 10th international conference, COSIT 2011*. Proceedings., Vol. 6899. Belfast, ME, USA: Lecture Notes in Computer Science Springer Berlin Heidelberg, 304–327. September 12-16, 2011.

Stasch, C., et al., 2014. Meaningful spatial prediction and aggregation. *Environmental Modelling & Software*, 51 (0), 149–165. doi:10.1016/j.envsoft.2013.09.006

Tomlin, D.C., 1990. *Geographic information systems and cartographic modeling*. Englewood Cliffs, NJ: Prentice Hall.

Worboys, M.F., 1994. A unified model for spatial and temporal information. *The Computer Journal*, 37 (1), 26–34. doi:10.1093/comjnl/37.1.26

Yuan, L., et al., 2010. CAUSTA: Clifford algebra-based unified spatio-temporal analysis. *Transactions in GIS*, 14, 59–83. doi:10.1111/tgis.2010.14.issue-s1

Yuan, M., 1999. Use of a three-domain representation to enhance gis support for complex spatiotemporal queries. *Transactions in GIS*, 3 (2), 137–159. doi:10.1111/1467-9671.00012

Zhao, J. and Hartig, O., 2012. Towards interoperable provenance publication on the linked data web. *In*: C. Bizer, et al., eds. *WWW2012 workshop on linked data on the web*. Vol. 937. Lyon: CEUR Workshop Proceedings CEUR-WS.org. 16 April, 2012.

Appendix A. Tractability of derivation graph generation

In order to assess the complexity of generating a derivation graph, we analyze the recursive algorithm *A.1*. The *correctness* of the algorithm is essentially given by the correctness of the type-checker (since each type-checked application is a valid derivation). We assume that the type-checker requires a constant effort. If $\boldsymbol{F}$ denotes a container for functions and $\boldsymbol{E}$ a container for expressions to which functions can be applied, then it must be the case that $\boldsymbol{F} \subseteq \boldsymbol{E}$, since we need to handle higher-order functions. For *completeness*, a derived function $\boldsymbol{f}$ must therefore be checked against both sets. Furthermore, the algorithm needs to enforce *termination*. This is done by the following three restrictions: (1) we make sure to generate only novel types (line 16), which prevent loops in the generated graph; (2) we allow functions to be applied only a constant number of times. In the case of *concrete functions* (functions without type variables), each function can be applied only *once*, since applying it more than once would produce the same type again, contradicting (1). For reasons of simplicity, we consider only concrete functions in the remainder, which can be removed from the container $\boldsymbol{F}$ as soon as their application was successful (line 17). (3) Even though the production of new functions adds to $\boldsymbol{F}$ (line 22), we assume this has a probability below 1 (i.e., not every application generates a new function). This has the effect that the set of newly generated functions decreases in size. We denote this probability by **1/*b***, with ***b*** > 1, assuming it is a constant. Together with (2) this assures that the container ***F*** is eventually consumed up, and thus the algorithm terminates.

Regarding *complexity*, we now have the following situation: by conditions 1–3, a complexity explosion as suggested by the recursive structure is prevented. In the best case, that is, if the set of functions is not expanded at all, F is consumed up linearly precisely after $|F|$ successful applications (where F is the initial set of functions). To find these applications, the effort of iterating over all function expression pairs is $||F|(|E|-1)+\sum_{i=1}^{|F|}|F|-i$ (compare lines 7–11 and line 16), where the second addend says that maximally $|F|$ new expressions can be added to ***E*** that need to be checked against the remaining functions in F. This simplifies to $|F|(|E|-1)+1/2|F|^2$. Thus, the order of complexity is lower bounded by $\Omega(n^2)$ (where $n = |E|$). Note that this result could be further improved by using an index on types to search through the sets E and F, yielding $\Omega(n \log n)$. A realistic runtime measure however depends additionally on the decay of the increase of new functions.

