Unpublished preprint

Algebra of core concept transformations

Procedural meta-data for geographic information

Niels Steenbergen[1*], Eric Top[1], Simon Scheider[1] and Enkhbold Nyamsuren[1]

[1*]Department of Human Geography and Spatial Planning, Utrecht University, Princetonlaan 8a, Utrecht, 3584CB, The Netherlands.

*Corresponding author(s). E-mail(s): n.steenbergen@uu.nl;
Contributing authors: e.j.top@uu.nl; s.scheider@uu.nl;
e.nyamsuren@uu.nl;

Acknowledgments

This work was developed within the QuAnGIS project, supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803498).

Abstract

Transformations are essential for dealing with geographic information. They are involved not only in converting between geodata formats and reference systems, but also in turning geodata into useful information according to some purpose. However, since a transformation can be implemented in various formats and tools, its function and purpose usually remains hidden underneath the technicalities of a workflow. To automate geographic information procedures, we therefore need to model the *transformations* implemented by workflows *on a conceptual level*, as a form of *procedural knowledge*. Although core concepts of spatial information provide a useful level of description in this respect, we currently lack a model for the space of possible transformations between such concepts. In this article, we present the algebra of core concept transformations (CCT). It consists of a type hierarchy which models core concepts as relations, and a set of basic transformations described in terms of function signatures that use such types. Type inference allows us to enrich GIS workflows with abstract machine-readable metadata, by compiling algebraic tool descriptions. This allows us to

automatically infer goal concepts across workflows and to query over such concepts across raster and vector implementations. We evaluate the algebra over a set of expert GIS workflows taken from online tutorials.

Keywords: Conceptual transformations, core concepts, geographic information, workflows

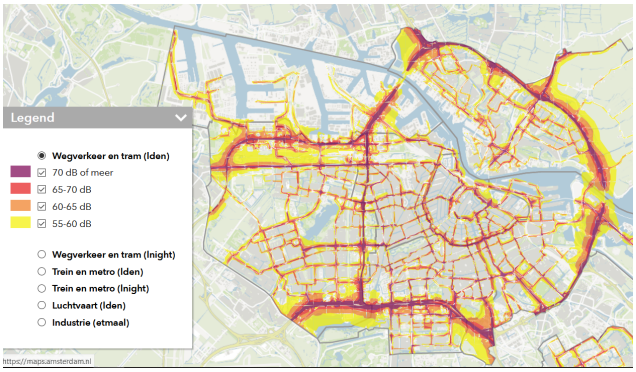
1 Introduction

Manipulating and processing geographic information is a core competence relevant for data analysts. It is not only needed to transform maps across different coordinate reference systems, it is also an essential means for preparing geodata for analysis and simulation. For example, to measure the effect of noise on the health of citizens [1], we may need to transform a noise contour map (Fig. 1a) into a statistical summary that quantifies the amount of noise within administrative regions. Using appropriate tools, we can derive a choropleth map showing the *proportion of the area covered by 70dB noise for each neighborhood* (Fig. 1b) from this contour map.

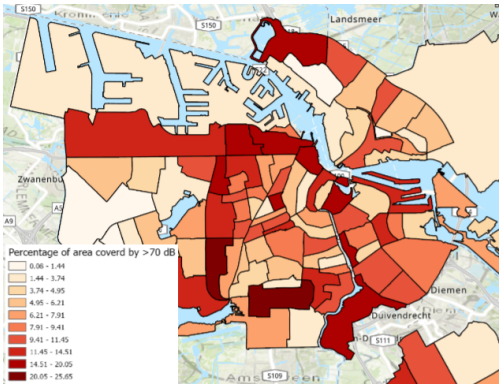
As we illustrate below, the contour and the choropleth map in the example above correspond not only to different computational or cartographic steps, but also to different *forms of geographic information*, i.e., to different conceptualizations. We therefore call such manipulations *conceptual transformations*. In this article, we are interested in understanding how such transformations are designed accurately, and how this can be modeled. What *kind of procedural knowledge* is needed to understand transformations, i.e., to understand why a tool can be picked for transforming the contour map into a map that measures the intended proportions?

Today, it seems that, although we have textbooks explaining the required methods in detail [3] and workflow management has become standard practice in Geographic Information Systems (GIS), we cannot claim to understand this workflow design practice very well [4] — at least not well enough for automating it [5]. In the past, we have seen attempts at annotating GIS operations and services with semantic descriptions to make them reusable and interoperable, in particular in the form of Web services [6–9]. Also, understanding GIS in terms of transformations has been suggested earlier [10].

However, while corresponding technologies are important enablers for (partial) automation, systematizing GIS functionality needed in these services turned out to be hard [4, 11, 12]. A useful theory that links GIS tools with tasks seems still out of sight [13]. We believe this is at least partially due to a lack of focus on the *know-how* of transformations on a conceptual level. In the past, researchers have focused on tool technicalities in a raster or vector format, needed for implementation of, but insufficient for, reasoning with GIS [14, 15]. Although geocomputational workflows are necessarily implemented



(a) Map of noise contours in Amsterdam. Source: [2]



(b) Choropleth map of proportion of area covered by noise $>$ 70 dB on the level of neighborhoods.

Fig. 1: Transformation of noise contour map into a choropleth map.

in some format, their design requires reasoning on a conceptual level which goes beyond geodata models.

To highlight this point, consider again our example from above. Note that analytically, the important feature of the noise contour map is not that it is in vector format, nor that noise values are given as integers, but that the data can be interpreted as a *spatial field*, i.e., a spatially continuous function of noise measures, as opposed to a collection of discrete objects. Aggregations therefore cannot be counts, but should be field integrals or field coverage amounts. In the latter case, we are measuring the area covered by a range of field values ($\geq$ 70 dB) within the extent of objects (neighborhoods) (cf. dotted boxes in Fig. 2), and then form an area proportion. Note that this holds *regardless* of whether such an operation, in turn, is implemented in terms of raster (zonal map algebra) or vector overlay (combining e.g. intersect and dissolve) (cf. schematized workflows in Fig. 2). Since the same task can

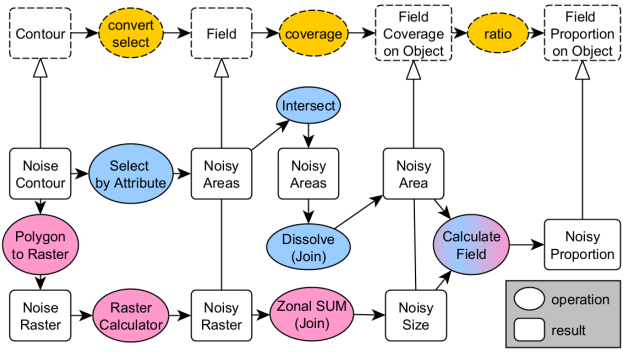


Fig. 2: How to transform a noise contour map into a measure of the proportion of noisy area? The dotted transformations are on a *conceptual level*, in parallel to computational procedures implemented with different data models (blue: vector version, red: raster version). Procedures are schematized versions of ArcGIS workflows from task 1b in Sect. 5.

be implemented by different workflows in different formats, *neither data nor tool formats provide a basis for understanding the function and purpose of these workflows*. Correspondingly, scripts may be enough for executing, but are neither sufficient for synthesizing nor for retrieving such a workflow [16].

A spatial field is an example of a *core concept of spatial information* [17]. Core concepts capture the various ways in which the geographic environment can be conceptualized. They have been successfully used as basis for semantic data types to model GIS functions [15] and for semantic workflow synthesis [18]. However, we still lack a model that can be used to describe the space of conceptual transformations over entire workflows and across different implementations. Such a model would make not only the automated composition, but also the description and retrieval of workflows independent from their implementation. In this article, we introduce the algebra of core concept transformations (CCT) which can be used to automatically infer conceptual descriptions of entire workflows from tool descriptions, and allows retrieving workflows based on how they transform concepts into one another.

In what follows, we first discuss previous work on core concepts and the state-of-the art in GIS algebra (Sect. 2), before explaining our methodology for designing the algebra (Sect. 3). We explain how core concepts can be modeled as types of relations in the sense of relational algebra (Sect. 4). We then introduce workflows in Sect. 5 and specify underlying tasks using such types. In Sect. 6, we explain how types can be used to add information to tools and workflows. Afterwards, we specify atomic transformations as function signatures in Sect. 7. Finally, we test our model by describing and retrieving expert workflows taken from a range of documented GIS scenarios (Sect. 8).

2 Related work

We review related work on using algebras in the context of GIS and workflows, and then explain core concepts as a way to conceptualize geographic information.

2.1 GIS and algebra

In general, our work is situated in the context of describing geo-operators [19] and automating GIS workflows [4] and corresponding services [9]. However, such descriptions usually focus on syntactic labels and inputs or outputs of tools, and thus captures functionality only on a superficial level. The idea of using *algebras* to describe GIS functionality goes back to Dana Tomlin's *map algebra* [20]. Since it captures only a small part of GIS functionality, which is closely related to the raster data model, there have been various attempts at extending it [21]. Other researchers have focused on functional algebras as more general models to design or implement GIS [22, 23], e.g. based on fields [24, 25] and GIS operations in relational databases [26]. These approaches [14, 21, 22, 26] propose generalized *data* models for *implementing* software, not *conceptual* models for *describing* it. Our algebra builds on the preliminary ideas in [15, 27], but goes beyond these in the following respects:

First, unlike other approaches, we are not targeting efficient or elegant implementations, since core concepts are not data types, [17] though data might be interpreted using such concepts. Previous focus on implementation may have prevented success of earlier attempts at systematizing GIS operations because they overlooked these concepts [12]. Conceptual transformations, in contrast, remain relatively stable across different implementations. Fields, e.g. are not treated as a data type [25], but as a concept that can have raster and vector implementations, as suggested in [15].

Second, we use our algebra as a definition, annotation and reasoning language for tools and workflows, primarily for the purpose of retrieval, in addition to synthesis (cf. [18]). Going beyond [27], our type system accommodates a form of subsumption reasoning with parameterized types which allows succinct, abstract signatures for atomic transformations. Third, in contrast to [27], concepts are not modeled as functions, but rather as *relations* in the sense of relational algebra [28], while functions stand for concept transformations. This makes it easy to model partial and inverted fields as well as spatial networks, because the latter are essentially relational concepts [29], as explained below.

2.2 Core concepts and amounts

Core concepts of spatial information were proposed by [14, 17, 30] as generic interfaces to GIS in the sense of conceptual 'lenses' through which the environment can be studied. Core concepts are results of human conceptualization and interpretation and are thus usually not explicit in data types. For our purpose, we make use of the following concepts:

- *Fields*: capture qualities [30] at locations of some metric space and at some time. A prime example is a temperature field. Since field values are separated by spatial distance, one can study change as a function of spatial distance. Though field qualities also change in time, we consider only spatial fields, i.e., snapshots of a field in time [27].
- *Objects*: Spatial objects are 'endurants' [31], meaning they can change their location and quality in time while remaining their identity. Objects are spatially bounded even if their boundary may be fuzzy. We consider geographic places with bona fide (perceivable) and fiat (conventional) boundaries as objects.
- *Networks*: Networks measure a relationship between objects [29, 30]. Networks are e.g. commuter flow matrices or traffic links in a road network. Networks can change their relational quality in time. Similar to networks, we also consider locations as keys of a relation. The latter are called *relational fields* [29]. Examples of the latter are Euclidean distance (a ratio scaled relation between locations) and the visibility relation in a digital terrain model (a boolean relation between locations).

We excluded *event* for the current version of the algebra because it is only applied to static scenarios, and thus can be reinterpreted as object for this purpose. Note that every concept also has a temporal dimension which is left implicit here but can be added in future versions. Since our algebra focuses on transformations, we furthermore need to consider how core concepts can be quantified in terms of *amounts*. We use 'amount' here in a technical sense, to signify *extensive quantities* with a mereological (part-whole) structure that allows forming sums and which can be measured on a linear scale [32]. They correspond to well known geographic operations (cf. [33]). For example, an amount of space (a region) can be partitioned and summed into regions, all of which can be measured in terms of size. An amount of objects can be partitioned into subsets and can be measured by counting, or by summing up their qualities. An amount of moisture content is measured by integrating (summing up) a moisture field. We distinguish two principal types of amounts: *Content amounts* are amounts formed by controlling space and measuring its content (e.g., number of objects (e.g. households) or integral of a field (e.g. precipitation) within a defined region). *Coverage amounts* are formed by controlling content (e.g. collections of objects or field values) and measuring (the size of) the spatial regions covered (cf. [32, 33]). Examples would be the area covered by roads in a city, or the area covered by a certain type of vegetation. Finally, qualities of core concepts as well as measures of amounts can be on different *levels of measurement* [33].

