JOURNAL OF SPATIAL INFORMATION SCIENCE
Number 27 (2023), pp. 51–92
doi:10.5311/JOSIS.2023.27.268

RESEARCH ARTICLE

Procedural metadata for geographic information using an algebra of core concept transformations

Niels Steenbergen, Eric Top, Enkhbold Nyamsuren, and Simon Scheider

Department of Human Geography and Spatial Planning, Utrecht University, The Netherlands

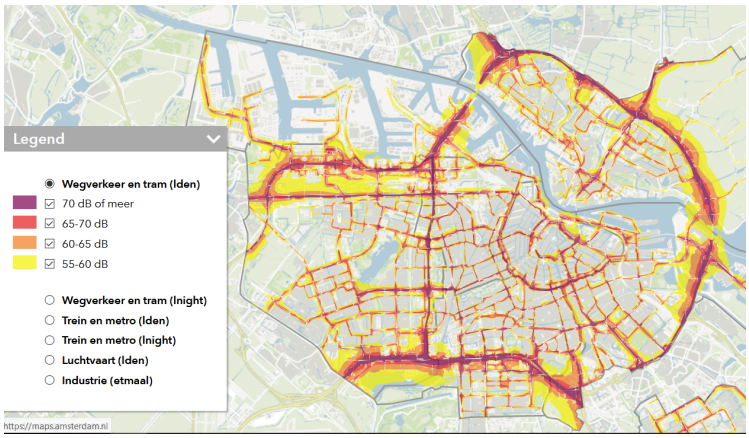
Received: December 12, 2022; returned: March 25, 2023; revised: June 10, 2023; accepted: October 5, 2023.

Abstract: Transformations are essential for dealing with geographic information. They are involved not only in the conversion 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 relation types, and a set of basic transformations described in terms of function signatures that use such types. We enrich GIS workflows with abstract machine-readable metadata, by compiling algebraic tool descriptions and inferring goal concepts across a workflow. In this article, we show how such procedural metadata can be used to retrieve workflows based on task descriptions derived from geo-analytical questions. Transformations can be queried independently from their implementations or data formats.

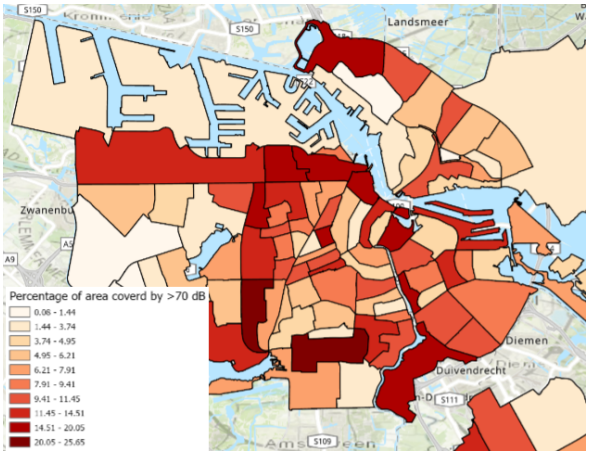
Keywords: core concepts, geographic information, workflows, transformations, type inference, semantic web

1 Introduction

Manipulating and processing geographic information is a core competence relevant for data analysts. It is not only needed for transforming maps across different coordinate reference systems, it is also an essential means for transforming geodata in ways that allow performing particular analytical tasks to answer questions [58]. For example, to learn about the effect of noise on the health of citizens [65], 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.



(a) Map of noise contours in Amsterdam. Source: [3]



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

Figure 1: Transformation of noise contour map into a choropleth 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., having different underlying conceptualizations. We therefore call such manipulations *conceptual transformations*. In this article, we are interested in understanding how such transformations can be chosen according to the purpose given by some geo-analytical task. 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 geo-analytical methods in detail [19] and workflow management has become standard practice in Geographic Information Systems (GIS), we cannot claim to understand this workflow design practice very well [40]—at least not well enough for automating it [35]. 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 [30, 46, 48, 51]. Also, understanding GIS in terms of transformations has been suggested earlier [33]. However, while corresponding technologies are important enablers for (partial) automation, describing and systematizing GIS functionality needed in these services turned out to be hard [10, 11, 40]. A useful theory that links GIS tools with tasks seems still out of sight [54]. We believe this is at least partially due to a lack of understanding of the *know-how* required for handling 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 [44, 57]. Although geocomputational workflows are necessarily implemented in some geometric format, their design and selection requires reasoning beyond geodata models and geometries.

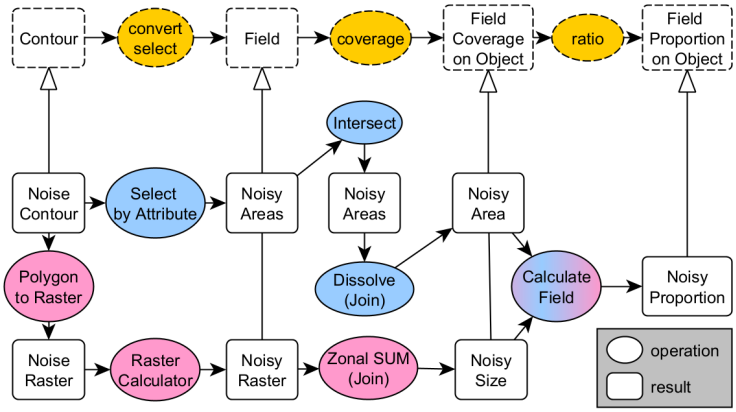


Figure 2: How to transform a noise contour map into a measure of the proportion of noisy area? The dotted transformations are on a *conceptual level*, in parallel to computational procedures implemented with different data models (blue: vector version, red: raster version). Procedures are schematized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

To highlight this point, consider again our example from above. 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 proportions*. For the latter, 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 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 [53].

A spatial field is an example of a *core concept of spatial information* [43]. Core concepts capture the various ways in which the geographic environment can be conceptualized. Together with concepts of measurement, they have recently been successfully used as a basis not only to deconstruct geographic questions [67], but also as semantic data types for GIS functions [57] and for GIS workflow synthesis [42]. Since the well known geodata types do not capture such concepts [50], we need a higher layer of abstraction to describe and retrieve geo-analytical workflows. In particular, we lack models that capture 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. This enables linking automated GIS workflow synthesis on the one hand [42] to descriptions of geo-analytical tasks derived from questions [67] on the other hand, which we both addressed in the cited previous work (see Fig. 3). Eventually, this allows us to automate GIS tasks for the purpose of *geo-analytical question-answering*[1].

In this article, we introduce the *algebra of core concept transformations (CCT)*, a conceptual interface linking geo-analytical tasks to workflow implementations (Fig. 3). It can be used to automatically infer conceptual descriptions of entire workflows from tool descriptions, in order to retrieve workflows based on how they transform concepts into one another. This addresses part of the challenge of *geo-analytical question-answering* [53, 58], namely the subproblem of matching tasks to workflows. That is, we assume that, through external processes, we already have potential workflows to match, and can derive task descriptions from geo-analytical questions. We have published on these problems independently, in [42,55] and [67] respectively. Furthermore, we leave the execution of the workflows wholly out of scope.

Our transformation algebra is based on *relational types*, which present a novel way of modeling geographic information concepts. Concepts are represented as parameterized types that can be *transformed* into each other, which is an innovation over previous ontologies of core concepts [44, 56, 57]. Relation types can represent measurements with one control (e.g. object height) or multiple (e.g. network distance). Using type inference, we propagate type information through a given workflow. Furthermore, we generalize over data formats and implementations. For example, a spatial field may be implemented in raster as well as in vector format. In the algebra, a field occurs simply as a measurement at arbitrary locations.

In what follows, we first discuss previous work on conceptual and algebraic models of GIS (Sect. 2), before outlining our methodology for designing and testing the algebra (Sect. 3). We then explain how core concepts can be modeled as types of relations (Sect. 4). We then introduce 11 empirical workflow scenarios for evaluating our model in Sect. 5 and

[1] Geo-analytical QA is defined as the problem of matching questions with GIS workflows that generate answer maps [58].

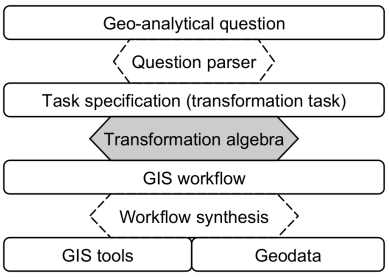


Figure 3: Role of the transformation algebra as an intermediate layer for specifying geo-analytical tasks and for retrieving workflows in geo-analytical question-answering.

specify underlying tasks using conceptual types. In Sect. 6, we explain how such types can be used to describe the functionality of tools and workflows. For this purpose, we specify geo-analytical transformations as function signatures in Sect. 7. Finally, we test our model using a retrieval study over expert workflows taken from the scenarios (Sect. 8).