Algorithm *A.1* An algorithm for generating a data derivation graph.

Input: Γ = a container of typed spatio-temporal data generation functions, and Δ = a container of typed spatio-temporal data sets. **type** allows checking type equivalence (**type** x = **type** y) and error (**type** x = error).

Output: Expanded container E (of expressions), a set of their *Types,* as well as a derivation graph G.

```
 1: function EXPAND($\Gamma$, $\Delta$)
 2:     $F \leftarrow \Gamma$                                   ▷ Container for functions
 3:     $E \leftarrow \Gamma \cup \Delta$                       ▷ Container for expressions
 4:     Types ← null
 5:     for $e \leftarrow E.next()$ do
 6:         Types.add(type e)                                 ▷ Adding initial types
 7:     while $f \leftarrow F.next()$ do                     ▷ This is where the magic starts!
 8:         checkFunction(f, E)
 9:     function CHECKFUNCTION(f, E)
10:         while $e \leftarrow E.next()$ do
11:             apply(f, e)
12:     function CHECKEXPRESSION(e, F)
13:         while $f \leftarrow F.next()$ do
14:             apply(f, e)
15:     function APPLY(f, e)
16:         if $f \neq e$ and type $f\ e \neq$ error and type $f\ e \notin$ Types then    ▷ Enter recursion only if
    functions are applicable and derived type is novel
17:             F.remove(f)                               ▷ Functions can be applied only once
18:             G.add(edge(e,f)).add(edge(f,f e)
19:             Types.add (type f e)
20:             if type f e = ('a =>' b) then
21:                 checkFUNCTION( f e, E)
22:                 F .add(f e)
23:             checkExpression(f e, F)
24:             E .add(f e)
25:     return E, G, Types
```

If we assume the probability of generating a new function by way of a function application is $1/b$, then the total increase of functions is described by the series $\mathbf{1/b}\ |\boldsymbol{F}| + 1/b(1/b\ |\boldsymbol{F}|) + \ldots$ (until the increase falls below 1, with F being the initial set of functions), and so the total sum of newly generated functions is $|F^*|\ \sum_{j=1}^{\text{floor}(\log_b|F|)}(1/b)^j|F|$ (compare Figure A1). Note that $|F^*|$ is less than $|F|$ for probabilities <1/2, while it is approximately exponential $\left(|F|^{\text{floor}(\log_b|F|)}\right)$ for probabilities near 1. For each function in F^*, we need to iterate over all expressions (such that novel functions add to expressions, see lines 24 and 21) as well as the remaining functions (such that old functions are removed and new ones added, see lines 17 and 23): $_{j=0}^{|F^*|-1}(|E|+j)+(|F|+j-j-1)$. For probabilities <1/2 ($|F| \approx |F^*|$), this simplifies to $|F|(|E| + (|F| - 1)) +$

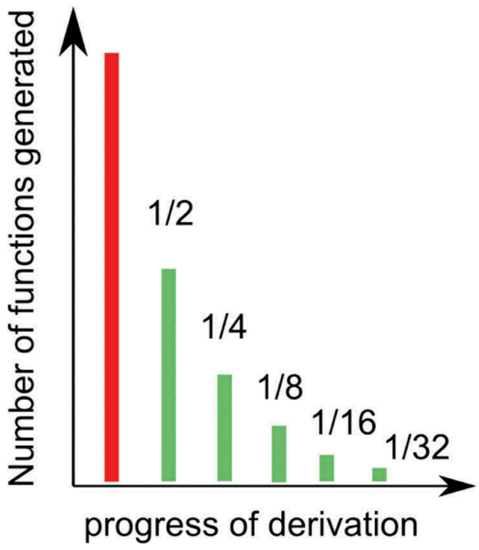


Figure A1. The series of increase of functions with a probability of 1/2.