2.3 Type inference

Type inference is a common feature of programming languages to perform static checks of *implementations*. In this work, we instead use it to infer *conceptual knowledge*. The use of type inference in knowledge bases is not new —

in fact, the Web Ontology Language (OWL) facilitates a form of it through `owl:allValuesFrom` constraints on RDF properties. [34] However, we use an independent type inference module to access features like type parameterization. To our knowledge, this has not been used before to produce metadata about workflows.

3 Methodology

3.1 Requirements and approach

To serve as a model for conceptual transformations and their purposes, our algebra needs to be:

- *Implementation-free*. This means it needs to generalize over different algorithmic realizations of GIS methods, such as different implementations of spatial interpolation. The primitives of our algebra therefore should be *implementation free*, similar to Computational Independent Models (CIM) in software engineering [35].
- *Functionally universal and precise*. The algebra needs to be universal enough to cover functionality that can occur in a workflow in practice, and at the same time precise enough to distinguish workflows that are suitable for a task from those that are not.

To meet these requirements, we combine the idea of core concepts with two technical approaches: For one, our operators' signatures represent *higher-order functions* to express operations in terms of other operations (cf. [27]). Since the algebra does not concern itself with implementation, powerful type inference features, including subsumption and bounded parametric polymorphism, can be incorporated without the complexity of providing a concrete procedure with the same level of generality. This allows us to handle a large variety of possible transformations with only a limited set of primitives. The other idea is *relational algebra* [28], which provides a general model for interpreting core concepts in terms of types of relations.

3.2 Overview

Fig. 3 gives an overview of the terminology used in the following. The methodological approach is shown in Fig. 4. The algebra was developed initially based on 12 GIS related tasks selected from GIS student exercises within several development cycles[1]. When we reached a first stable version, we formally specified a type model of core concepts based on relational algebra (Sect. 4). We then used this model to specify type signatures for algebraic operators which represent atomic conceptual transformations (Sect. 7), and which can be composed to derive more complex transformations that describe GIS functionality. Operators were incrementally extended throughout the work.

[1] When we talk about a GIS task, we mean a question concrete enough to be answered by a GIS workflow, such as: 'What is the average distance of buildings in Utrecht to the nearest hospital?', see https://figshare.com/s/c9d5c94599e83a5cdd97.

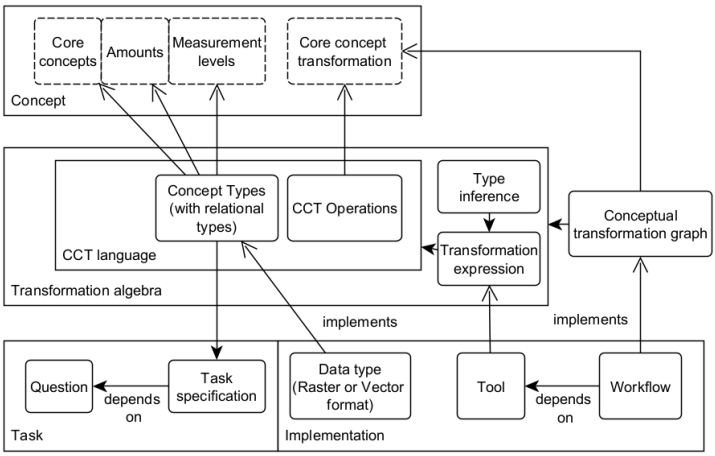


Fig. 3: Terminology used in the paper. The transformation algebra models concepts and their transformations (dotted lines), and is used to describe tasks and implementations. CCT is a particular vocabulary using the algebra syntax. Thick arrows denote dependencies, thin arrows denote implementation or modeling relations.

To serve as empirical basis for evaluation of the algebra, we collected *11 analytical scenarios* from online tutorials and GIS courses. We first manually described the underlying analytic task in terms of a directed acyclic graph between types, based on the task documentation (Sect. 5). We call this graph a *conceptual task specification* and, for brevity, we will sometimes refer to it as simply the *task*.

For each task, we then created one or more GIS workflows that implement them, and encoded these as graphs in the RDF format[36] (Sect. 6.1). In what follows, these graphs are what we will refer to simply as *workflows*.

All GIS tools occurring in the workflows were then described using transformation expressions (Sect. 6.1). All these steps were developed in an iterative manner over multiple development cycles.

We then developed a *type inference* system for the algebra (Sect. 6.2). This system allows us to compute the type of each step in the conceptual transformation that is implemented by a tool or workflow, by propagating type information from inputs to outputs.

We evaluated in two ways (Sect. 8): for one, we checked whether composing the transformations of individual tools into full workflows leads to type-correct transformation graphs. This tests whether tool combinations imagined by workflow authors are foreseen in our algebra model. Second, we performed retrieval tests for workflows using task specifications. The retrieval tests assess to what extent the conceptual transformations underlying analytic tasks match inferred workflow types, and whether they are distinctive enough to pick a

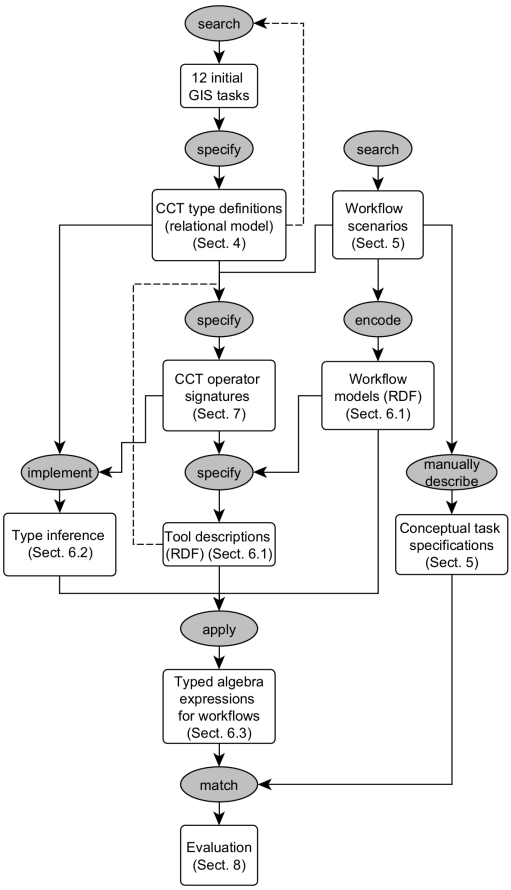


Fig. 4: Flowchart of methods used in this article. The dotted arrows indicate feedback in the development cycle.

valid workflow for a given task. A large scale evaluation on more workflows is still future work.

4 Types

In this section, we present core concepts of geographic information modeled as types of *relations*. Although the idea of representing and manipulating information in terms of relations is borrowed from relational algebra [28], its formal underpinnings are not essential for our current purposes. Where we recap relational algebra, we do so only to build an intuition for the conceptual types. Furthermore, we dismiss the idea that our types stand for data structures,

such as a database table. They are rather means of conceptualizing and manipulating geographic information, and thus vastly generalize over implemented data structures.

4.1 Values of measurement

A measurement or observation in GIS yields a *value*. These have a *type*, which represents simply the domain of values belonging to that type. Our class of value types, Δ_{V}, consists of the mutually exclusive types of spatial objects Obj, which can be understood as object names; locations Loc, which are populated by coordinate tuples; and quality values of measurement systems Qlt. The latter is further subdivided into a hierarchy of measurement levels, including the types of nominal (Nom), ordinal (Ord), interval (Itv), ratio (Ratio), count (Count) and boolean (Bool) values. Finally, we define the overarching type of values as V.

$$
\begin{aligned}
\Delta_{\mathsf{V}} &= \{\mathsf{Obj}, \mathsf{Loc}, \mathsf{Qlt}, \mathsf{Count}, \mathsf{Ratio}, \mathsf{Itv}, \mathsf{Ord}, \mathsf{Nom}, \mathsf{Bool}\}\,, \\
\mathsf{V} &= \bigcup \Delta_{\mathsf{V}}, \\
\mathsf{Count} &\subset \mathsf{Ratio} \subset \mathsf{Itv} \subset \mathsf{Ord} \subset \mathsf{Nom} \subset \mathsf{Qlt}, \\
\mathsf{Bool} &\subset \mathsf{Nom}.
\end{aligned}
$$

4.2 A relational model of spatial information

A *relation* associates some values with some other values. Formally, a relation $r = \langle A, K, \delta, B\rangle$ in the set of relations $\mathcal{R}$ consists of:

- A set of independent or *key attributes* $K = \{a_1, \cdots, a_n\}$, as well as dependent attributes $A - K = \{a_{n+1}, \cdots, a_m\}$. Together, they form A, the *heading* of the relation. The function $\delta : A \to \Delta$ assigns a type to each attribute.
- The *body* of the relation, $B \subseteq \delta(a_1) \times \cdots \times \delta(a_m)$, every tuple of which provides each attribute with an *attribute value* from the corresponding domain. Any combination of key attribute values may occur only once in the body, thus *uniquely identifying* a tuple of dependent attribute values.

First, note that the product type $\alpha_1 \times \cdots \times \alpha_n$ represents the type of tuples in which the value at index i is drawn from the type α_i. Secondly, we write ϵ for the unit type, which has only a single value, namely, the empty tuple.

With this in mind, we introduce the *type operator* R to express the type of a relation. It is parameterized by two types (one for the key and one for the dependent attributes), and is defined as the set of those relations in which the attribute values draw from corresponding types:

$$
\mathsf{R}(\alpha_1 \times \cdots \times \alpha_n,\ \ \alpha_{n+1} \times \cdots \times \alpha_m) \ = \ \{\ \langle K, A, \delta, B\rangle \in \mathcal{R} \mid
$$

$$
\begin{aligned}
&K = \{a_1, \cdots, a_n\},\\
&A = \{a_1, \cdots, a_m\},\\
&\forall_i \delta(a_i) = \alpha_i,\\
&B \subseteq \alpha_1 \times \cdots \times \alpha_m \ \}
\end{aligned}
$$

The class of relation types is Δ_{R}. The type hierarchy on value types induces a partial order on relation types, too, since sets of relations subset each other if their attribute types subset each other. For example, R(Loc, Itv) $\subset$ R(Loc, Qlt), so interval scaled fields are also fields.

Collections

The simplest kind of relation is called a *collection*. A collection is a *unary relation*, that is, a relation with only a single attribute, which is also the key. This simply corresponds to a *subset of values of the type of its attribute*. For example, R(Obj, ϵ) is the type of collections of objects. We write C(Obj) as shorthand. Furthermore, we use Reg, which stands for spatial regions like points, lines and areas, as a shorthand for C(Loc).

Unary concept types

Unary concept types have a single key attribute which uniquely identifies a relation tuple, and the other attribute is the only independent one. We call these types 'unary' because a single key serves as an index for the other attribute. This allows us to define concepts which combine values from more than one domain in order to refer to characteristics of some phenomenon. Unary concept types can serve as a model for particular *core concepts of spatial information*:

- *Spatial and location fields* R(Loc, Qlt) are interpreted here as relations controlling quality values by locations, e.g., a temperature field. A variant is a *location field* R(Loc, Loc), which is a field that controls locations by (neighboring) locations, e.g., when measuring drainage directions.
- *Inverted fields/coverage amounts* R(Qlt, Reg) map quality values into regions that are covered by this value, e.g. landuse coverages or contour regions. Note that in both cases, regions are controlled by quality values, but not vice versa (e.g., the region covered by more than 70 dB noise). The *sizes* of such a coverage are represented by the relation type R(Qlt, Ratio).
- *Object extents* R(Obj, Reg) and *object qualities* R(Obj, Qlt) are, respectively, relations of object values and their regions or quality values (denoting the space that the object, e.g. a house, occupies and the value of one of its qualities, e.g. its height).
- *Field sample/content amount* relations R(Reg, Qlt) denote either point-wise samples of a field, or content amounts which summarize values over some region. Content amounts are expressed by unary concepts that have regions as keys and amount measures as value. For example, the number of buildings within the boundaries of an arbitrary region is represented by the type

R(Reg, Count). A pointwise sample of precipitation would be modeled by R(Reg, Ratio).