2 Related work

We review related work on algebraic and conceptual models of GIS tasks and workflows.

2.1 Modelling GIS services

Our work is situated in the context of describing geo-operators [12] for automating the composition of geoprocessing services [47]. For example, [46] suggested integrating semantic and syntactic signatures to describe geo-operations on the Web using Semantic Web service principles [28]. [30] used Datalog rules to specify the functionality by input and output types for geoprocessing services. Such service descriptions may then be combined to a geoprocessing web [69] or for workflow development [40]. However, service descriptions usually focus on syntactic or semantic labels of inputs or outputs of tools [12]. This is required for service composition, yet it is insufficient for interoperability and for retrieval of services [54], since purposes, functionality and applicability constraints can be represented only to a limited extent [56]. Part of the reason is that describing GIS functionality on this level remains hard [11], since it is not independent from software implementations.

2.2 GIS and algebra

The idea of using *algebras* to describe GIS functionality on a conceptual level goes back to Dana Tomlin's *map algebra* [62]. Since the latter captures only a small part of GIS functionality closely related to the raster data model, there have been various attempts at extending it (cf. [49]). Other researchers have focused on functional algebras as more general models to design or implement GIS [31,32]. Algebras were developed specifically for modeling fields [13,29] and for GIS operations in relational databases [37]. However, such approaches focused on generalized *data* models for *implementing* software, not on *conceptual* models for

generalizing over functionality. More recent approaches therefore used functional languages centered around core concepts [44], see below. Our algebra builds on the preliminary ideas in [53, 56, 57], but differs in the following respects:

First, unlike other approaches, we are not targeting implementations, since core concepts are not data types but conceptual models [43]. Focus on implementation details may have prevented success of earlier attempts at systematizing GIS operations precisely when overlooking these concepts [10]. Conceptual transformations, in contrast, remain relatively stable across different implementations. Fields, e.g., should not be treated as a data type [13], but as a concept that can have raster and vector implementations, as suggested in [57].

Second, we use our algebra as an annotation and reasoning language for tools and workflows, primarily for the purpose of *retrieval* and *question-answering*, in addition to synthesis (cf. [42]). To the best of our knowledge, this has not been done previously. Going beyond [56], our type system accommodates a form of subsumption reasoning with parameterized types which allows succinct, abstract descriptions of atomic transformations. So far, such type systems have been mainly used for type checking programming languages, not for semantic annotations for retrieval or question-answering purposes.

Third, in contrast to [56], concepts are not modeled as functions, but rather as *relations* in the sense of relational algebra [17], while functions stand for concept transformations. This makes it easy to distinguish transformations from concepts. Furthermore, we can handle partial and inverted fields as well as spatial networks all in terms of relational concepts [55], as explained below. All this allows us to effectively propagate conceptual types through workflows for the purpose of retrieval, as envisioned in [53].

2.3 Concepts for interpreting spatial information

Which kinds of concepts are required for modeling transformations of spatial information? Here, we give a short review of and justification for such concepts.

2.3.1 Core concepts

Core concepts of spatial information were proposed by [43–45] 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*: Fields capture qualities [45] 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 [56].
- *Objects*: Spatial objects are 'endurants' [34], 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 some relationship between objects [45, 55], e.g., commuter flows between cities or traffic links between intersections in a road network.

The measured quality can thereby change in time. Similar to objects, we also consider locations as keys of a network relation (cf. *relational fields*) [55]. Examples are Euclidean distance (a ratio scaled relation between locations) or the visibility relation in a digital terrain model (a Boolean relation between locations).

For the current version of the algebra, we excluded the *event* concept, because we consider only static scenarios, for which events can be reinterpreted as objects. In addition, note that all concepts discussed above have an *intrinsic temporal dimension* (objects move, fields and networks change in time, and events have a beginning and an end), which is out of scope in the current version of our algebra. It can be added in the future simply by adding another value type for temporal domains of measurement, which can then be used in corresponding relations, cf. Sect. 4.

2.3.2 Amounts and measurement levels

Since our algebra focuses on transformations, we furthermore need to consider how core concepts can be quantified. For this purpose, we use the concepts of *amounts* and *measurement levels* in addition to core concepts.

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 [63]. They correspond to well known geographic operations (cf. [15]). 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 amount measurement: *Content amounts* involve measurements of 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)[2]. *Coverage amounts* are formed by controlling content (e.g. collections of objects or field values) and measuring (the size of) the spatial regions covered[3] (cf. [15, 63]). Examples would be the area covered by roads in a city, or the area covered by a certain type of vegetation.

Finally, such measurements of core concepts and of geographic amounts can be on different *levels of measurement* [15]; cf. [61]. These comprise classes of measurement scales that preserve certain operational characteristics, such as (in ascending order of complexity): equality (nominal level), ordering (ordinal level), differences (interval level), ratios (ratio level), as well as counts (results of countings, where values are integers) [15]. Since the operations of less complex classes of scales apply also to more complex ones (e.g., equality can be distinguished also among ordinal levels), the latter are treated as subclasses of the former.

2.3.3 Some notes on the relevance of these concepts

One may ask why the level of *core concepts*, *measurement levels* and *amounts* should be considered adequate for modeling spatial information, and not more specific domain concepts,

[2] In [63], this corresponds to magnitudes of amounts formed by *capacity measurements*, i.e., measurements that control for space and measure content.

[3] In [63], these are magnitudes of amounts formed by *occupancy measurements*, i.e., measurements that control for content and measure its spatial extent.

such as noise or humidity, or well established data types such as raster or vector [66]. The reason is that they all correspond to different levels of knowledge, with different purposes. While knowledge of meteorology is required for answering questions about weather and climate, it actually plays a limited role in handling cartographic principles. In contrast, we know that the knowledge captured by core concepts (cf. [43]), amounts [63], and measurement levels [15] is necessary for interpreting maps and for solving geo-computational tasks, regardless of the represented domain [44, 54]. To understand the role of these concepts, it should be noted that they are not covered by current geodata types, and that they are part of recent theories which are still under development [44, 57, 63]). The algebraic model proposed in this article is beneficial primarily because it demonstrates how these theoretical concepts can be used to model workflow transformations. Note, however, that spatial information cannot be fully understood without additional concepts like spatial resolution or accuracy, which capture aspects of *data quality* and *uncertainty* [43], or without *domain specific categories*, such as noise or humidity. Although the former are required to implement a workflow, and the latter to select data for it, such details are rarely relevant when deciding whether a workflow is in principle relevant for a purpose, or whether it can answer a given question. Such concepts are therefore out of scope of the current article. Furthermore, executing a workflow usually requires computational parameters, but we leave them out of consideration because we focus on modeling the intended functionality (cf. [57]).

2.4 Type inference

The usefulness of types in programming lies in information hiding, a way of assuring compatibility while abstracting from implementation details [38]. For example, type constraints allow us to check that a function can be applied without knowing all the details of the input. Furthermore, the ability to combine and to infer types in an automated fashion (via subtyping or type parameters) is useful to handle the functional variability of GIS operations [15] and to account for information provenance. For instance, we do not have to determine output types a priori, but we can infer such types based on input types using some algebraic expression that describes a workflow's functionality.

Type inference is a common feature of programming languages to perform static checks of *implementations* [41]. 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. [9] However, we use an independent type inference module to access features like type variables. To our knowledge, this idea has not been applied to produce workflow metadata before.

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 [36].

- *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* so as to express operations in terms of other operations (cf. [56]). Since the algebra does not concern itself with implementation, powerful type inference features, including subtype- and 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* [17], which provides a general model for interpreting core concepts in terms of types of relations. We will justify each of these ideas below.