$1/2|F|^2$ and the total number of functions is $|F^*| + |F| \approx 2|F|$. Thus, if we assume $\boldsymbol{b} = 2$, the worst case time complexity is $|F|(|E| - 1) + |F|^2 + |F|(|E| + (|F| - 1))$ and thus upper bounded by $O(n^2)$. Note if we assume a less conservative increase of functions, this can lead to exponential complexity. However, a realistic probability parameter still remains to be investigated.

Appendix B. Definitions

Here is a list of all operations used in this article that are derived from our algebra and were not introduced in the text:

B.1. *Operations on functions*

def $comb_2$:: $('u \Rightarrow 'x \Rightarrow 'y) \Rightarrow ('u \Rightarrow 'x \Rightarrow 'z) \Rightarrow ('u \Rightarrow 'x \Rightarrow ('y \times 'z))$ **where**
  $comb_2\ f_1\ f_2 == curry((uncurry\ (f_1))\ \Theta\ (uncurry(f_2)))$
def $comp_2$:: $('x \Rightarrow 'y) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('y \Rightarrow 'b \Rightarrow 'z) \Rightarrow ('x \Rightarrow 'a \Rightarrow 'z)$ **where**
  $comp_2\ f_1\ f_2\ f == switch(f_2 \circ (switch(f_1 \circ f)))$
def $subdomproj$:: $('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ set$ **where**
  $subdomproj\ f\ b == subdom\ f\ (sglton\ b)$

B.2. *Function aggregations*

def agg_l:: $('x_0 \Rightarrow 'x_i \Rightarrow 'y) \Rightarrow ('x_0 \Rightarrow bool) \Rightarrow (('y \Rightarrow bool) \Rightarrow 'y) \Rightarrow ('x_i \Rightarrow 'y)$ **where**
  $agg_l\ f\ p\ n == ((switch((switch\ f) \circ imageb))\ p) \circ n$
def agg_r:: $('x_0 \Rightarrow 'x_i \Rightarrow 'y) \Rightarrow ('x_0 \Rightarrow bool) \Rightarrow (('y \Rightarrow bool) \Rightarrow 'y) \Rightarrow ('x_i \Rightarrow 'y)$ **where**
  $agg_r\ f\ p\ n == agg_l\ (switch\ f)\ p\ n$
def $aggT$:: $('x_0 \Rightarrow 'x_i \Rightarrow 'y) \Rightarrow ('x_0 \Rightarrow 'x_i \Rightarrow 'A) \Rightarrow (('y \Rightarrow bool) \Rightarrow 'y) \Rightarrow ('x_0 \Rightarrow 'x_i \Rightarrow bool) \Rightarrow$
  $('A \Rightarrow 'y)$ **where**

aggT f r n d = = (subdomproj (uncurry r)) ∘ *(*∩ *(ptoset (uncurry d)))* ∘ *((switch map) (uncurry f))* ∘ *settop* ∘ *n*

def *aggTr:: (*'x_0 ⇒ 'x_i ⇒ 'y*)* ⇒ *(*'x_i ⇒ 'A*)* ⇒ *(('y* ⇒ *bool)* ⇒ 'y*)*⇒ *('x_i* ⇒ *bool)* ⇒ *(* 'x_0 ⇒ *'A* ⇒ 'y*)* **where**

aggTr f r n d = = curry ((uncurry (f ∘ *(comp2 id ((subdomproj r)* ∘ *(*∩ *(ptoset d))) (switch map))))* ∘ *settop* ∘ n*)*

def *aggTl:: (*'x_0 ⇒ 'x_i ⇒ 'y*)* ⇒ *(* 'x_0 ⇒ 'A*)* ⇒ *((* 'y ⇒ *bool)*⇒ *'y)* ⇒ *(* 'x_0 ⇒ *bool)* ⇒ *(*'A ⇒ 'x_i ⇒ 'y*)* ***where*** *aggTl f r n d = = switch (aggTr (switch f) r n d)*