Binary concept types

In contrast to unary concept types, which capture characteristics of single phenomena, binary concept types have *two* keys and a single dependent attribute. This allows us to talk about relationships *between* concepts, and to measure (quantify) some characteristics of these relationships. In particular, we draw attention to *quantified relations*, in which the dependent attribute is a quality. They also have an interpretation as core concepts:

- R(Obj × Obj, Qlt) captures the idea that a *spatial network* is a quantified relation between spatial objects. For example, the network might measure the amount of flow between pairs of cities, or it might capture information about whether a pair of train stations is connected by some train line (the latter being a boolean spatial network).
- The same idea can also be applied to locations as key pairs, as in R(Loc × Loc, Qlt). We call these concept types *relational fields*. A relational field quantifies relationships between locations. An example would be Euclidean *distance*. Another example, using Boolean quality values, would be *visibility*: is some location visible from another?

Multiple attributes

Finally, we have relations with a single key attribute and *multiple* independent attributes, for representing multiple characteristics of a single thing. For example, for representing a type that captures both footprint and height of buildings, we use the type R(Obj, Reg × Ratio). This reflects GIS layers with several attributes.

The fact that a data source or tool is associated with more than one attribute is incidental, as is the way in which they are bundled. We are dealing with concepts, not data structures, and so we could just as well have reversed the attributes or used a tuple of unary concepts. Ideally, the type parameter for the dependent attributes should not be a product but an intersection, so that order is irrelevant and so that R(x, $y \cap z$) is a supertype of R(x, y) and R(x, z). While this should be addressed in a future version, it does not currently lead to problems, because we adhere to a consistent way of annotating tools.

Simple example of a conceptual transformation

To illustrate the use of our types for describing conceptual transformations, take again the example from Fig. 1. The first step is to convert the contour map (which is an ordinally scaled coverage amount, since attributes denote interval ranges and polygons denote the area covered by these ranges) into a field. This would correspond to a transformation from R(Ord, Reg) to R(Loc, Ord). This transformation may, for example, be implemented by the *PolygonToRaster* tool. [37]

5 Tasks and workflows

We collected 11 documented workflow scenarios as an empirical basis for testing, taken from course material of a GIS minor[38], and from ESRI's ArcGIS online learning hub[39] and similar online tutorials. Scenarios are thematically diverse, complex and specific enough to be of practical relevance. For all online coursework, corresponding tutorials can be found under the indicated web resources.

We manually interpreted each scenario in terms of the underlying task, by providing a *task specification* in terms of a flowchart of CCT types. Starting from the input and feeding down into the goal concept types, the nodes and edges in these visual descriptions represent the high-level concept types and transformations inherent to the task *itself*, disregarding any tool that could be used to fulfill it. In the remainder, we will refer to task specifications as simply the *tasks*.

Additionally, we implemented one or two ArcGIS *workflows* for each scenario by following the tutorials and modifying them for different possible approaches.

Machine-readable task specifications and details of the workflows are available at the https://github.com/quangis/cct repository, under the `tasks/` and `workflows/` directories, respectively. The repository has additionally been archived at 10.6084/m9.figshare.19688712.[2]

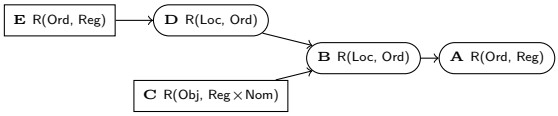
We will illustrate two scenarios, along with their task specification and associated workflows. Due to space constraints, the explanations of the remaining nine have been moved to Appendix A.

Scenarios 1a and 1b: Noise

What is the proportion of noise ≥ 70*dB in Amsterdam?* [38]

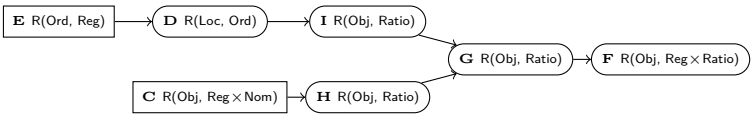
The task is to quantify traffic noise in Amsterdam. This scenario has two subscenarios. In the simplest one, NoisePortion, the noise contour map in Fig. 1a needs to be transformed into the area covered by noise $\geq$ 70dB. Alternatively, in the NoiseProportion variant, we normalize this area, e.g., by the area of objects, generating the proportion of the area covered by 70dB noise within each neighborhood.

For NoisePortion, we generate the coverage of noise (**A**) from some ordinal field (**B**) which was generated from a noise field (**D**) that was constrained to the spatial region of Amsterdam (**C**) and which originates from a noise contour map (**E**) (the latter with ordinally scaled noise intervals).



[2]Will be published after review.

For the *NoiseProportion* case, we generate a ratio scaled quality of neighborhoods (**G** and **F**), a proportion, of the sizes of neighborhood regions (**H**) and the sizes of the ordinal fields (**D**) within these neighbourhoods (**I**), originating from a noise contour map (**E**).



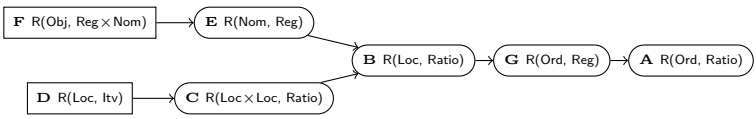
The NoisePortion implementation rasterizes the contour map, clips the raster to the spatial extent of the municipal polygon of Amsterdam, constrains the cell values (≥ 70) using local map algebra, and converts the constrained raster layer into a vector polygon.

There are different raster and vector solutions that implement *NoiseProportion*, as depicted in Fig. 5. NoiseProportionRaster is the raster version based on *local and zonal map algebra operations*. Here, a local map algebra tool (*Raster Calculator*) is used to constrain the noise field to 70 dB. *Zonal Statistics* is used to aggregate this field into the municipality polygon, the size of which is measured with *Add Geometry Attributes*. Note how the functionality of some tools is described on an aggregated level (supertool). *Overlay (Intersect)* and *Dissolve* can be used in combination to do an equivalent thing for vector data, which yields another workflow for the same task (NoiseProportionVector).

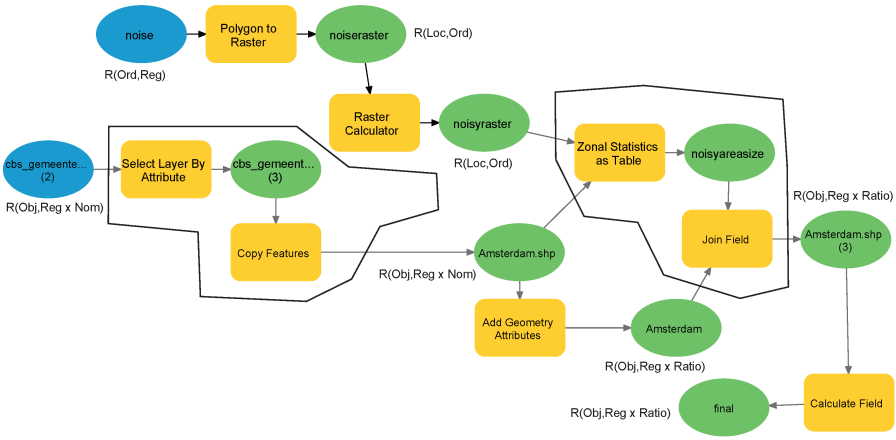
Scenario 9: Floods

What is the stream runoff during a predicted rainstorm in Vermont, US? [40]

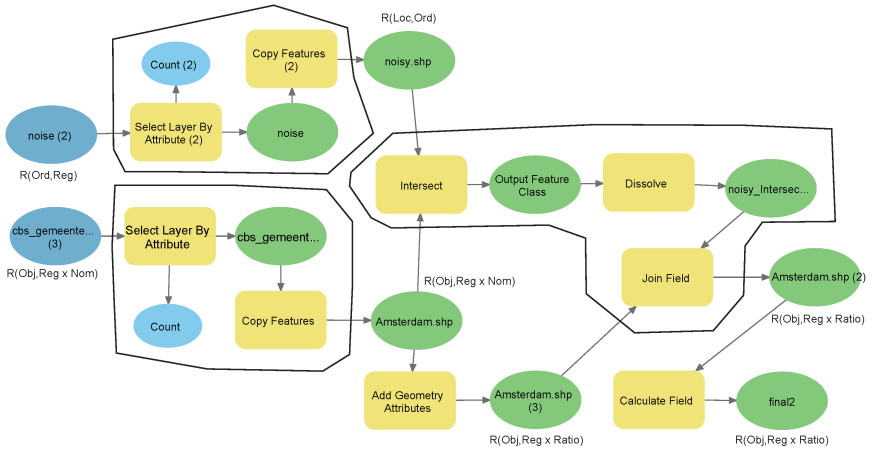
In this scenario, we estimate a unit hydrograph in order to predict floods in a catchment area. A unit hydrograph is a relation of areas draining within a given time interval, obtained using a digital elevation model within the catchment area. Conceptually, this corresponds to *generating a drainage time (isochrone) field from a height field, and inverting it into a coverage of the area draining within a given time interval*: from a terrain model **D** we derive a quantified (drainage time) relation between locations (**C**), which is minimized with respect to closest location in the region **E** of some pour point object (**F**). The resulting drainage time field (**B**) is classified into ordinal time intervals. Inverting this field yields (**G**) and measuring the area covered by each time interval yields the unit hydrograph (**A**).



The entire Floods workflow is implemented using various map algebra operations on raster maps.



(a) Computing noise proportion using Raster GIS (NoiseProportionRaster).



(b) Computing noise proportion using Vector GIS (NoiseProportionVector).

Fig. 5: Different ArcGIS workflows for implementing scenario 1b (computing noise proportion) using noise contours and municipalities as input (blue). Input/output types from tool annotations are added as labels for illustration purposes. Polygons circumscribe supertools (cf. Sect.6) .

6 Conceptual workflow description

In the previous sections, we introduced a formal framework in which we modelled core concepts as types (Sect. 4) and we described our tasks in terms of those types (Sect. 5). Now, before the next section, in which we provide the operators that comprise our geo-analytical vocabulary (Sect. 7), we first show how we combine such operators into transformation graphs. These

graphs describe tools, and are stitched together to describe full workflows. The conceptual type at each step in the graph is automatically inferred.

6.1 Workflows as tool applications

The workflows of Sect. 5 have been encoded as RDF graphs, linking tool application nodes to their inputs and output, as in Figure 6.

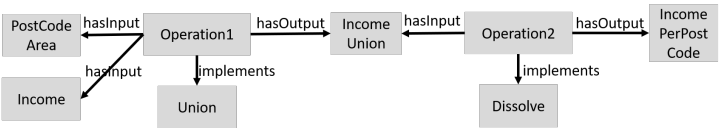


Fig. 6: Example of a workflow encoded as RDF triples. Arrows denote RDF properties.

The tool nodes themselves refer roughly to ArcGIS Pro or QGIS tools, which have been manually annotated with a *transformation expression*, which is a semantic signature that describes the underlying conceptual transformation. The annotations are available in the `tools/` directory of the repository. Note, however, that software tools and tool nodes do not correspond to each other 1-to-1. This is for the following reasons:

- A given tool might be interpreted into several variants of conceptual transformation, corresponding to its internal function and parameterizations.[3] For example, the zonal statistics tool might aggregate the size of the area covered by a raster *ZonalStatisticsSize*, but may also use average to aggregate over raster values *ZonalStatisticsMeanRatio*, as well as fields or objects for zone definitions. All of these choices correspond to different (tool-internal) conceptual transformations.
- Semantic descriptions were sometimes lifted to the level of *supertools*, standing for sub-workflows that can only be meaningfully interpreted into a core concept transformation *as a whole*. For example, the supertool *IntersectDissolve* stands for a combination of the *Intersect* and *Dissolve* tool. This implements the functionality of *IntersectDissolveField2Object*, which aggregates some vector representation of a field into some object quality. Comparing the algebraic description of this tool with *ZonalStatisticsSize*, it can be seen that it implements an identical core concept transformation, even though both tools are implemented on different data types and in various pieces of software.

[3]This should not be confused with simply having a polymorphic type. Whether a hammer is used to drive a nail into plywood or into a tree, the operation has the same purpose or function, though the type of result may differ. If it is used to break a plank instead, the function is a different one.

This illustrates that tool implementations do not necessarily align with conceptualizations of their function, which can be handled by separating semantic from technical tool descriptions.