3.2 Overview

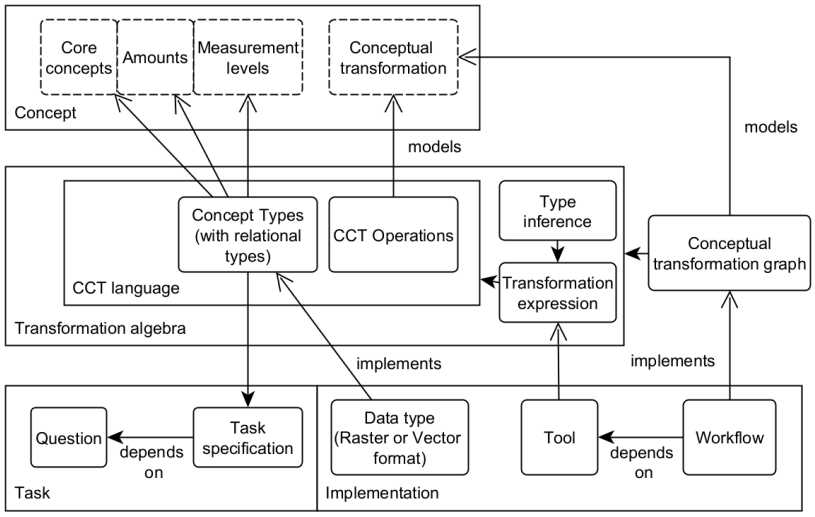


Figure 4: Terminology used in the paper. The transformation algebra models concepts and their transformations (dotted boxes), and is used to describe tasks and implementations. CCT is a particular vocabulary using the algebra syntax. Thick arrows denote dependencies, thin arrows denote implementation or modeling relations.

Fig. 4 gives an overview of the terminology used in the following. The methodological approach is shown in Fig. 5. The algebra was developed initially based on 12 GIS tasks selected from GIS student exercises over several development cycles. By 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?'.

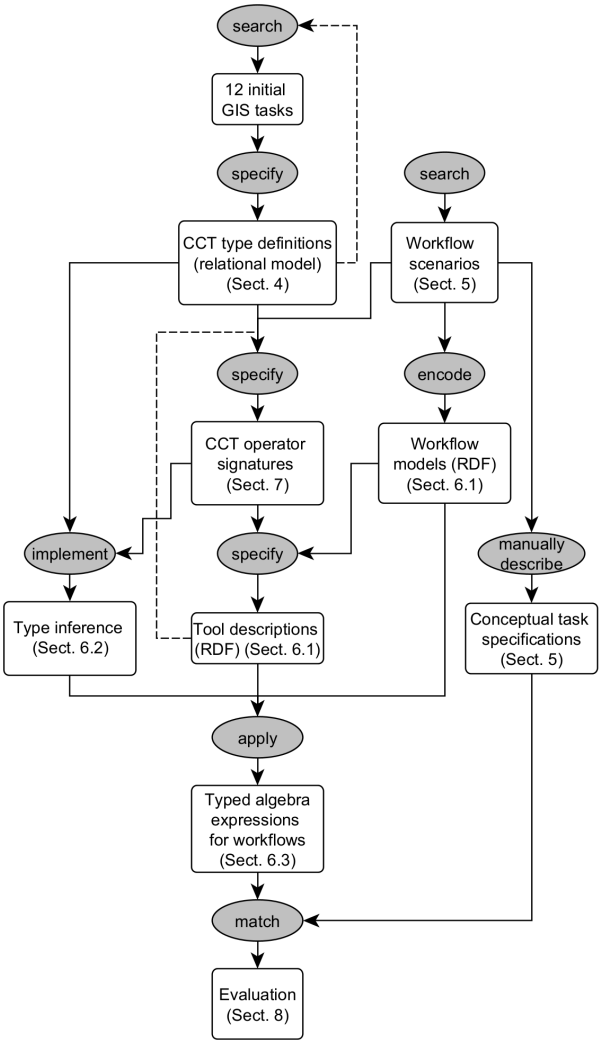


Figure 5: Flowchart of methods used in this article. The dotted arrows indicate feedback in the development cycle.

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.

To serve as empirical basis for evaluation of the algebra, we collected *11 analytical scenarios* from online tutorials and GIS courses. Since the knowledge required to solve geo-analytical problems is not fully specified in a task, it requires expert knowledge for interpreting it in terms of transformations. Since this knowledge is tacit, it is not readily

available as data and needs to be infused into the learning process [58, 59]. Online GIS tutorials are a useful source for acquiring such knowledge, since they were created as a learning source based on procedural knowledge. For this purpose, we selected scenarios that covered diverse types of GIS analysis tasks from introductory GIS courses and tutorials.

We first manually described the underlying analytic task for each scenario in terms of a directed acyclic graph between concept types, based on the task documentation (Sect. 5). We call this graph a *task transformation graph* and, for brevity, we will sometimes refer to it as simply the *task*.

For each task, we then implemented one or more GIS workflows following the tutorial descriptions. A GIS workflow is a 'blueprint' for performing manipulations on data in a geographic information system. It can be represented as a directed acyclic graph with nodes that represent either data artefacts or applications of GIS tools. We have encoded this in the RDF format [8] (Sect. 6.1). In what follows, these graphs are what we will refer to simply as *workflows*. All these steps were developed in an iterative manner over multiple development cycles.

All GIS tools occurring in the workflows were then described using transformation expressions (Sect. 6.1). We 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 workflow, by propagating type information through each operator.

By composing the transformations of individual tools into full workflows, we show that we obtain type-correct transformation graphs and thus that tool combinations imagined by workflow authors are foreseen in our algebra model. We then evaluate these workflows by performing retrieval tests using task transformation graphs (Sect. 8). The retrieval tests assess to what extent the conceptual transformations underlying geo-analytic tasks match the types in the workflow, and whether they are distinctive enough to pick a valid workflow for a given task. A large scale evaluation on more workflows is still future work.

4 Types

In this section, we show how core concepts of geographic information, amounts and their measurements can be modeled as *types of relations*.

4.1 Why relation types?

As explained in Sect. 2, core concepts relate different domains of measurement: fields relate locations to measurements of qualities, objects relate their measured qualities with the spatial regions they occupy, and networks relate pairs of objects (or locations) to measured qualities. Following Sinton [60], in GIS and more generally in measurement, measuring amounts implies that we control for other amounts [63]. For example, amounts of space (regions) control the amounts of population they contain, and a particular amount of noise can be used to control the space occupied by it. Finally, such measurements can be considered on different levels. To express the complexity of controlling and measuring such forms of spatial information, a hierarchy of relation types is thus well suited. Controls then become key attributes in corresponding relations, as we will detail in the following. Although the idea of representing and manipulating information in terms of relations is

borrowed from relational algebra [16,17], 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. In earlier work, we modeled fields as partial functions [56], but in this paper, we make a clear distinction between the conceptualization of static computational results (modeled as relation types) and computational steps (modeled as function types). Furthermore, we dismiss the idea that our types stand for particular data structures, such as a database table. They are rather means of conceptualizing and manipulating geographic information, and thus generalize over implemented data structures.

4.2 Values of measurement

A measurement or observation in GIS yields a *value*. These have a *type*, which is simply the domain of values belonging to that type. Our class of value types, Δ_{Val}, consists of the types of spatial objects Obj, which are represented by object names; locations Loc, represented 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. The type hierarchy implies that every measurement on a ratio level is also on an interval level. For example, measurement of noise in decibels is at the same time also on an ordinal level, because decibel measurements can be ordered. Finally, we call the overarching type of values Val.

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

4.3 A relational model of spatial information

Our notion of a *relation* is similar to an 'abstract' database table: each element assigns a value to each of a set of attributes, as in relational algebra [16]. Despite the name, these relations do not coincide exactly with set-theoretical relations, because some attributes are *key*, in that each combination of values for those attributes must be unique.

While we will not further examine the contents of any specific relation, we do consider *types* of relations. To this end, we introduce the *type operator* R. This operator is parameterized by two types: the first for the key attribute and the second for the dependent attribute. It is defined as the set of those relations in which the values draw from corresponding types. For example, $\mathsf{R}(\mathsf{Loc}, \mathsf{Itv})$ is the set of relations that uniquely associate locations with interval-scaled values. Note that a parameter of R may itself also be a relation type. The class of relation types is Δ_{R}.

We also introduce the product type $\alpha_1 \times \cdots \times \alpha_n$, representing the type of tuples in which the value at index i is drawn from the type α_i. The product type can be used to characterize relations with multiple attributes, as in $\mathsf{R}(\mathsf{Obj} \times \mathsf{Obj}, \mathsf{Itv})$.

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, because Itv $\subseteq$ Qlt, we also have R(Obj $\times$ Obj, Itv) $\subseteq$ R(Obj $\times$ Obj, Qlt).

Finally, we write ϵ for the unit type, which has only a single value, namely, the empty tuple (i.e. $\epsilon = \{\langle\rangle\}$). This type can be used in case we want to speak about relations that have only key attributes: then the independent attributes can use the empty tuple as a dummy value.

Collections The simplest kind of relation is called a *collection*. A collection is a relation with only a single attribute, which is also the key. This can be thought of simply as a subset of values of the type of its key 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).

Simple relations *Simple relations* have a single key attribute which uniquely identifies an entry, and the other attribute is the only independent one. We call these types 'simple' 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. These relations can serve as a model for particular *core concepts of spatial information*:

- *Spatial fields 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 and coverage amounts* R(Qlt, Reg) map quality values into regions that are covered by this value, e.g. landuse coverages or contour regions. 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 and 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 relations 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).

Composite relations Composite relations have *two* keys and a single dependent attribute. This allows us to talk about relationships *between* concepts, and to measure 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 $\times$ 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 $\times$ 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?

Simple relations with 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 $\times$ 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 order of the dependent attributes, or used a tuple of simple relations. 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. [7]

5 Scenarios

Our analytic tasks are selected from GIS expert tutorials that are used to showcase the functionality of GIS software and from GIS introductory courses. Thus, we can assume that the underlying practices and workflows are suitable for constructing a standard benchmark of geo-analytical tasks. We collected 11 documented workflow scenarios as an empirical basis for testing, taken from course material of a GIS minor [4], and from ESRI's ArcGIS online learning hub [6] and similar online tutorials. Scenarios are thematically diverse, complex and specific enough to be of practical relevance for GIS analysts. For all online coursework, corresponding tutorials can be found under the indicated web resources.

5.1 Task transformation graphs

We manually interpreted each scenario in terms of the underlying task, by providing a *task transformation graph*. This is a directed acyclic graph with CCT types as nodes, with

edges between them indicating that one type is (directly or indirectly) derived from another. They capture the high-level concept types and transformations inherent to the task *itself*, disregarding any tool that could be used to fulfill it. As such, they might capture only some important conceptual steps, glossing over steps that would be annotated in the transformation graph for the workflow that implements it. In the remainder, we will refer to task transformation graphs as simply *tasks*.

Sensible task transformation graphs can be automatically derived from a (subset of) natural language questions, but doing so is a subproblem that we addressed in other articles [67]. In this section, instead, we manually construct transformation graphs for each problem and explain why we expect suitable workflows to conform to it.

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 transformation graphs and details of the workflows are available at the CCT repository at https://github.com/quangis/cct/tree/article1, under the `tasks/` and `workflows/` directories, respectively. The repository has additionally been archived at 10.6084/m9.figshare.19688712.

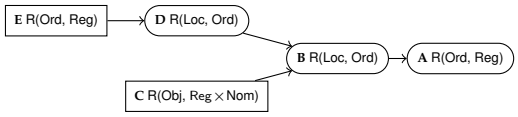
We will illustrate two scenarios, along with their task and associated workflows. Due to space constraints, the explanations of the remaining nine have been moved to Appendix A.

Scenarios 1a and 1b: Noise

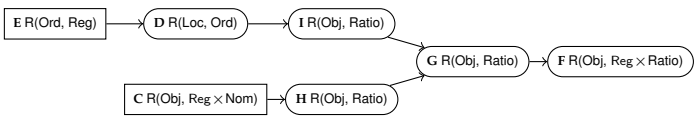
What is the proportion of noise $\geq$ 70dB in Amsterdam? [4]

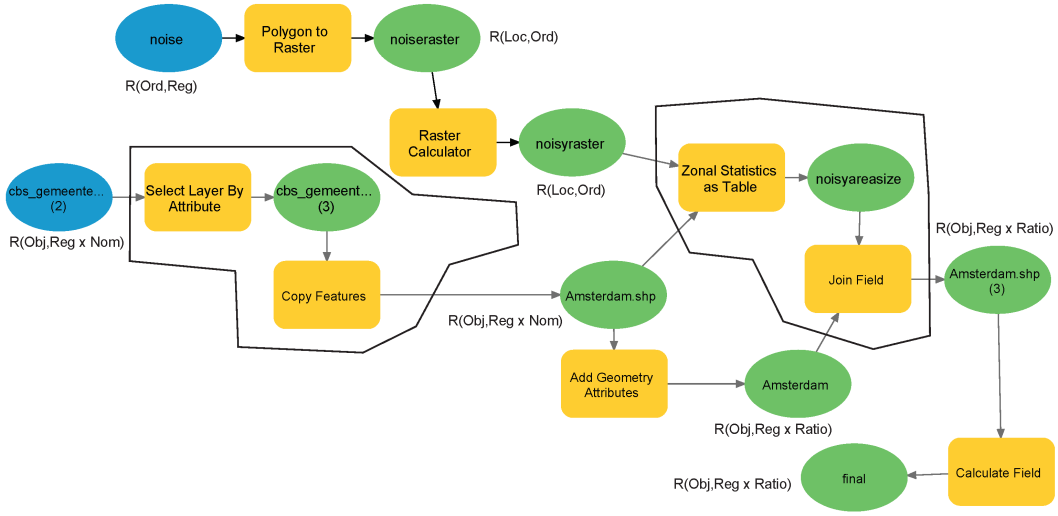
The task is to quantify traffic noise in Amsterdam. This scenario has two subscenarios. In the simplest one, *NoisePortion*, the noise contour map in Fig. 1a needs to be transformed into the area covered by noise $\geq$ 70dB. Alternatively, in the *NoiseProportion* variant, we normalize this area, e.g.,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

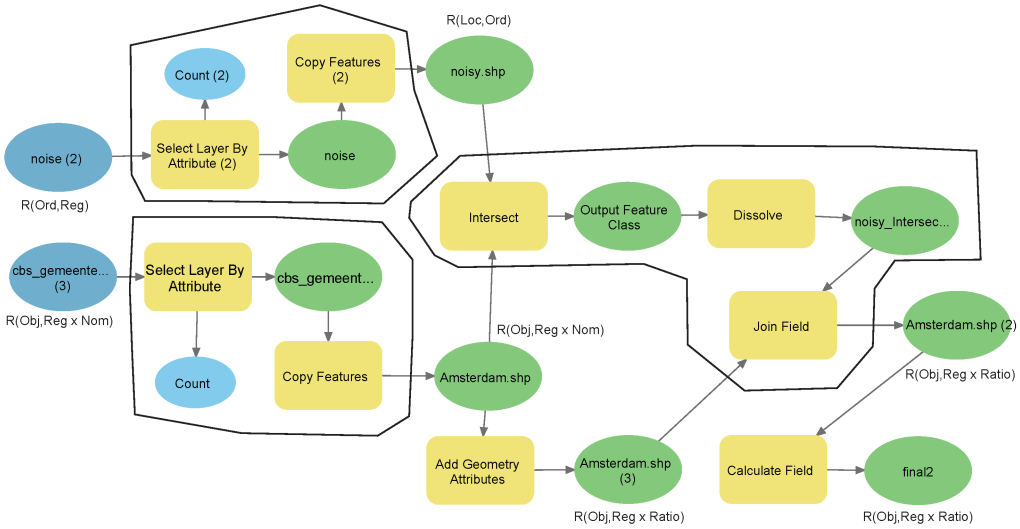


For the *NoiseProportion* case, we generate a ratio scaled quality of neighborhoods (**G** and **F**), a proportion, of the sizes of neighborhood regions (**H**) and the sizes of the ordinal fields (**D**) within these neighbourhoods (**I**), originating from a noise contour map (**E**).





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



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

Figure 6: 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) .

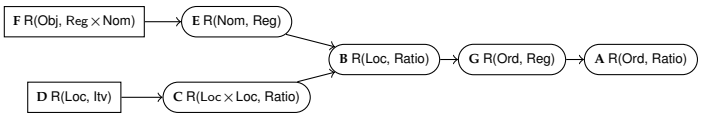
The NOISEPORTION implementation rasterizes the contour map, clips the raster to the spatial extent of the municipal polygon of Amsterdam, constrains the cell values (≥ 70) using local map algebra, and converts the constrained raster layer into a vector polygon.

There are different raster and vector solutions that implement *NoiseProportion*, as depicted in Fig. 6. 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? [27]

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.

6 Conceptual workflow description

There are various levels at which a scientific workflow could be described semantically, such as by their tool names and internal and external data types [35]. In previous work on automated workflow synthesis in GIS, [42] we distinguished tools from their applications in workflow graphs on the one hand, and from their semantic signatures on the other hand. Automatic synthesis is then possible based on different semantic and syntactic input and output types that a tool can have. Workflow composition can be automated for various kinds of GIS software [55].