6.2 Type inference

A transformation expression should provide rich descriptions of the tools used in a workflow, capturing the underlying functionality in detail. At the same time, it should be succinct, composed of a limited number of general operators.

Although the full meaning of atomic operators is not formally specified,[4], they do have an intended reading and a *type signature*.

For example, an operator for adding a value of type x to a collection of values of type x would have a signature $x \to \mathsf{C}(x) \to \mathsf{C}(x)$. This constrains its input, which can be enforced by using the type inference algorithm as a type checker, so that nonsensical transformations are rejected.

Moreover, the operators may generalize over various collections, unary concepts and quantified relations. Therefore, they are *polymorphic*, in that they may apply to values of more than one type, and *higher-order*, in that transformations may be described in terms of other transformations. The concrete type produced by a particular application is automatically inferred.

We have implemented the algorithm as a stand-alone Python module. This approach allows for integration in a broader ecosystem for workflow synthesis. It also enables flexibility, as our use case is atypical in its disregard for concrete implementation. The algorithm was loosely inspired by [41]; details in Appendix B. The code is freely available and documented at https://github.com/quangis/transformation-algebra.

Subtype polymorphism

A type is a domain to which a value or relation belongs: if $a \in \alpha$, then $a : \alpha$. Therefore, we immediately obtain a type hierarchy that mirrors subset relations. Because $\mathsf{Count} \subset \mathsf{Ratio}$, for example, we have that $x : \mathsf{Count}$ implies $x : \mathsf{Ratio}$.

Our algorithm accommodates *subtype polymorphism* in the sense that an operator of type $\alpha \to \beta$ will also accept any input of type α^- such that $\alpha^- \subseteq \alpha$, and will produce a value of any type β^+ such that $\beta \subseteq \beta^+$. The most specific possible type is selected.

Parametric polymorphism

However, to define transformations that work on multiple types that are not merely subtypes of each other, we additionally accommodate *parametric polymorphism*. That is to say: signatures may contain type variables.

[4]However, for some operators, a straightforward semantics can be borrowed from relational algebra or functional programming. Furthermore, our algebra allows for *lambda abstraction* to define new functions from primitives. Since we do not focus in this article on handling such definitions, we kept this aspect out of scope.

Such a *schematic type* may be further *bounded* by a typeclass. Because types are not associated with specifically defined behaviour, there is no need to specify a typeclass beyond the set of types it comprises. It also implicitly accommodates *functional dependencies*: if we know that $\mathsf{R}(\alpha,\ \beta)$ is bounded by a finite typeclass like $\{\mathsf{R}(\mathsf{Obj},\ \mathsf{Count}), \mathsf{R}(\mathsf{Reg},\ \mathsf{Ratio})\}$, then once we find out that $\alpha = \mathsf{Obj}$, we can immediately conclude that $\beta = \mathsf{Count}$.

For a schematic type $\alpha \to \beta$ in which α is bound by the typeclass $\mathcal{C}$, we write $\alpha \in \mathcal{C} \implies \alpha \to \beta$.

6.3 Workflows as transformations

The transformation expression for the *SelectLayerByObjectTessObjects* tool is given below as an example. This tool is the first step in the NoisePortion workflow, selecting neighbourhoods of Amsterdam.

$$\mathsf{subset}\ (- : \mathsf{C}(\mathsf{Obj}))\ (1 : \mathsf{R}(\mathsf{Obj},\ \mathsf{Reg} \times \mathsf{Nom}))$$

In this expression, the tool's first and only input, denoted 1, which must be a subtype of $\mathsf{R}(\mathsf{Obj},\ \mathsf{Reg} \times \mathsf{Nom})$, is subset by some other collection of objects $\mathsf{C}(\mathsf{Obj})$. While this other collection is a conceptual input to the operator, it is not associated with a concrete input to the tool, because it is only implicitly inserted by the author of the workflow. Therefore, it is left unspecified and denoted $-$.

We will learn in Sect. 7 that the subset operator has type $\mathsf{C}(x) \to \mathsf{R}(k,\ v) \to \mathsf{R}(k,\ v)$ where x occurs in k or v. With this knowledge, we can infer the output type of this specific *application* of the operation, and, in turn, of the tool. For example, if input 1 has type $\mathsf{R}(\mathsf{Obj},\ \mathsf{Reg} \times \mathsf{Count})$, the output must be a value of that type too. Moreover, if its input has an incongruent type, like $\mathsf{R}(\mathsf{Loc},\ \mathsf{Obj})$, we can immediately reject it.

A transformation expression can be represented as a tree. By connecting the trees for every tool application in a workflow, and running type inference alongside, we synthesize the transformation graph for an entire workflow.

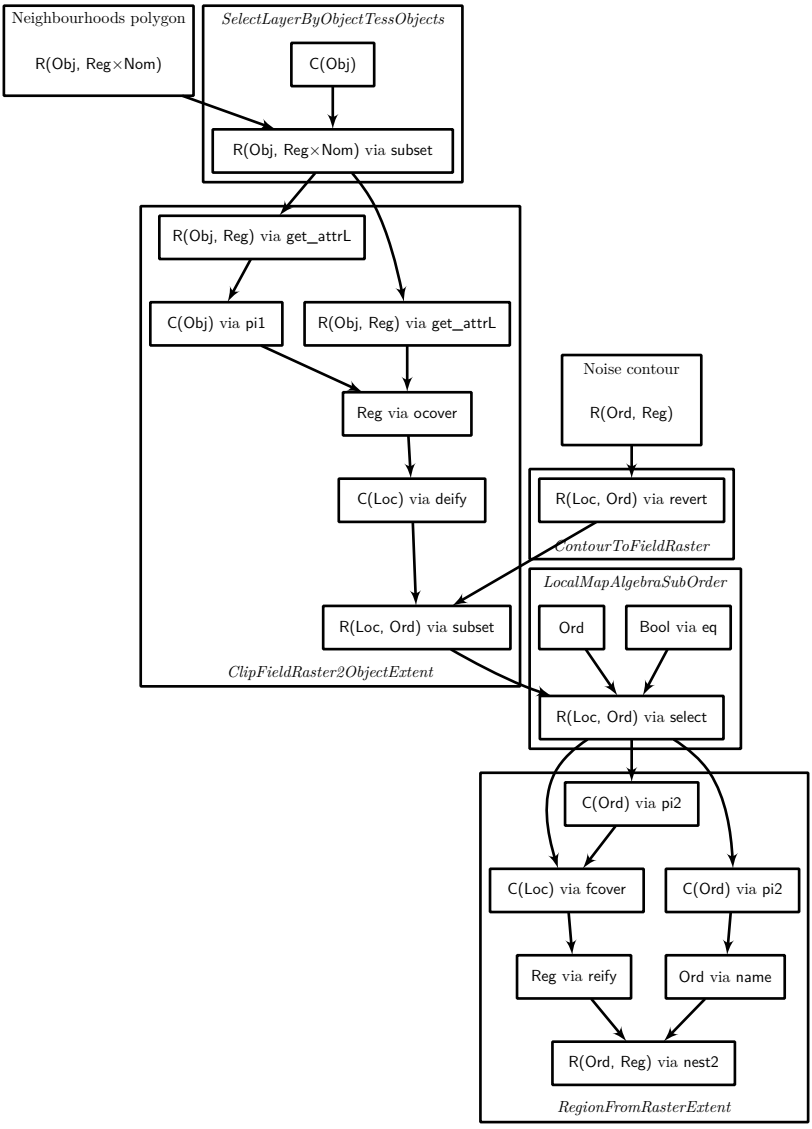
Figure 7 illustrates the resulting transformation graph for the NoisePortion workflow. We will describe the meaning of the operators in the next section.

7 Geo-analytical transformations

We specify geo-analytical transformations in terms of the operators introduced in this section. We give a complete overview of the operators in order to illustrate the range of GIS functionality that can be captured with them. However, to understand our general approach, it is not essential to understand the particularities of each, especially since operators are subject to continuous revision and in this article, we used them only to produce types.[5]

[5] The operators themselves could also be used as semantic markers in the task specifications (see discussion).

Fig. 7: Simplified procedural annotations for the NoisePortion workflow.



Keep in mind that an expression describes transformations on a conceptual level, that is, in the mind of an analyst guiding computational procedures. Consequently, the operators generalize over data types like Raster and Vector, and they may be used to describe a diversity of tools and implementations. Conversely, the operators allow for many ways in which a single tool can be described. Nevertheless, we expect that pertinent concepts will show up in any sensible description of a tool.

All operators in this section have a machine-readable counterpart in the `cct.py` file in the repository.

7.1 Transformations of geo-analytic values

Value derivations

$$\begin{aligned} \mathsf{objectify} &: \mathsf{Nom} \to \mathsf{Obj} & \mathsf{nominalize} &: \mathsf{Obj} \to \mathsf{Nom} \\ \mathsf{leq} &: \mathsf{Ord} \to \mathsf{Ord} \to \mathsf{Bool} & \mathsf{eq} &: \mathsf{V} \to \mathsf{V} \to \mathsf{Bool} \\ \mathsf{and} &: \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} & \mathsf{not} &: \mathsf{Bool} \to \mathsf{Bool} \\ \mathsf{ratio}, \mathsf{product} &: \mathsf{Ratio} \to \mathsf{Ratio} \to \mathsf{Ratio} & \mathsf{classify} &: \mathsf{Itv} \to \mathsf{Ord} \end{aligned}$$

The objectify and nominalize convert between object identifiers and names. classify provides a way to reclassify interval scaled values to ordinal classes, a typical GIS operation implemented in reclassification tables. The other operators have obvious meanings (with eq for equality and leq for less-than-or-equal).

Aggregations of collections

$$\begin{aligned} \mathsf{count} &: \mathsf{C}(\mathsf{Obj}) \to \mathsf{Count} & \mathsf{size} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{Ratio} \\ \mathsf{merge} &: \mathsf{C}(\mathsf{Reg}) \to \mathsf{Reg} & \mathsf{name} &: \mathsf{C}(\mathsf{Nom}) \to \mathsf{Nom} \\ \mathsf{centroid} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{Loc} \end{aligned}$$

These are operations that aggregate collections, with straightforward meanings, such as the centroid of a collection of locations. name can be used to grant a single name to a collection of landuse types, or to a collection topological relation values.

Statistical summaries

$$\begin{aligned} \mathsf{avg} &: \mathsf{R}(\mathsf{V}, \mathsf{Itv}) \to \mathsf{Itv} & \mathsf{min} &: \mathsf{R}(\mathsf{V}, \mathsf{Ord}) \to \mathsf{Ord} \\ \mathsf{sum} &: \mathsf{R}(\mathsf{V}, \mathsf{Ratio}) \to \mathsf{Ratio} & \mathsf{max} &: \mathsf{R}(\mathsf{V}, \mathsf{Ord}) \to \mathsf{Ord} \end{aligned}$$

Note that these operations are on unary concepts, not collections, since in order to capture the concept of a statistical distribution, we need to be able to control values by other values. For example, temperature measurements can be controlled by different locations.