However, in the current article, we focus instead only on *selecting* such synthesized workflows for a given task. For this purpose, we need to be explicit regarding a workflow's *functionality* to transform concepts, while data types and tool names can be disregarded. Correspondingly, algebraic descriptions are not used for *performing* the underlying task,

but for *describing* it in such a way that the produced metadata can be used for workflow retrieval. In this section, we show how we combine geo-analytical operators (cf. Sect. 7) into *workflow transformation graphs*.

6.1 Workflow transformation graphs

Geo-analytical questions contain conceptual information about what needs to be done; GIS workflows contain implementation details about how to do it. They are speaking about the same problem, but using different vocabularies. Whereas task transformation graphs capture the concepts underlying questions, workflow transformation graphs provide a conceptual look inside a workflow. This exposes a common layer, which makes it possible to match the former against the latter.

The GIS workflows of Sect. 5 are represented as graphs with data artefacts and applications of GIS tools as nodes. These GIS tools refer roughly to ArcGIS Pro or QGIS tools, which have been manually annotated with a *transformation expression*, which describes the underlying conceptual transformation. These expressions are then stitched together into workflow transformation graphs to describe full workflows. The conceptual type at each step in the graph is automatically inferred.

The annotations are available in the `tools/` directory of the CCT repository. Note, however, that actual software tools and the tool nodes we use 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. 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. 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 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* tools. 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.

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

Moreover, the operators may generalize over various concepts. 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 types produced by a particular application are only calculated in the context of the whole transformation graph.

We have implemented the algorithm as a stand-alone Python module. The code is freely available and documented at https://github.com/quangis/transforge. This approach allows for flexibility for our atypical use case: it needs to be integrated with knowledge graphs and embedded in a broader ecosystem for workflow synthesis. The algorithm is inspired by [64] for the implementation of type parameters and subtyping, while type constraints were added ad-hoc to allow for a form of functional dependencies (details in Appendix C).

In this article, we do not study the formal correctness of the algorithm via proofs of termination or soundness and completeness. Such information is of interest to computer scientists and provides valuable guarantees for using the system in the wild, but it is nontrivial and does not affect the experimental results in this paper. After all, our experiments show that the inference process does correctly terminate for our limited collection of expert-annotated tools. Under the constraints of our empirical domain, general considerations become less important (cf. [39]).

Nevertheless, it is worthwhile to point out that there are type systems with subtyping that have been proven terminating and correct, like [52] and the aforementioned [64]. Such proofs also exist (with some caveats) for systems that support functional dependencies [20]. This is far from a guarantee that the simultaneous support for these features also enjoys decidability (or, indeed, that our particular implementation has no bugs). We consider it future work to establish this by formal analysis, or, preferably, fitting our machinery with a well-investigated algorithm that has the appropriate features.

Subtype polymorphism The symbol : means 'of type' in our operator signatures and expressions. Since 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 and functional dependencies However, to define transformations that work on multiple types that are not merely subtypes of each other, we addi-

tionally accommodate *parametric polymorphism*. That is to say: signatures may contain type variables.

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

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.

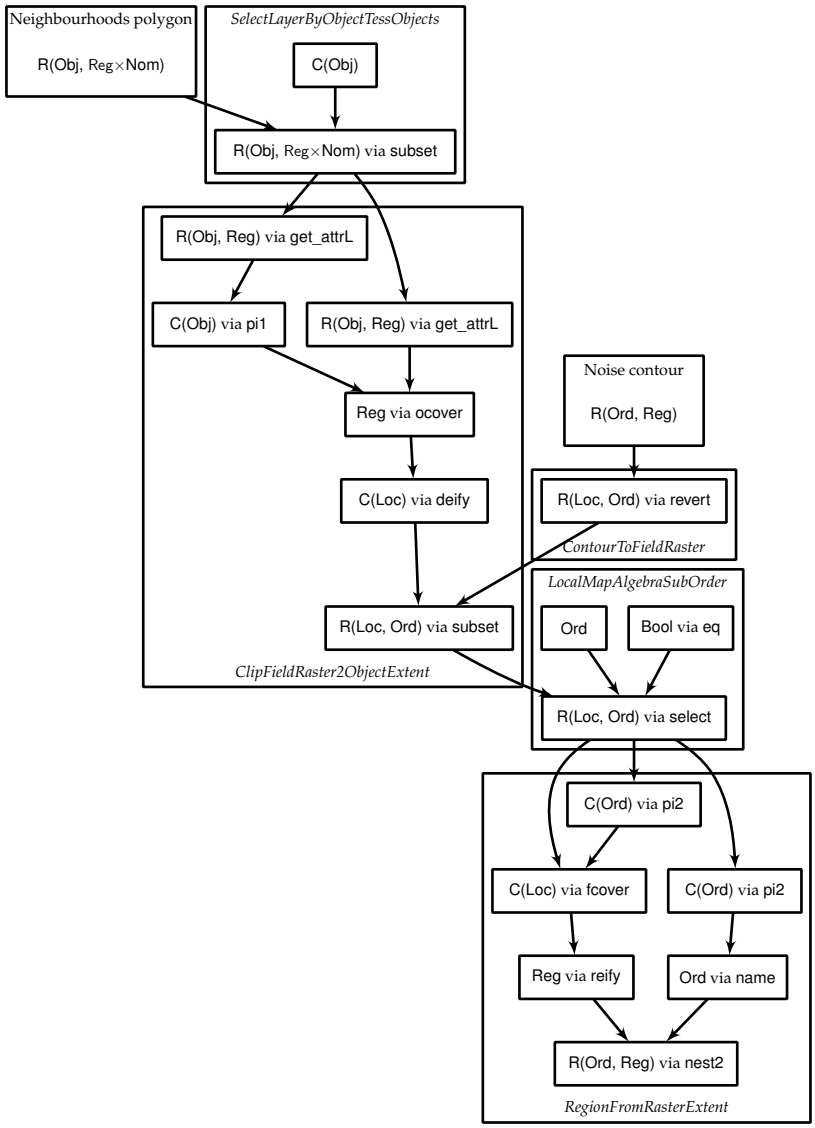
Figure 7 illustrates the resulting transformation graph for the NOISEPORTION workflow. We will describe the meaning of these operators in the next section.

7 Geo-analytical transformations

We specify geo-analytical transformations in terms of the operators introduced in this section. We give an overview of most essential operators in order to illustrate the range of GIS functionality that can be captured with them. Familiar functional and relational operators can be found in Appendix B. However, to understand our general approach, it is not essential to understand the particularities of each. In this article, we used them only to produce types; they have no other content.[4]

[4]The operators themselves could be used as semantic markers in the tasks. It is also possible to give them a 'body', by assigning a lambda term to operators so as to define some operators in terms of others. We chose not to do so in this article. See discussion for more information.

Figure 7: Simplified procedural annotations for the NOISEPORTION workflow.



Moreover, operators are subject to continuous revision. While the operators are sufficient for describing the functionality needed for the workflow scenarios, they are expected to be updated or modified in the future.[5]

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. We nevertheless 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 CCT repository.

7.1 Transformations of geo-analytic values

Value derivations

objectify : Nom → Obj
nominalize : Obj → Nom
leq : Ord → Ord → Bool
eq : Val → Val → Bool
and : Bool → Bool → Bool
not : Bool → Bool
ratio, product : Ratio → Ratio → Ratio
classify : Itv → Ord

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

count : C(Obj) → Count
size : C(Loc) → Ratio
merge : C(Reg) → Reg
name : C(Nom) → Nom
centroid : C(Loc) → Loc

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

avg : R(Val, Itv) → Itv
min : R(Val, Ord) → Ord
sum : R(Val, Ratio) → Ratio
max : R(Val, Ord) → Ord

Note that these operations are on simple relations, 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.

[5] Because atomic operations have no content beyond their name and type signature, there are infinite ways to pick them. An operator do_something with a catch-all type would be 'correct', but certainly not specific enough to allow us to distinguish workflows.

7.2 Geometric transformations

Constructors for composite relations

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

We start with operations to *construct* composite relations. 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. [2]

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.

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

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.

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

lVis measures visibility between locations, given some terrain model as a field. It is the basis of *visibility analysis* in GIS. gridgraph builds a relational field 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

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

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

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

Based on distance relations between locations, we can define a boolean field that indicates whether the distance from any field location to the nearest object is within a certain range. 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 (spatial 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[6], 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 [62] can be introduced in this way, but are not needed for our workflows.

7.3 Amount operations

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

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.