7.2 Geometric transformations

Constructors for binary concepts

$$\begin{aligned} \mathsf{lDist} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{C}(\mathsf{Loc}) \to \mathsf{R}(\mathsf{Loc} \times \mathsf{Loc}, \mathsf{Ratio}) \\ \mathsf{lTopo} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{Reg} \to \mathsf{R}(\mathsf{Loc} \times \mathsf{Reg}, \mathsf{Nom}) \\ \mathsf{loDist} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{R}(\mathsf{Obj}, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Loc} \times \mathsf{Obj}, \mathsf{Ratio}) \end{aligned}$$

$$\begin{aligned}\mathsf{oDist} &: \mathsf{R}(\mathsf{Obj}, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Obj}, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Obj} \times \mathsf{Obj}, \mathsf{Ratio})\\ \mathsf{loTopo} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{R}(\mathsf{Obj}, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Loc} \times \mathsf{Obj}, \mathsf{Nom})\\ \mathsf{oTopo} &: \mathsf{R}(\mathsf{Obj}, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Obj}, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Obj} \times \mathsf{Obj}, \mathsf{Nom})\end{aligned}$$

We start with operations to *construct* binary concepts. lDist generates distance relations between locations, and lTopo generates topological relations between locations and regions, for example, whether a point is inside, outside or bordering a region, corresponding to the point set basis of the 9 intersection model. [42]

Related versions of distance and topological constructors for objects and regions are derived from this. In a similar way, further variants of topological constructors between regions and objects (lrTopo, rTopo, orTopo) can be specified.

$$\begin{aligned}\mathsf{nbuild} &: \mathsf{R}(\mathsf{Obj}, \mathsf{Reg} \times \mathsf{Ratio}) \to \mathsf{R}(\mathsf{Obj} \times \mathsf{Obj}, \mathsf{Ratio})\\ \mathsf{nDist} &: \mathsf{C}(\mathsf{Obj}) \to \mathsf{C}(\mathsf{Obj}) \to \mathsf{R}(\mathsf{Obj} \times \mathsf{Obj}, \mathsf{Ratio}) \to \mathsf{R}(\mathsf{Obj} \times \mathsf{Obj}, \mathsf{Ratio})\end{aligned}$$

The next operations build spatial distance networks from objects with impedance qualities, and measure distances between objects on a spatial network. The latter needs to be fed with a spatial network as input.

$$\begin{aligned}\mathsf{lVis} &: \mathsf{C}(\mathsf{Loc}) \to \mathsf{C}(\mathsf{Loc}) \to \mathsf{R}(\mathsf{Loc}, \mathsf{Itv}) \to \mathsf{R}(\mathsf{Loc} \times \mathsf{Loc}, \mathsf{Bool})\\ \mathsf{gridgraph} &: \mathsf{R}(\mathsf{Loc}, \mathsf{Loc}) \to \mathsf{R}(\mathsf{Loc}, \mathsf{Ratio}) \to \mathsf{R}(\mathsf{Loc} \times \mathsf{Loc}, \mathsf{Ratio})\\ \mathsf{lgDist} &: \mathsf{R}(\mathsf{Loc} \times \mathsf{Loc}, \mathsf{Ratio}) \to \mathsf{C}(\mathsf{Loc}) \to \mathsf{C}(\mathsf{Loc})\end{aligned}$$

lVis measures visibility between locations, given some terrain model as a field. It is the basis of *visibility analysis* in GIS. gridgraph builds a network of locations from a location field and an impedance field (where field impedance values are taken as network qualities), and lgDist measures distances on such a location network. These functions are fundamental for global map algebra, such as runoff modeling on a terrain model.

Conversions

$$\begin{aligned}\mathsf{invert} &: \mathsf{R}(\mathsf{Loc}, x) \to \mathsf{R}(x, \mathsf{Reg})\\ \mathsf{revert} &: \mathsf{R}(x, \mathsf{Reg}) \to \mathsf{R}(\mathsf{Loc}, x)\\ \mathsf{getamounts} &: x \subseteq \mathsf{QIt} \implies \mathsf{R}(\mathsf{Obj}, \mathsf{Reg} \times x) \to \mathsf{R}(\mathsf{Reg}, x)\end{aligned}$$

invert and revert convert fields into contours (regions denoting some field value interval) and nominal coverages (regions denoting homogeneous nominal field values) and back. getamounts obtains content amounts (e.g. number of schools in some region) from object qualities (number of schools in Utrecht).

Interpolation and buffering

$$\begin{aligned}\mathsf{extrapol} &: \mathsf{R}(\mathsf{Obj},\ \mathsf{Reg}) \to \mathsf{R}(\mathsf{Loc},\ \mathsf{Bool})\\ \mathsf{interpol} &: \mathsf{R}(\mathsf{Reg},\ \mathsf{Itv}) \to \mathsf{C}(\mathsf{Loc}) \to \mathsf{R}(\mathsf{Loc},\ \mathsf{Itv})\end{aligned}$$

Based on distance relations between locations, we can define a boolean field that indicates where a certain distance value applies. Buffers, in addition, turn this boolean field into a region. The operation called extrapol is the basis for the generation of buffers. Note that the operation needs a distance parameter which is left implicit here.

Point interpolation uses some field sample (e.g. point-wise measures) to estimate a field that is at least interval scaled, within a collection of locations (extent).

Map algebra

$$\begin{aligned}\mathsf{slope} &: \mathsf{R}(\mathsf{Loc},\ \mathsf{Itv}) \to \mathsf{R}(\mathsf{Loc},\ \mathsf{Ratio})\\ \mathsf{aspect} &: \mathsf{R}(\mathsf{Loc},\ \mathsf{Itv}) \to \mathsf{R}(\mathsf{Loc},\ \mathsf{Ratio})\\ \mathsf{flowdirgraph} &: \mathsf{R}(\mathsf{Loc},\ \mathsf{Itv}) \to \mathsf{R}(\mathsf{Loc},\ \mathsf{Loc})\\ \mathsf{accumulate} &: \mathsf{R}(\mathsf{Loc},\ \mathsf{Loc}) \to \mathsf{R}(\mathsf{Loc},\ \mathsf{C}(\mathsf{Loc}))\end{aligned}$$

Local, focal and global map algebra turns fields into other fields based on coinciding locations, moving window locations, or by searching over all locations. In this paper, we consider primitives for computing the focal operations slope, aspect and flowdirgraph (flow direction from a height field), as well as the global function accumulate which aggregates a location network over all connected locations into a field of collections of locations reachable from a given location. Further map algebra functions [20] can be introduced in this way, but are not needed for our workflows.

7.3 Amount operations

$$\begin{aligned}\mathsf{fcont} &: x, y \subseteq \mathsf{QIt} \implies (\mathsf{R}(\mathsf{V},\ x) \to y) \to \mathsf{R}(\mathsf{Loc},\ x) \to \mathsf{Reg} \to y\\ \mathsf{ocont} &: \mathsf{R}(\mathsf{Obj},\ \mathsf{Reg}) \to \mathsf{Reg} \to \mathsf{Count}\\ \mathsf{fcover} &: x \subseteq \mathsf{QIt} \implies \mathsf{R}(\mathsf{Loc},\ x) \to \mathsf{C}(x) \to \mathsf{Reg}\\ \mathsf{ocover} &: \mathsf{R}(\mathsf{Obj},\ \mathsf{Reg}) \to \mathsf{C}(\mathsf{Obj}) \to \mathsf{Reg}\end{aligned}$$

Finally, we discuss operations to summarize the content of collections of objects and fields as *amounts*. We distinguish operations that summarize a field quality, e.g. by estimating an integral (fcont), or by summing up objects (ocont). These amounts are called *content amounts*. Vice versa, starting with value and object collections, we can also measure the area covered by a field quality (fcover) and by a collection of objects (ocover), respectively. The latter are called *coverage* amounts.

7.4 Relational operators

The following operators take inspiration from relational algebra and functional programming. Their interpretation is not geographical, but rather, they are used to combine geographical operations. Their type signatures are therefore quite general. There are many different operators we could have used, and they can be reduced by defining them in terms of eachother. The choices we make here affect the 'resolution' of our transformations.

Set operators

$$\begin{aligned}
\mathsf{relunion} &: r \in \Delta_{\mathsf{R}} \implies \mathsf{C}(r) \to r & \mathsf{get} &: \mathsf{C}(x) \to x \\
\mathsf{set_diff} &: r \in \Delta_{\mathsf{R}} \implies r \to r \to r & \mathsf{add} &: v \to k \to \mathsf{R}(k,\, v) \to \mathsf{R}(k,\, v) \\
\mathsf{prod3} &: \mathsf{R}(z,\, \mathsf{R}(x,\, y)) \to \mathsf{R}(x \times z,\, y) & \mathsf{nest} &: x \to y \to \mathsf{R}(x,\, y) \\
\mathsf{inrel} &: x \to \mathsf{C}(x) \to \mathsf{Bool}
\end{aligned}$$

These operators are inspired by counterparts from set theory: prod3 builds a Cartesian product from a nested relation, which results in a binary concept. nest allows us to generate singular relations and and get gets some value out of a collection. add adds elements to collections, and inrel tests whether some element is contained in a collection.

Projection operator

The projection operators project a given relation to its i'th attribute, resulting in a collection. We define the typeclass $\mathcal{C}_i(x) = \{\alpha_0 \times \cdots \times \alpha_n \mid \alpha_i = x\}$ for those tuple types that contain x as it's i'th operand.

$$\mathsf{pi_i} : k \times v \in \mathcal{C}_i(x) \implies \mathsf{R}(k,\, v) \to \mathsf{C}(x)$$

Selection operator

We may select a subset of a relation using a constraint on attribute values, using some binary comparison operator, like equality. The subset is a particular use of this selection, using inrel to determine whether a value is contained in a collection.

$$\begin{aligned}
\mathsf{select} &: (k \times v \to \mathsf{Bool}) \to \mathsf{R}(k,\, v) \to \mathsf{R}(k,\, v) \\
\mathsf{subset} &: k \times v \in \bigcup_i \mathcal{C}_i(x) \implies \mathsf{R}(k,\, v) \to \mathsf{C}(x) \to \mathsf{R}(k,\, v)
\end{aligned}$$

Join operators

$$\begin{aligned}
\mathsf{join} &: \mathsf{R}(x,\, y) \to \mathsf{R}(y,\, z) \to \mathsf{R}(x,\, z) & \mathsf{get_attrL} &: \mathsf{R}(x,\, y \times z) \to \mathsf{R}(x,\, y) \\
\mathsf{join_attr} &: \mathsf{R}(x,\, y) \to \mathsf{R}(x,\, z) \to \mathsf{R}(x \times y,\, z) & \mathsf{get_attrR} &: \mathsf{R}(x,\, y \times z) \to \mathsf{R}(x,\, z)
\end{aligned}$$

We introduce an operator for the *natural join* of two unary concepts, and for constructing multiple attributes from two unary concepts and vice versa.

$$\begin{aligned}
\mathsf{groupbyL} &: r \in \{\mathsf{C}(x), \mathsf{R}(x,\, q)\} \implies (r \to q') \to \mathsf{R}(x \times y,\, q) \to \mathsf{R}(x,\, q') \\
\mathsf{groupbyR} &: r \in \{\mathsf{C}(y), \mathsf{R}(y,\, q)\} \implies (r \to q') \to \mathsf{R}(x \times y,\, q) \to \mathsf{R}(y,\, q') \\
\mathsf{groupby} &: q \subset \mathsf{QIt} \implies (\mathsf{C}(k) \to q) \to \mathsf{R}(k,\, v) \to \mathsf{R}(v,\, q)
\end{aligned}$$

This operator groups quantified relations by the left (right) key, summarizing lists of quality values with the same key value into a new value per key, resulting in a new unary concept. For example, using the function avg, we can summarize relational fields by their left (right) location. Another variant of this operator is used to summarize keys of *unary concepts* by their foreign keys on the right hand side.

Operators on functions

$$\begin{aligned}
\mathsf{compose} &: (b \to c) \to (a \to b) \to (a \to c) \\
\mathsf{id} &: x \to x \\
\mathsf{compose2} &: (c \to d) \to (a \to b \to c) \to (a \to b \to d) \\
\mathsf{swap} &: (x \to y \to z) \to (y \to x \to z) \\
\mathsf{apply} &: (x \to y) \to \mathsf{C}(x) \to \mathsf{R}(x,\, y) \\
\mathsf{apply1} &: (x \to y) \to \mathsf{R}(a,\, x) \to \mathsf{R}(a,\, y) \\
\mathsf{apply2} &: (x \to y \to z) \to \mathsf{R}(a,\, x) \to \mathsf{R}(a,\, y) \to \mathsf{R}(a,\, z) \\
\mathsf{prod} &: (x \to y \to z) \to \mathsf{R}(a,\, x) \to \mathsf{R}(b,\, y) \to \mathsf{R}(a,\, \mathsf{R}(b,\, z)) \\
\mathsf{join_key} &: r \in \{\mathsf{R}(x,\, q_2), \mathsf{R}(y,\, q_2)\} \implies \mathsf{R}(x \times y,\, q_1) \to r \to \mathsf{R}(x \times y,\, q_2)
\end{aligned}$$

We use higher-order functions known from functional programming, like function composition, argument swapping, the identity function, and the apply operators to map a function to each member of a relation. prod combines two unary concepts using a binary function. This operator is fundamental to compute any quantified relations from unary concepts, like distance relations between object regions. Finally, join_key substitutes the quality of a quantified relation with some quality of one of its keys.

8 Evaluation

To support the claim that our algebra captures properties that are conceptually relevant for transforming spatial information, we test whether the building blocks of our language are *general* enough to allow for the various tool decisions made by workflow authors, yet *specific* enough to allow for distinguishing workflows based on independent specifications of the underlying tasks.