8 Evaluation

To support the claim that our algebra captures properties that are conceptually relevant for transforming spatial information, we test whether (1) the building blocks of our language are *general* enough to allow for the various tool decisions made by workflow authors, yet (2) *specific* enough to allow for distinguishing workflows based on independent descriptions of the underlying tasks.

[6] This function computes the flow direction from a height field. The direction is measured in terms of a location network, i.e., as a location vector in the neighborhood of a location.

8.1 Criteria of evaluation

Type correctness of workflows The type inference algorithm addresses the first concern as a by-product, as it requires that the computed output type of every tool application in a workflow is subsumed by the most general possible input type of its successor. This provides evidence 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 transformation graphs to the types in the workflow transformation graph, as automatically constructed from the workflows and tool descriptions.

A type in the workflow is still 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 an error: 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 tasks match that of the input and output of the workflow as a whole.
2. Additionally, we test whether all types that occur in the tasks occur in the annotations *as inputs or outputs of tools*.
3. Finally, we test whether they occur *anywhere*, even internal to the tool.

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

- We test the effect of matching types in the *order* specified in the flowchart. Remember, here, that a task is almost always a partial specification: when matching it to a workflow, any number of intermediate steps may be skipped. That is, if a workflow flows from A to B to C, it will happily be matched by a task that asks us to go from A to C—but not one that goes from C to A, unless order is disregarded.
- 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 actual input data,[7] which

[7] 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.

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 may be different for each particular tool application. We find the extent to which this *passthrough* affects the result.

The idea here is that we can *contrast* the performance of the model when it has access to all internal information exposed by the CCT expressions, versus a *baseline* situation in which everything but the input and output types of tools are hidden, and in which those types cannot influence each other across tool applications. The latter *simulates* a situation in which we do not use CCT expressions at all—in which tools are annotated with static input and output types. The information available then becomes similar to that in work that annotates tools with input and output types. The exception is that our baseline still uses *conceptual* types rather than *data* types, because geo-analytical questions by nature do not contain information about data formats. Even so, we believe that the performance against this baseline provides valuable information about the contribution of CCT expressions in the presence of conceptual types.

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.

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

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).

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	○	○	○	○	○	○	○	○	○	○	●

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 graph 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. This leaves room for future improvement. Furthermore, while current task graphs contained 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 transformation graphs.

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 [57]). Both can be done in future versions.

Taking into account the ordering of the task transformation graphs 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 descriptions 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 transformation graphs 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 descriptions, we cannot rule out that knowledge of the tools has influenced the description 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.

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.

Finally, we have not provided algorithmic analysis on our type inference module. As stated in Sect. 6.2, this is a pragmatic decision: our specific experimental results are not

affected by the presence of proofs about generalities. Nevertheless, rigorous analysis would provide guarantees about how the system would fare in the wild. This limitation should be addressed in future work, or, if possible, eliminated by fitting an existing algorithm into the system.

9 Conclusion and outlook

We have introduced a relational model of core concepts of geographic information, which can be used to specify tasks and GIS workflows in terms of conceptual transformations. Our algebra provides a vocabulary to specify transformations between these concepts, and makes it possible to automatically add procedural knowledge to geographic workflows using type inference.

Our tests suggest that there is merit to this approach, by showing that algebraic annotations and subsequently inferred types provide important but implicit procedural knowledge associated with analytical GIS tasks. We have implemented a generic toolset to facilitate the production of procedural annotations in GIS as a basis for distinguishing workflows. This modular toolset exists in the established Python and RDF ecosystems and 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.

This makes a new form of automation possible, namely the retrieval of workflows using conceptual task descriptions that are independent of data formats and implementation details. GIS tools can be described by transformation expressions, which can be used to propagate conceptual types through workflows. In this way, natural language questions interpreted as tasks can be matched with workflows that generate answer maps. This is a core challenge for (indirect) geo-analytical question answering [58].

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. It is very probable that our current operator model is not general enough for all GIS tasks. We had to add new operators when considering new workflows. Such additions do not make the general model, type system and retrieval mechanism obsolete, and they became less and less frequent the more workflows were added. The principle of specification can be extended if needed for retrieving GIS workflows for future GIS tasks. Furthermore, query results may be further improved by using both types and function names for retrieval.

There are several possible areas of application of our algebra as a language. First, the algebra serves as a description and retrieval language for workflows that allows querying over different implementations. It could therefore be used in the future to describe and compare software systems measuring their conceptual similarity. 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 [18]. 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 [67]. To this end, the method should be integrated into the full stack of a

geo-analytical QA system [58]. To reach this goal, we need to combine workflow synthesis using data types as in [42] with geo-analytical question parsers [67] using the algebra as an intermediary query layer. Two major open problems concern the handling of data quality, tool parameters and domain concepts in selecting workflows for particular data sources. While the former will require separate conceptual models for quality concepts and parameters, domain concepts might best be handled using NLP methods for keyword embeddings [68], or else by introducing additional types.

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).

Data availability

The datasets generated for the results (cf. Tables 1 and 2) are archived under 10.6084/m9.figshare.19727233.v2. Alternatively, they can be reproduced with the software and instructions available at https://github.com/quangis/cct/tree/article1.

References

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

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

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

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

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

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

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

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

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

[10] ALBRECHT, J. Semantic net of universal elementary GIS functions. In *Proceedings ACSM/ASPRS Annual Convention and Exposition Technical Papers* (1995), Citeseer.

[11] ALBRECHT, J. Universal analytical GIS operations: A task-oriented systematization of data structure-independent GIS functionality. *Geographic Information Research: Transatlantic Perspectives* (1998), 577–591. doi:10.1201/9781482267938-42.

[12] BRAUNER, J. *Formalizations for geooperators: Geoprocessing in spatial data infrastructures*. PhD thesis, Technische Universität Dresden, 2015.

[13] CAMARA, G., EGENHOFER, M. J., FERREIRA, K., ANDRADE, P., QUEIROZ, G., SANCHEZ, A., JONES, J., AND VINHAS, L. Fields as a generic data type for big spatial data. In *International Conference on Geographic Information Science* (2014), Springer, pp. 159–172. doi:10.1007/978-3-319-11593-1_11.

[14] CARDELLI, L. A semantics of multiple inheritance. In *Semantics of Data Types* (Berlin, Heidelberg, 1984), G. Kahn, D. B. MacQueen, and G. Plotkin, Eds., Springer Berlin Heidelberg, pp. 51–67. doi:10.1201/9781482267938-42.

[15] CHRISMAN, N. R. *Exploring geographic information systems*. Wiley New York, 2002.

[16] CODD, E. F. Relational completeness of data base sublanguages. In *Courant Computer Science Symposia No. 6: Data Base Systems*. Prentice- Hall, New York, 1972, pp. 67–101.

[17] CODD, E. F. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems* 4, 4 (Dec. 1979), 38. doi:10.1145/320107.320109.

[18] COUCLELIS, H. The abduction of geographic information science: Transporting spatial reasoning to the realm of purpose and design. In *International Conference on Spatial Information Theory* (2009), Springer, pp. 342–356. doi:10.1007/978-3-642-03832-7_21.

[19] DE SMITH, M. J., GOODCHILD, M. F., AND LONGLEY, P. *Geospatial Analysis: a Comprehensive Guide to Principles, Techniques and Software Tools*. Troubador Publishing Ltd, 2007.

[20] DUCK, G. J., PEYTON-JONES, S., STUCKEY, P. J., AND SULZMANN, M. Sound and decidable type inference for functional dependencies. In *Programming Languages and Systems* (Berlin, Heidelberg, 2004), D. Schmidt, Ed., Springer Berlin Heidelberg, pp. 49–63.

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

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

[23] 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.

[24] 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.

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

[26] 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.

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

[28] FENSEL, D., FACCA, F. M., SIMPERL, E., AND TOMA, I. *Semantic Web Services*, vol. 357. Springer, 2011.

[29] FERREIRA, K. R., CAMARA, G., AND MONTEIRO, A. M. V. An algebra for spatiotemporal data: From observations to events. *Transactions in GIS 18*, 2 (2014), 253–269. doi:10.1111/tgis.12030.

[30] FITZNER, D., HOFFMANN, J., AND KLIEN, E. Functional description of geoprocessing services as conjunctive datalog queries. *Geoinformatica 15*, 1 (2011), 191–221. doi:10.1007/s10707-009-0093-4.

[31] FRANK, A. U. One step up the abstraction ladder: Combining algebras-from functional pieces to a whole. In *International Conference on Spatial Information Theory* (1999), Springer, pp. 95–107. doi:10.1007/3-540-48384-5_7.