8.1 Criteria of evaluation

Type correctness of workflows

The type inference algorithm addresses the first concern, by ensuring that the computed output type of every tool application in a workflow is subsumed by the stated input type of its successor. This shows that our algebraic descriptions are at least general enough to accommodate the ways in which the tools are applied in practice.

Descriptive power of workflow annotations

To address the second concern, we matched the expected types of the task specification to the types in the transformation graph as automatically constructed from the tool descriptions.

A type in the workflow is considered a match if it specifies a strict subtype of a type in the task, but not vice versa. A task description that does not match a workflow for the associated task produces a false negative; one that matches a different workflow makes a false positive. The latter case is, however, not necessarily indicative of a mistake: two workflows might simply be conceptually similar enough to serve as an approach for the same task.

The resulting degree of *precision* (i.e. relevant matches as a fraction of all matches) is an indicator for whether our task descriptions are specific enough to distinguish concrete workflows. The resulting degree of *recall* (i.e. relevant matches as a fraction of all relevant workflows) is an indicator for whether the tool descriptions and their constituent operators are adequately rich in information, and whether the subsequently inferred types are correct and at least as specific as those in the task description.

We perform several variants of this retrieval test to investigate whether automatically annotating *internal* concept transformations, as we do, adds essential information that would not be available if either the entire workflow, or its constituent tools, were treated as 'black box' transformations between more general concept types. That is, we test at three levels:

1. We test whether the source and goal types of the task specification match that of the input and output of the workflow as a whole.
2. Additionally, we test whether all types that occur in the task specification occur in the annotations *as inputs or outputs of tools*.
3. Finally, we test whether they occur *anywhere*, even internally.

At each level, we measure the effect of transformation order and type inference:

- We test the effect of matching types in the chronological *order* specified in the flowchart.
- We can compute internal types and output types of a tool based either on the most general type that a tool can work with, or on the type of the

Table 1: Quality of the matching between task and workflow. Where it made a difference, results for the loosened *Floods* task specification are in parentheses.

Level	Order	Pass	Type-I	Type-II	Precision	Recall
1. Workflows	n/a	☐	8	1	0.571	0.923
	n/a	☑	8	0	0.591	1.000
2. Tools	☐	☐	0	11	1.000	0.154
	☑	☐	0	11	1.000	0.154
	☐	☑	0	10	1.000	0.231
	☑	☑	0	10	1.000	0.231
3. Internal	☐	☐	2	2 (1)	0.846 (0.857)	0.846 (0.923)
	☑	☐	2	2 (1)	0.846 (0.857)	0.846 (0.923)
	☐	☑	2	1 (0)	0.857 (0.867)	0.923 (1.000)
	☑	☑	2	1 (0)	0.857 (0.867)	0.923 (1.000)

Table 2: Level 3 results with ordering and passthrough. Tasks are on the columns, workflows are on the rows. An full circle indicates a match, an empty one a mismatch. Crossing out means that the (mis)match is erroneous.

	1a	1b	2	3	4	5	6	7	8	9	10
NoisePortion	●	○	○	○	○	○	○	○	○	○	○
NoiseProportionRaster	○	●	○	○	○	○	○	○	⊗	○	○
NoiseProportionVector	○	●	○	○	○	○	○	○	⊗	○	○
Population	○	○	●	○	○	○	○	○	○	○	○
Temperature	○	○	○	●	○	○	○	○	○	○	○
HospitalsNear	○	○	○	○	●	○	○	○	○	○	○
HospitalsNetwork	○	○	○	○	●	○	○	○	○	○	○
Deforestation	○	○	○	○	○	●	○	○	○	○	○
Solar	○	○	○	○	○	○	●	○	○	○	○
RoadAccess	○	○	○	○	○	○	○	●	○	○	○
Aquifer	○	○	○	○	○	○	○	○	●	○	○
Floods	○	○	○	○	○	○	○	○	○	⊗	○
Malaria	○	○	○	○	○	○	○	○	○	○	●

actual input data,[6] which may be more specific. In the former case, the types produced by a tool are the same no matter the context in which it is applied; in the latter, types are inferred for each particular tool application. We test the extent to which this *passthrough* affects the result?

Table 1 shows the retrieval quality for each variant. We must note here that we additionally report the results with a modification of the *Floods* task, in which the one concept type that hinders retrieval is dropped. Doing so provides valuable information for the discussion.

These results can be reproduced with the tools in the CCT repository.

[6]This is a simplification, as source data has not been annotated with types: types are only passed between the output of one tool to the next. This does not make a difference for our dataset, but should be addressed when scaling up.

8.2 Discussion of results

Observe that recall is very good if we treat workflows as a black box, but precision suffers. This makes sense: we can assume that source inputs and final output of a workflow will conform to the task's sources and goal, but it casts too wide a net to filter out false positives (type-I errors).

Conversely, at level 2, in which we look for the task's intermediate concepts and expect them to be directly produced by the corresponding workflow's tools, we get rid of those false positives — and gain many false negatives (type-II errors).

To get the best of both worlds, we must capture more of the procedural knowledge available in the task. Once we ascend to level 3, we can access the concept types 'inside' the tools. As a result, recall and precision significantly improve, although neither to the point of perfection.

To investigate why recall is not perfect, we turn to the specification of the *Floods* task. In it, the final transformation between **A** and **B** represents a grouping of a drainage time field by area size, during which we expected the concept R(Ord, Reg) (the region covered by a time interval). As it turns out, this intermediate type is never recorded in the transformation graph, as it is hidden in the operation groupby.

This shows that the design decisions of the algebra have an effect on the accuracy of results. The intention behind requesting the intermediate type R(Ord, Reg) was that the output of the unit hydrograph workflow should measure the size of the *regions* covered by some interval of a drainage time field. However, locations are aggregated only inside the groupby operation, which is a *primitive* in the CCT algebra. This shows that the 'resolution' of operators has an effect on accuracy. The smaller the atomic steps, the better conceptual transformations may be distinguished, leaving room for future improvement. Furthermore, while current task specifications were done with types alone, more precise retrieval could be be achieved by including the operators themselves (size in this example). In a follow-up study, we intend to provide internal structure to operators, as well as examine the structure of task specifications.

Perfect precision is likewise prevented by a single task: *Aquifer*, which, in addition to the Aquifer workflow, also triggers the retrieval of the workflows NoiseProportionVector and -Raster. The associated tasks are indeed superficially similar, in that they both require turning (contour) coverages into fields and output objects. However, in the case of *Aquifer*, these objects are only selected, whereas in the *NoiseProportion* case, we aggregate fields within the extent of the object. To differentiate this, we would either need to specify operations in the task, or distinguish nominal values that are explicitly *not* ordinal (called *plain*-nominal in [15]). Both can be done in future versions.

Taking into account the ordering of the task specification did not change the results in our sample, for better or for worse. Evidently, unordered types are usually enough to disambiguate our workflows, so less detailed procedural annotations would have been sufficient. However, more subtle differentiation

may be needed if the workflow repository grows larger. In any case, it is worth noting that the expected ordering really is present in each workflow.

Passthrough does have an effect. This is fully due to a single workflow, NoisePortion. In Figure 7, we can see the reason. Its final tool, *RegionFromRasterExtent*, can be applied very generally: it takes a R(Loc, Nom) and produces a R(Nom, Reg). However, by taking advantage of type inference, we can deduce that *in this instance*, a value of the more specific type R(Ord, Reg) is produced. Again, future work should reveal whether similar behaviour is common in larger workflow repositories.

As you can see in Table 2, task specifications are agnostic about whether distances are measured in a Euclidean manner or over a network for *Hospital*'s workflows, and about whether noise proportions are measured using raster or vector formats for *NoiseProportion*'s workflows. We correctly retrieve the two pairs of workflows that implement a single task in different formats, ignoring the technicalities of software implementation.

Caveats

Our results make a credible case for the claim that annotating conceptual-procedural knowledge may open new avenues to avoid the limitations of describing GIS workflows via implementation details. However, it remains an open question whether the particular vocabulary of operators we chose present a satisfactory abstraction of GIS. To improve this, our preliminary algebra should be condensed and tested with further tool applications and workflow examples.

The strength of the evidence that the operators accommodate the decisions of workflow authors is somewhat further limited by the fact that the operators and tool descriptions have been, inevitably, created with knowledge of these workflows. Similarly, because the task specifications require expert knowledge that is in limited supply, they have been provided by the same authors who produced the workflows' tool descriptions. Therefore, although care has been taken to separate the two specifications, we cannot rule out that knowledge of the tools has influenced the specification of tasks. These caveats are unavoidable at this point. In the future, to provide a more rigorous evaluation, we plan to describe independent workflows using the same tools, and thus to specify tasks independently of the associated workflows.

Finally, the expert knowledge required to produce transformation expressions is non-trivial. In this article, our annotations were done by two of the authors in several rounds, which showed that annotator agreement is initially low and improves only based on clear instructions. To generate a larger knowledge base across various experts, detailed *algebraic annotation instructions* are therefore needed. A particular challenge in this respect lies in the *ambiguity* of interpreting data in concepts, which should be avoided for the purpose of retrieval. For example, whether to interpret a contour map as coverage amount or as a field is a question that requires scrutiny. This requires more research on geodata conceptualizations.

9 Conclusion and outlook

We have introduced a relational model of core concepts of geographic information, which can be used to specify tasks in terms of conceptual transformations underlying GIS workflows. Our algebra provides a vocabulary to specify transformations between these concepts and to automatically add procedural knowledge to geographic workflows using type inference. This allows us to effectively retrieve workflows for a given task, disregarding data formats and implementation details.

Our tests suggest that there is merit to this approach, by showing that algebraic annotations and inferred types provide important but implicit procedural knowledge associated with an analytical GIS task. We have implemented a generic toolset to facilitate the production of procedural annotations in GIS as a basis for distinguishing workflows. Situated in the well-established ecosystems of Python and RDF, this modular toolset includes an algorithm for type inference, a parser to turn workflows with annotated tools into rich RDF graphs, and a method to query those graphs. The toolset is universal in that it can be applied to any other knowledge domain in which procedures can be captured as function signatures.

Building on this proof of concept, future work should focus on improving the operator vocabulary and transformation expressions, creating independent workflows and tool annotations, and increasing the size of the workflow repository.

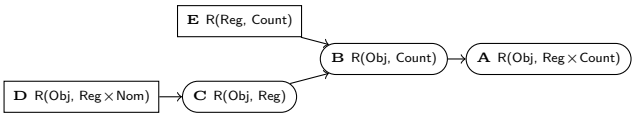
There are several possible areas of application of our algebra as a metalanguage. First, the algebra serves as a description and retrieval language for workflows that allows querying over different implementations. Second, the algebra provides a way to specify tasks, and thereby serves to account for the purposes of analysis. A language to capture the different purposes of geo-analytical transformations is currently lacking in GIScience [43]. In a similar way, the algebra provides a missing link between workflows and the questions they answer, by tying together the loose ends of recent work on grammatical interpretation of geo-analytical questions in terms of conceptual transformation graphs [44]. Finally, the algebra could also be used in the future to describe and compare software systems by measuring their conceptual similarity.

Appendix A Tasks and workflows

Scenario 2: Population

What is the number of inhabitants for each neighborhood in Utrecht? [38]

The task is to assess the number of inhabitants for each neighborhood in Utrecht from the number of inhabitants given per 100×100m square statistical cell[45]. On a conceptual level, the task consists of *summing up amounts of objects within the regions of objects*: counts of objects covering cell regions (**E**) are aggregated into neighborhood counts (**A**,**B**) within neighborhood regions (**C**) obtained from neighborhood objects (**D**).

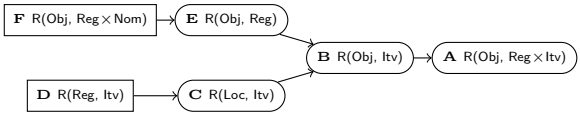


To implement this task, the POPULATION workflow involves a spatial join of cells with neighborhood polygons using a sum operator and using some topological relation.

Scenario 3: Temperature

What is the average temperature for each neighborhood in Utrecht? [38]

The task is to assess an average temperature for each neighborhood in the Netherlands from point measurements[7]. Conceptually, this task corresponds to *averaging a field within the regions of objects*: we first need to interpolate point-wise measurements from weather stations (**D**) into a temperature field (**C**), which is then averaged over neighborhood regions (**E**) obtained from neighborhood data (**F**), resulting in an average temperature for each neighborhood (**B**, **A**).