[32] FRANK, A. U., AND KUHN, W. Specifying open GIS with functional languages. In *International Symposium on Spatial Databases* (1995), Springer, pp. 184–195. doi:10.1007/3-540-60159-7_12.

[33] GAHEGAN, M. Specifying the transformations within and between geographic data models. *Transactions in GIS 1*, 2 (1996), 137–152. doi:10.1111/j.1467-9671.1996.tb00040.x.

[34] GALTON, A. Fields and objects in space, time, and space-time. *Spatial cognition and computation 4*, 1 (2004), 39–68. doi:10.1207/s15427633scc0401_4.

[35] GIL, Y., DEELMAN, E., ELLISMAN, M., FAHRINGER, T., FOX, G., GANNON, D., GOBLE, C., LIVNY, M., MOREAU, L., AND MYERS, J. Examining the challenges of scientific workflows. *Computer 40*, 12 (2007), 24–32. doi:10.1109/MC.2007.421.

[36] GUARINO, N., GUIZZARDI, G., AND MYLOPOULOS, J. On the philosophical foundations of conceptual models. *Information Modelling and Knowledge Bases 31*, 321 (2020), 1. doi:10.3233/FAIA200002.

[37] GÜTING, R. H. Geo-relational algebra: A model and query language for geometric database systems. In *International Conference on Extending Database Technology* (1988), Springer, pp. 506–527. doi:10.1007/3-540-19074-0_70.

[38] GUTTAG, J. V., AND HORNING, J. J. The algebraic specification of abstract data types. *Acta informatica 10*, 1 (1978), 27–52.

[39] HITZLER, P., AND VAN HARMELEN, F. A reasonable semantic web. *Semantic Web 1* (2010), 39.

[40] HOFER, B., MÄS, S., BRAUNER, J., AND BERNARD, L. Towards a knowledge base to support geoprocessing workflow development. *International Journal of Geographical Information Science 31*, 4 (2017), 694–716. doi:10.1080/13658816.2016.1227441.

[41] HUGHES, J. Why functional programming matters. *The Computer Journal 32*, 2 (1989), 98–107.

[42] KRUIGER, J. F., KASALICA, V., MEERLO, R., LAMPRECHT, A.-L., NYAMSUREN, E., AND SCHEIDER, S. Loose programming of GIS workflows with geo-analytical concepts. *Transactions in GIS 25*, 1 (2021), 424–449. doi:10.1111/tgis.12692.

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

[44] KUHN, W., AND BALLATORE, A. Designing a language for spatial computing. In *AGILE 2015*. Springer, 2015, pp. 309–326. doi:10.1007/978-3-319-16787-9_18.

[45] KUHN, W., HAMZEI, E., TOMKO, M., WINTER, S., AND LI, H. The semantics of place-related questions. *Journal of Spatial Information Science*, 23 (2021), 157–168. doi:10.5311/JOSIS.2021.23.161.

[46] LEMMENS, R., WYTZISK, A., DE BY, R., GRANELL, C., GOULD, M., AND VAN OOSTEROM, P. Integrating semantic and syntactic descriptions to chain geographic services. *IEEE Internet Computing 10*, 5 (2006), 42–52. doi:10.1109/MIC.2006.106.

[47] LI, Z., GUI, Z., HOFER, B., LI, Y., SCHEIDER, S., AND SHEKHAR, S. Geospatial information processing technologies. *Manual of Digital Earth* (2020), 191–227.

[48] LUTZ, M. Ontology-based descriptions for semantic discovery and composition of geoprocessing services. *Geoinformatica 11*, 1 (2007), 1–36. doi:10.1007/s10707-006-7635-9.

[49] MENNIS, J., VIGER, R., AND TOMLIN, C. D. Cubic map algebra functions for spatio-temporal analysis. *Cartography and Geographic Information Science 32*, 1 (2005), 17–32. doi:10.1559/1523040053270765.

[50] NYAMSUREN, E., TOP, E. J., XU, H., STEENBERGEN, N., AND SCHEIDER, S. Empirical evidence for concepts of spatial information as cognitive means for interpreting and using maps. In *15th International Conference on Spatial Information Theory (COSIT 2022)* (2022), Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[51] PAOLUCCI, M., KAWAMURA, T., PAYNE, T. R., AND SYCARA, K. Semantic matching of web services capabilities. In *International Semantic Web Conference* (2002), Springer, pp. 333–347. doi:10.1007/3-540-48005-6_26.

[52] PARREAUX, L. The simple essence of algebraic subtyping: Principal type inference with subtyping made easy (functional pearl). *Proc. ACM Program. Lang. 4*, ICFP (aug 2020). doi:10.1145/3409006.

[53] SCHEIDER, S., AND BALLATORE, A. Semantic typing of linked geoprocessing workflows. *International Journal of Digital Earth 11*, 1 (2018), 113–138. doi:10.1080/17538947.2017.1305457.

[54] SCHEIDER, S., BALLATORE, A., AND LEMMENS, R. Finding and sharing GIS methods based on the questions they answer. *International journal of digital earth 12*, 5 (2019), 594–613. doi:10.1080/17538947.2018.1470688.

[55] SCHEIDER, S., AND DE JONG, T. A conceptual model for automating spatial network analysis. *Transactions in GIS* (2021). doi:10.1111/tgis.12855.

[56] SCHEIDER, S., GRÄLER, B., PEBESMA, E., AND STASCH, C. Modeling spatiotemporal information generation. *International Journal of Geographical Information Science 30*, 10 (2016), 1980–2008. doi:10.1080/13658816.2016.1151520.

[57] SCHEIDER, S., MEERLO, R., KASALICA, V., AND LAMPRECHT, A.-L. Ontology of core concept data types for answering geo-analytical questions. *Journal of Spatial Information Science 2020*, 20 (2020), 167–201. doi:10.5311/JOSIS.2020.20.555.

[58] SCHEIDER, S., NYAMSUREN, E., KRUIGER, H., AND XU, H. Geo-analytical question-answering with GIS. *International Journal of Digital Earth 14*, 1 (2021), 1–14.

[59] SCHEIDER, S., AND RICHTER, K.-F. Pragmatic GeoAI: Geographic information as externalized practice. *KI-Künstliche Intelligenz* (2023), 1–15.

[60] SINTON, D. The inherent structure of information as a constraint to analysis: Mapped thematic data as a case study. *Harvard Papers on Geographic Information Systems 7* (1978).

[61] SUPPES, P., AND ZINNES, J.-L. Basic measurement theory. In *Handbook of Mathematical Psychology*, R. Luce, R. Bush, and E. Galanter, Eds., vol. I. John Wiley and Sons, Inc., New York and London, 1963, pp. 1–76.

[62] TOMLIN, C. D. *Geographic Information Systems and Cartographic Modelling*. No. 910.011 T659g. New Jersey, US: Prentice-Hall, 1990.

[63] TOP, E., SCHEIDER, S., XU, H., NYAMSUREN, E., AND STEENBERGEN, N. The semantics of extensive quantities in geographical information. *Applied Ontology 17*, 3 (2022), 337–364. doi:10.3233/AO-220268.

[64] TRAYTEL, D., BERGHOFER, S., AND NIPKOW, T. Extending Hindley-Milner type inference with coercive structural subtyping. In *Asian Symposium on Programming Languages and Systems* (2011), Springer, pp. 89–104. doi:10.1007/978-3-642-25318-8_10.

[65] VAN DILLEN, S. M., DE VRIES, S., GROENEWEGEN, P. P., AND SPREEUWENBERG, P. Greenspace in urban neighbourhoods and residents' health: Adding quality to quantity. *J Epidemiol Community Health 66*, 6 (2012), e8–e8. doi:10.1136/jech.2009.104695.

[66] VISSER, U., STUCKENSCHMIDT, H., SCHUSTER, G., AND VÖGELE, T. Ontologies for geographic information processing. *Computers & Geosciences 28*, 1 (2002), 103–117.

[67] XU, H., NYAMSUREN, E., SCHEIDER, S., AND TOP, E. A grammar for interpreting geo-analytical questions as concept transformations. *International Journal of Geographical Information Science 37*, 2 (2023), 276–306. doi:10.1080/13658816.2022.2077947.

[68] YANG, J., JANG, H., AND YU, K. Analyzing geographic questions using embedding-based topic modeling. *ISPRS International Journal of Geo-Information 12*, 2 (2023), 52.

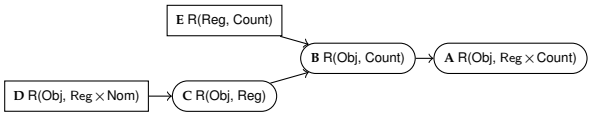
[69] ZHAO, P., FOERSTER, T., AND YUE, P. The geoprocessing web. *Computers & Geosciences 47* (2012), 3–12.

A Tasks and workflows

Scenario 2: Population

What is the number of inhabitants for each neighborhood in Utrecht? [4]

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 [1]. 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**).

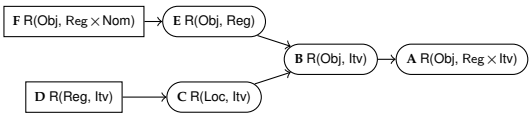


To implement this task, the POPULATION workflow involves a spatial join of cells with neighborhood polygons using a sum operator and using some topological relation.

Scenario 3: Temperature

What is the average temperature for each neighborhood in Utrecht? [4]

The task is to assess an average temperature for each neighborhood in the Netherlands from point measurements[8]. Conceptually, this task corresponds to *averaging a field within the regions of objects*: we first need to interpolate pointwise 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.

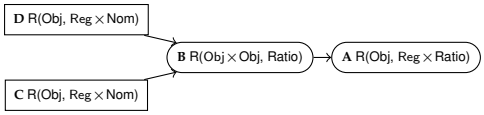
Scenario 4: Hospitals

What is the travel distance to the nearest hospital in California? [24]

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

[8]Temperature time series per station by the Royal Dutch Meteorological Institute (KNMI). [5]

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.

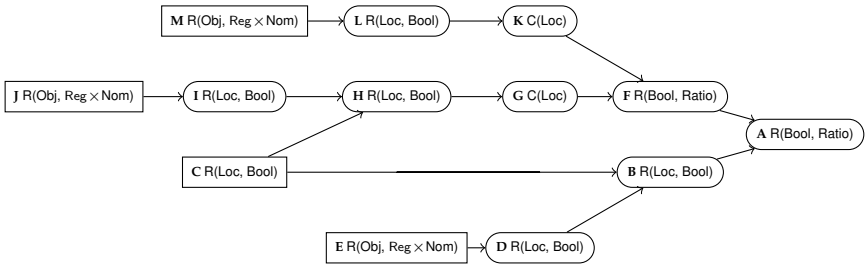


To solve this task in a workflow, we used catchment area (*closest facility*) analysis on a road network (workflow HOSPITALSNETWORK). Alternatively, an equivalent result can be obtained using the *Near* tool based on Euclidean distances (workflow HOSPITALSNEAR).

Scenario 5: Deforestation

What is the impact of roads on deforestation in the Amazon rain forest? [26]

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**).



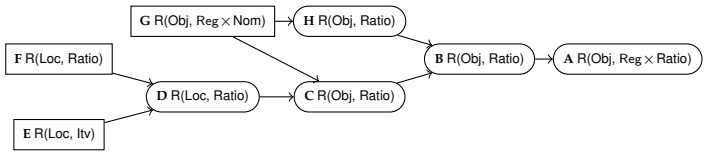
In the workflow, we are given deforested areas as polygons, current and planned roads in the Amazon in terms of line vectors, and we use vector buffer and overlay operations to measure proportions.

Scenario 6: Solar power

What is the potential of solar power for each rooftop in the Glover Park neighborhood in Washington, D.C? [22]

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**).

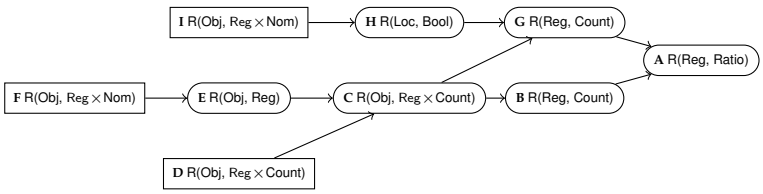


The SOLARPOWER workflow can be implemented using local map algebra on a solar potential raster with both slope and aspect as Boolean constraints using a digital terrain model, which is then averaged over the building polygons using zonal statistics and multiplied by their size.

Scenario 7: Road access

What is the percentage of rural population within 2 km distance to all-season roads in Shikoku, Japan? [21]

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**.

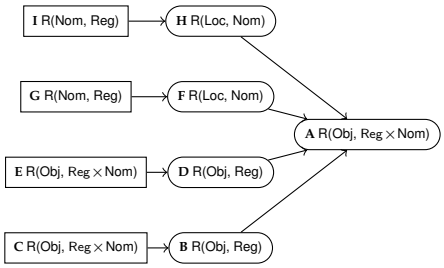


In the ROADACCESS workflow, rural population numbers are given for metropolitan polygons, and we use a simple areal interpolation method (with weighted overlay) to estimate the rural population living within road buffers. We then build a ratio of this number with the total population.

Scenario 8: Aquifer

Which urban areas are at risk from water depletion in Ogallala (High Plains) Aquifer, US? [23]

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**.

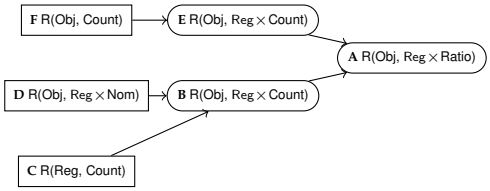


The AQUIFER workflow is implemented entirely using vector polygon operations.

Scenario 10: Malaria

What is the malaria incidence rate per 1000 inhabitants in the Democratic Republic of the Congo? [25]

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.

B 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

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

$$\begin{aligned}
\text{get} &: \mathsf{C}(x) \to x \\
\text{add} &: v \to k \to \mathsf{R}(k, v) \to \mathsf{R}(k, v) \\
\text{nest} &: x \to y \to \mathsf{R}(x, y)
\end{aligned}$$

These operators are inspired by counterparts from set theory: prod3 builds a Cartesian product from a nested relation, which results in a composite relation. 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.

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

Join operators

$$\mathsf{join} : \mathsf{R}(x, y) \to \mathsf{R}(y, z) \to \mathsf{R}(x, z) \qquad \mathsf{get\_attrL} : \mathsf{R}(x, y \times z) \to \mathsf{R}(x, y)$$
$$\mathsf{join\_attr} : \mathsf{R}(x, y) \to \mathsf{R}(x, z) \to \mathsf{R}(x \times y, z) \qquad \mathsf{get\_attrR} : \mathsf{R}(x, y \times z) \to \mathsf{R}(x, z)$$

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

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

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 simple relation. 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 simple relations by their foreign keys on the right hand side.

Operators on functions

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

We use higher-order functions known from functional programming, like function composition, argument swapping, the identity function, and the apply operators to apply a function to each member of some relation (where apply2 applies a binary function to the members of two simple relations). prod combines two simple relations using a binary function. This operator is fundamental to compute any quantified relations from simple relations, like distance relations between object regions. Finally, join_key substitutes the quality of a quantified relation with some quality of one of its keys.

C Type inference algorithm

The type inference algorithm is inspired by [64], 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, it is extended with a form of typeclass constraints with functional dependencies.

C.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 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 [14].

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 $[\theta]\sigma \subseteq [\theta]\tau_i$.

C.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 constraints θ^{C} have not been violated.

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 $C \not\subseteq D$) 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(`$t \dot{\supseteq} C$`)`
    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(`$t \dot{\subseteq} C$`)`
    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** $t \dot{\supseteq} C \in \theta^{\mathsf{S}}$ **do** `AddLowerBound(`$s \dot{\supseteq} C$`)`
    foreach $t \dot{\subseteq} C \in \theta^{\mathsf{S}}$ **do** `AddUpperBound(`$s \dot{\subseteq} C$`)`
Function `AddUpperBound(`$t \dot{\subseteq} C$`)`**:**
  the current bounds on t, if any, are $(t \dot{\supseteq} L), (t \dot{\subseteq} U) \in \theta^{\mathsf{S}}$ **if** $C \subset L$, or neither $C \subseteq U$ nor $C \supseteq U$ **then**
    unification fails
  else
    insert $t \dot{\subseteq} C$ into θ^{S}, remove $t \dot{\subseteq} U$ if needed
Function `AddLowerBound(`$t \dot{\supseteq} C$`)`**:**
  the current bounds on t, if any, are $(t \dot{\supseteq} L), (t \dot{\subseteq} U) \in \theta^{\mathsf{S}}$ **if** $C \supset U$, or neither $C \subseteq L$ nor $C \supseteq L$ **then**
    unification fails
  else
    insert $t \dot{\supseteq} C$ into θ^{S}, remove $t \dot{\supseteq} L$ if needed