To solve this task in the TEMPERATURE workflow, point measurements for the entire Netherlands can be interpolated using Inverse Distance Weighting (IDW) to generate a raster, which is then aggregated into administrative regions using zonal statistics.

Scenario 4: Hospitals

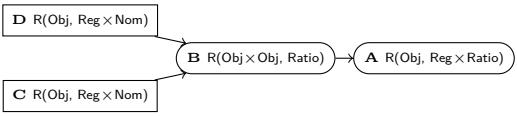
What is the travel distance to the nearest hospital in California? [47]

In this scenario, we need to determine, for a number of accidents, the distance to the closest hospital in California. Conceptually, this corresponds to *minimizing a distance matrix*: We need to generate a distance matrix (**B**)

[7] Temperature time series per station by the Royal Dutch Meteorological Institute (KNMI). [46]

from events (**C**), here interpreted as objects, to objects **D**, and minimize **B** over objects, resulting in minimal distances for each incident (**A**).

Note: In an earlier iteration, the output type of **A** was specified as R(Obj, Ratio). This fails due to incidental bundling multiple attributes together, as described in Sect. 4. It will be addressed in a future version.

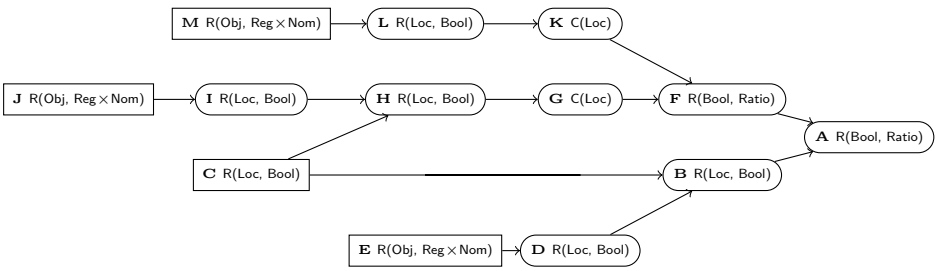


To solve this task in a workflow, we used catchment area (*closest facility*) analysis on a road network (workflow HospitalsNetwork). Alternatively, an equivalent result can be obtained using the *Near* tool based on Euclidean distances (workflow HospitalsNear).

Scenario 5: Deforestation

What is the impact of roads on deforestation in the Amazon rain forest? [48]

We determine the proportion of the deforested area within a buffer of current roads in the Amazon, in order to estimate the size of deforested area near a planned road. Conceptually, the task is to assess the *proportion of area covered by a landuse category within some distance of an object*: Existing road objects (**J**) are buffered, generating a boolean field (**I**) that denotes whether a location is inside or outside the buffer distance. **I** is combined with the deforested area field **C** to derive the intersection **H**, whose spatial coverage **G** is measured as a proportion (**F**) of the coverage (**K**) of the road (**M**) buffers (**L**). This proportion (**F**) is then used to derive the size of the area covered (**A**) by the landuse field (**B**), which is the part of **C** within buffers (**D**) of new roads (**E**).

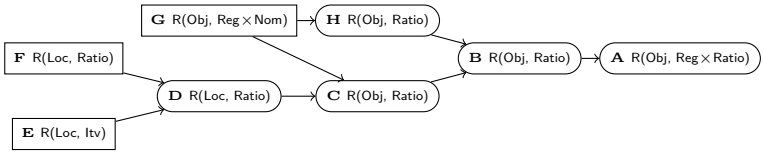


In the workflow, we are given deforested areas as polygons, current and planned roads in the Amazon in terms of line vectors, and we use vector buffer and overlay operations to measure proportions.

Scenario 6: Solar power

What is the potential of solar power for each rooftop in the Glover Park neighborhood in Washington, D.C? [49]

In this scenario, we are estimating the sum of solar energy available on each rooftop in Glover Park. This corresponds to *constraining a field and summing it up over the area covered by objects*: we use the terrain field (**E**) to constrain the solar potential field **F** to **D**, which is aggregated over each region of a building (rooftop) **G** to yield an average potential **C**, which together with the size **H** of rooftops is used to sum an amount of energy per rooftop (**B** and **A**).

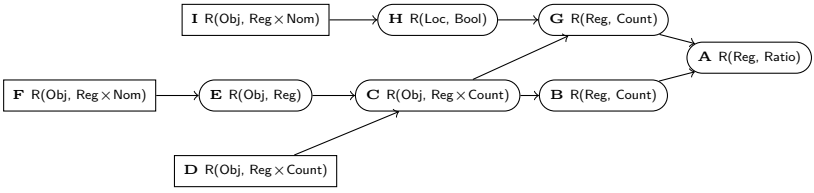


The SolarPower workflow can be implemented using local map algebra on a solar potential raster with both slope and aspect as Boolean constraints using a digital terrain model, which is then averaged over the building polygons using zonal statistics and multiplied by their size.

Scenario 7: Road access

What is the percentage of rural population within 2 km distance to all-season roads in Shikoku, Japan? [50]

This scenario is about estimating the proportion of rural population that have access to roads. Conceptually, this is asking for a *proportion of object count amounts within some distance from objects*: given population counts for each metropolitan region **D**, we select those **C** that are within rural **E** administrative areas **F**. We then build buffers **H** around roads **I** and derive content amounts **G** for those buffer areas by (areal) interpolation from **C**. Finally, we build the proportion **A** of this amount **G** with respect to the total population amount in rural areas **B**.

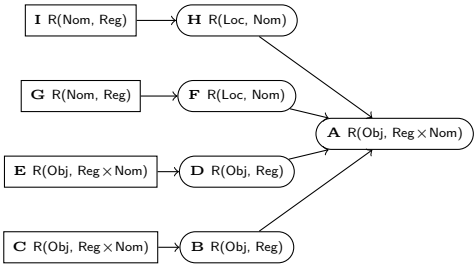


In the RoadAccess workflow, rural population numbers are given for metropolitan polygons, and we use a simple areal interpolation method (with weighted overlay) to estimate the rural population living within road buffers. We then build a ratio of this number with the total population.

Scenario 8: Aquifer

Which urban areas are at risk from water depletion in Ogallala (High Plains) Aquifer, US? [51]

The scenario about water depletion in Nebraska deals with finding out urban areas that are within 150 miles of the Ogallala aquifer, and which have high irrigation needs and low precipitation. This is done by *selecting (urban area) objects overlapping with the coverage of some (low precipitation and high irrigation) fields*: from coverages of low precipitation (**G**) and high irrigation (**I**), we derive corresponding fields (**F**) and (**H**), which are combined to select overlapping urban regions (**D** of objects **E**) that need to be within some distance from the region **B** of the aquifer **C**.

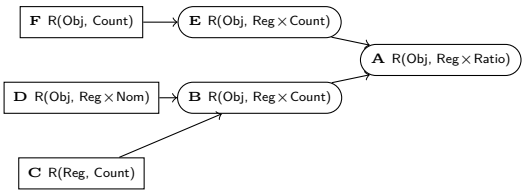


The Aquifer workflow is implemented entirely using vector polygon operations.

Scenario 10: Malaria

What is the malaria incidence rate per 1000 inhabitants in the Democratic Republic of the Congo? [52]

In scenario 10, we form an incidence rate of malaria in proportion to population numbers for each administrative region of the Democratic Republic of Congo. To obtain total population numbers, we need to sum up population amounts given as statistical squares into administrative regions. Conceptually, this corresponds to a *proportion of event and object count amounts within the regions of objects*: we sum up population content amounts for squares **C** within the regions of administrative areas **D** to obtain population counts **B**, which together with malaria incidents **F** on the same administrative areas **E** form proportions **A**.



The Malaria workflow implements this entirely in terms of various table joins of vector polygon data.

Appendix B Type inference algorithm

The type inference algorithm is inspired by [41], who extended the Isabelle theorem prover with automatic insertion of type coercions. Our algorithm is simplified in that the separate steps of subtype constraint generation, simplification, graphing and resolution are condensed into two steps: subtype-unification and subtype resolution. Furthermore, our algorithm is extended with typeclass constraints.

B.1 Notation

We will provide an informal overview of the procedure. To do so, we will first establish notation and terminology.

An expression of a transformation algebra is a repeated application of operators from a set of typed operators $\Sigma_{\mathsf{op}} = \{f_1 : \phi_1 \to \psi_2,\ \ldots,\ f_n : \phi_n \to \psi_n\}$ and a set of typed data sources $\Sigma_{\mathsf{data}} = \{x_1 : \tau_1,\ \ldots,\ x_n : \tau_n\}$.

A *schematic type* is an type containing *schematic type variables*, written α, β, etcetera. Such a type stands for all the *type instances* that can be created following the schema.

A type instance τ may be a *concrete type variable*, denoted a, b, etcetera, or a *type operation*, denoted $C\ \tau_1 \cdots \tau_n$, $D\ \tau_1 \cdots \tau_n$, etcetera. Type operations have an arity n; they are called *base* when $n = 0$ and *compound* otherwise. The portion of a type operation preceding its parameters, if any, is called the *operator*.

Any base type C may have no more than one *supertype* D such that $C \subseteq D$.

Each parameter of a compound type $C\ \tau_1 \cdots \tau_n$ is associated with a *variance*, denoted $\nu(C) \in \{\oplus, \ominus\}^n$, expressing how the subtype of the compound type relates to each of its constituent parameters. The i'th parameter of C is called *covariant* if $\nu(C)_i = \oplus$ and *contravariant* if $\nu(C)_i = \ominus$. For example, for $C\ \tau_1 \cdots \tau_n \subseteq D\ \tau_1{}' \cdots \tau_n{}'$ to hold, its operators must be equal ($C = D$), and for every i'th parameter, $\nu(C)_i = \oplus$ implies $\tau_i \subseteq \tau_i{}'$ whereas $\nu(C)_i = \ominus$ implies $\tau_i{}' \subseteq \tau_i$.

Note that the function operator $\to$ is merely a special compound operator that is contravariant in its input parameter and covariant in its output parameter, e.g., $\nu(\to) = \langle \ominus, \oplus \rangle$. This reflects that that a function should also work on any value of a more conservative input type, and that it produces a value that is also a member of a more liberal output type; see also [53].

The $\texttt{skeleton}(\tau)$ of a type τ is a version of that type where all base types have been replaced with fresh type variables.

A *substitution* θ contains mappings $t \mapsto \tau$ that assign a type τ to type variable t. We write $[\theta]\tau$ for a version of τ where all relevant type variables have been substituted with those types.

A *subtype constraint set* θ^{S} contains lower bounds of the form $(t \dot{\supseteq} C)$ and upper bounds of the form $(t \dot{\subseteq} C)$. They demand that t should eventually be bound to some base type with subtype resp. supertype C.

A *typeclass constraint set* θ^{C} contains user-supplied constraints of the form $(\sigma \dot{\in} \{\tau_1, \ldots, \tau_n\})$ that indicate that there must be at least one instance $[\theta]\tau_i$ in the constraints such that $[\theta]\sigma \subseteq [\theta]\tau_i$.

B.2 Algorithm

With this notation in hand, we sketch the general procedure. Suppose that a transformation of type $\alpha \to \beta$ is applied to an argument of type τ. We then proceed as follows:

1. We try to find a substitution θ and a set of subtype constraints θ^{S} that would ensure that τ has an appropriate type, that is, $\tau \subseteq \alpha$. To do this, we use a variation of standard unification that takes into account subtypes, as outlined in Algorithm 1. In this algorithm, the state of the substitution θ and the constraints θ^{S} are kept in the global state as we recursively descend through the type structure.
2. Now, we know which type variables should be bound to which types for the argument type τ to match with the input type α. While some of the type variables are not yet bound, we *do* know that they are supposed to be bound to some base type. Although exactly *which* could not be determined yet during the previous step, lower and upper bounds were put in place via θ^{S}. Now, we are ready to resolve these variables to concrete base types. At the top level, every type variable with a lower bound is substituted with that bound. Type variables that occur in contravariant parameters will instead be substituted with the bound of the opposite polarity. Note that it is possible that types will not be fully resolved at the end of this process. Consider, for example, applying a function of type $(x \to y) \to x$ to one of type $\mathsf{Nom} \to \mathsf{Obj}$: while we know that $x \subseteq \mathsf{Nom}$, Nom is not necessarily the most specific bound on x. In this case, subtype constraints will carry over to subsequent unifications.
3. Recall that the output type of our transformation $\alpha \to \beta$ was β. We apply the substitution θ that was built during the previous steps to find the final output type $[\theta]\beta$.
4. We check that the typeclass constraints θ^{C} have not been violated.

The above description constitutes a generic type inference system with subsumption. For our particular transformation algebra CCT, base types are those drawn from Δ_{V}, and compound types are those that can be constructed with the type operator R.

Algorithm 1 Subtype-unification.

Data: Unification candidates ϕ and ψ, substitution θ, constraints θ^{S}.
Result: Final substitution θ and subtype constraints θ^{S}.

Function `UnifySubtype`(ϕ, ψ)**:**
  set $\phi := [\theta]\phi$ and $\psi := [\theta]\psi$ **if** $\phi = C\ \tau_1 \cdots \tau_n$ and $\psi = D\ \sigma_1 \cdots \sigma_n$ **then**
    if ($n = 0$ and $C \not\sqsubseteq D$) or ($n \geq 1$ and $C \neq D$) **then**
      unification fails
    for $i = 1$ **to** n **do**
      if $\nu(C)_i = \oplus$ **then**
        `UnifySubtype`(τ_i, σ_i)
      else if $\nu(C)_i = \ominus$ **then**
        `UnifySubtype`(σ_i, τ_i)
    end for
  else if $\phi = C\ \tau_1 \cdots \tau_n$ and $\psi = t$ **then**
    if $n = 0$ **then**
      `AddLowerBound`($t \mathrel{\dot{\supseteq}} C$)
    else
      add $t \mapsto$ `skeleton`(ϕ) to θ `UnifySubtype`(ϕ, ψ) once more
  else if $\phi = t$ and $\psi = C\ \tau_1 \cdots \tau_n$ **then**
    if $n = 0$ **then**
      `AddUpperBound`($t \mathrel{\dot{\subseteq}} C$)
    else
      add $t \mapsto$ `skeleton`(ψ) to θ `UnifySubtype`(ϕ, ψ) once more
  else if $\phi = t$ and $\psi = s$ **then**
    add the substitution $t \mapsto s$ to θ **foreach** $t \mathrel{\dot{\supseteq}} C \in \theta^{\mathsf{S}}$ **do** `AddLowerBound`($s \mathrel{\dot{\supseteq}} C$)
    foreach $t \mathrel{\dot{\subseteq}} C \in \theta^{\mathsf{S}}$ **do** `AddUpperBound`($s \mathrel{\dot{\subseteq}} C$)
Function `AddUpperBound`($t \mathrel{\dot{\subseteq}} C$)**:**
  the current bounds on t, if any, are $(t \mathrel{\dot{\supseteq}} L), (t \mathrel{\dot{\subseteq}} U) \in \theta^{\mathsf{S}}$ **if** $C \sqsubset L$, or neither $C \sqsubseteq U$ nor $C \sqsupseteq U$ **then**
    unification fails
  else
    insert $t \mathrel{\dot{\subseteq}} C$ into θ^{S}, remove $t \mathrel{\dot{\subseteq}} U$ if needed
Function `AddLowerBound`($t \mathrel{\dot{\supseteq}} C$)**:**
  the current bounds on t, if any, are $(t \mathrel{\dot{\supseteq}} L), (t \mathrel{\dot{\subseteq}} U) \in \theta^{\mathsf{S}}$ **if** $C \sqsupset U$, or neither $C \sqsubseteq L$ nor $C \sqsupseteq L$ **then**
    unification fails
  else
    insert $t \mathrel{\dot{\supseteq}} C$ into θ^{S}, remove $t \mathrel{\dot{\supseteq}} L$ if needed

Data availability

The datasets generated for the results (cf. Tables 1 and 2) are available in the figshare repository at [54]. Alternatively, they can be reproduced with the software and instructions available at https://github.com/quangis/cct/tree/article1.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Van Dillen, S. M., de Vries, S., Groenewegen, P. P. & Spreeuwenberg, P. Greenspace in urban neighbourhoods and residents' health: adding quality to quantity. *J Epidemiol Community Health* **66** (6), e8–e8 (2012).

[2] Geluidskaart 2018. https://maps.amsterdam.nl/geluid/. Accessed: 2022-05-01.

[3] De Smith, M. J., Goodchild, M. F. & Longley, P. *Geospatial analysis: a comprehensive guide to principles, techniques and software tools* (Troubador publishing ltd, 2007).

[4] Hofer, B., Mäs, S., Brauner, J. & Bernard, L. Towards a knowledge base to support geoprocessing workflow development. *International Journal of Geographical Information Science* **31** (4), 694–716 (2017).

[5] Gil, Y. *et al.* Examining the challenges of scientific workflows. *Computer* **40** (12), 24–32 (2007).

[6] Paolucci, M., Kawamura, T., Payne, T. R. & Sycara, K. *Semantic matching of web services capabilities*, 333–347 (Springer, 2002).

[7] Fitzner, D., Hoffmann, J. & Klien, E. Functional description of geoprocessing services as conjunctive datalog queries. *Geoinformatica* **15** (1), 191–221 (2011).

[8] Lutz, M. Ontology-based descriptions for semantic discovery and composition of geoprocessing services. *Geoinformatica* **11** (1), 1–36 (2007).

[9] Lemmens, R. *et al.* Integrating semantic and syntactic descriptions to chain geographic services. *IEEE Internet Computing* **10** (5), 42–52 (2006).

[10] Gahegan, M. Specifying the transformations within and between geographic data models. *Transactions in GIS* **1** (2), 137–152 (1996).

[11] Albrecht, J. Universal analytical gis operations: A task-oriented systematization of data structure-independent gis functionality. *Geographic information research: Transatlantic perspectives* 577–591 (1998).

[12] Albrecht, J. *Semantic net of universal elementary gis functions* (Citeseer, 1995).

[13] Scheider, S., Ballatore, A. & Lemmens, R. Finding and sharing gis methods based on the questions they answer. *International journal of digital earth* **12** (5), 594–613 (2019).

[14] Kuhn, W. & Ballatore, A. in *Designing a language for spatial computing* 309–326 (Springer, 2015).

[15] Scheider, S., Meerlo, R., Kasalica, V. & Lamprecht, A.-L. Ontology of core concept data types for answering geo-analytical questions. *Journal of Spatial Information Science* **2020** (20), 167–201 (2020).

[16] Scheider, S. & Ballatore, A. Semantic typing of linked geoprocessing workflows. *International Journal of Digital Earth* **11** (1), 113–138 (2018).

[17] Kuhn, W. Core concepts of spatial information for transdisciplinary research. *International Journal of Geographical Information Science* **26** (12), 2267–2276 (2012).

[18] Kruiger, J. F. *et al.* Loose programming of gis workflows with geo-analytical concepts. *Transactions in GIS* **25** (1), 424–449 (2021).

[19] Brauner, J. Formalizations for geooperators-geoprocessing in spatial data infrastructures (2015).

[20] Tomlin, C. D. *Geographic information systems and cartographic modelling* 910.011 T659g (New Jersey, US: Prentice-Hall, 1990).

[21] Mennis, J., Viger, R. & Tomlin, C. D. Cubic map algebra functions for spatio-temporal analysis. *Cartography and Geographic Information Science* **32** (1), 17–32 (2005).

[22] Frank, A. U. & Kuhn, W. *Specifying open gis with functional languages*, 184–195 (Springer, 1995).

[23] Frank, A. U. *One step up the abstraction ladder: Combining algebras-from functional pieces to a whole*, 95–107 (Springer, 1999).

[24] Ferreira, K. R., Camara, G. & Monteiro, A. M. V. An algebra for spatiotemporal data: From observations to events. *Transactions in GIS* **18** (2), 253–269 (2014).

[25] Camara, G. *et al.* *Fields as a generic data type for big spatial data*, 159–172 (Springer, 2014).

[26] Güting, R. H. *Geo-relational algebra: A model and query language for geometric database systems*, 506–527 (Springer, 1988).

[27] Scheider, S., Gräler, B., Pebesma, E. & Stasch, C. Modeling spatiotemporal information generation. *International Journal of Geographical Information Science* **30** (10), 1980–2008 (2016).

[28] Codd, E. F. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems* **4** (4), 38 (1979).

[29] Scheider, S. & de Jong, T. A conceptual model for automating spatial network analysis. *Transactions in GIS* (2021).

[30] Kuhn, W., Hamzei, E., Tomko, M., Winter, S. & Li, H. The semantics of place-related questions. *Journal of Spatial Information Science* (23), 157–168 (2021).

[31] Galton, A. Fields and objects in space, time, and space-time. *Spatial cognition and computation* **4** (1), 39–68 (2004).

[32] Top, E., Scheider, S., Xu, H., Nyamsuren, E. & Steenbergen, N. The semantics of extensive quantities in geographical information (2022). In press.

[33] Chrisman, N. R. *Exploring geographic information systems* (Wiley New York, 2002).

[34] OWL Web Ontology Language. https://www.w3.org/TR/owl2-overview/ (2012). Accessed: 2022-05-01.

[35] Guarino, N., Guizzardi, G. & Mylopoulos, J. On the philosophical foundations of conceptual models. *Information Modelling and Knowledge Bases* **31** (321), 1 (2020).

[36] RDF - Semantic Web Standard. https://www.w3.org/RDF/. Accessed: 2022-05-01.

[37] Polygon to raster - conversion. https://pro.arcgis.com/en/pro-app/2.9/tool-reference/conversion/polygon-to-raster.htm. Accessed: 2022-05-01.

[38] GI Minor - Learn GIS: Geographic Information Systems, Science and Studies. https://nationalegiminor.wordpress.com/. Accessed: 2022-05-01.

[39] Learn ArcGIS. https://learn.arcgis.com/. Accessed: 2022-05-01.

[40] ESRI. Predict floods with unit hydrographs. https://learn.arcgis.com/en/projects/predict-floods-with-unit-hydrographs/. Accessed: 2022-05-01.

[41] Traytel, D., Berghofer, S. & Nipkow, T. *Extending Hindley-Milner type inference with coercive structural subtyping*, 89–104 (Springer, 2011). URL https://www21.in.tum.de/~nipkow/pubs/aplas11.pdf.

[42] DE-9IM. https://en.wikipedia.org/wiki/DE-9IM. Accessed: 2022-05-01.

[43] Couclelis, H. *The abduction of geographic information science: Transporting spatial reasoning to the realm of purpose and design*, 342–356 (Springer, 2009).

[44] Xu, H., Nyamsuren, E., Scheider, S. & Top, E. A grammar for interpreting geo-analytical questions as concept transformations. *International Journal of Geographical Information Science* (2022). In press.

[45] Aantal inwoners - 500 meter vierkant (2018). https://cbsinuwbuurt.nl/#sub-vierkant500m2018_aantal_inwoners. Accessed: 2022-05-01.

[46] KNMI Dataplatform. https://dataplatform.knmi.nl/. Accessed: 2022-05-01.

[47] ESRI. Identify the closest facility. https://pro.arcgis.com/en/pro-app/2.9/help/analysis/networks/closest-facility-tutorial.htm. Accessed: 2022-05-01.

[48] ESRI. Predict deforestation in the Amazon rain forest. https://learn.arcgis.com/en/projects/predict-deforestation-in-the-amazon-rain-forest/. Accessed: 2022-05-01.

[49] ESRI. Estimate solar power potential. https://learn.arcgis.com/en/projects/estimate-solar-power-potential/. Accessed: 2022-05-01.

[50] ESRI. Estimate access to infrastructure. https://learn.arcgis.com/en/projects/estimate-access-to-infrastructure/. Accessed: 2022-05-01.

[51] ESRI. Find areas at risk from aquifer depletion. https://learn.arcgis.com/en/projects/find-areas-at-risk-from-aquifer-depletion/. Accessed: 2022-05-01.

[52] ESRI. Monitor malaria epidemics. https://learn.arcgis.com/en/projects/monitor-malaria-epidemics/. Accessed: 2022-05-01.

[53] Cardelli, L. Kahn, G., MacQueen, D. B. & Plotkin, G. (eds) *A semantics of multiple inheritance*. (eds Kahn, G., MacQueen, D. B. & Plotkin, G.) *Semantics of Data Types*, 51–67 (Springer Berlin Heidelberg, Berlin,

Heidelberg, 1984).

[54] Steenbergen, N., Top, E., Nyamsuren, E. & Scheider, S. S. Core concept transformation algebra: early evaluations (2022). URL https://figshare.com/articles/dataset/Core_concept_transformation_algebra_early_evaluations/19727233. https://doi.org/10.6084/m9.figshare.19727233.v2.