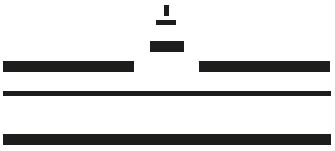


University of Münster

Master's Thesis

Communicating meaning and purpose of spatio-temporal data analysis

Author:
Matthias Hinz

Supervisors:
Prof. Dr. Edzer Pebesma
Dr. Simon Scheider

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Geoinformatics

in the

Spatio-Temporal Modelling Lab (STML)
Institute for Geoinformatics (ifgi)

September 27, 2016

Declaration of Academic Integrity

I hereby confirm that this thesis on "Communicating meaning and purpose of spatio-temporal data analysis" is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

_____

(date and signature of student)

"*Music is a combination of sounds differing, first, in regard to space, secondly, in regard to time. The space between two or more sounds I call the degree of rapidity at which the air vibrates, in consequence of some cause which cleaves it. The cause of the conception of space is movement; movement necessitates the conception of points, or moments.*"

Leo Tolstoy[1] - July 14th, 1850

[1] Tolstoy, L., Chertkov, V. G., Hogarth, C. J., & Sirnis, A. (1917). The diaries of Leo Tolstoy. New York: Dutton, p. 219.

UNIVERSITY OF MÜNSTER

Abstract

Faculty of Geosciences
Institute for Geoinformatics (ifgi)

Master of Science in Geoinformatics

Communicating meaning and purpose of spatio-temporal data analysis

by Matthias HINZ

Spatio-temporal data analyses can be hard to understand. Given the complexity of advanced analysis procedures and big, heterogeneous datasets, comprehending the work of others can be difficult if not impossible, if no adequate documentation is supplemented. On the other hand, manual creation of metadata and documentation is a cumbersome work that scientists neglect because it does not pay off immediately.

For a given analysis that is shared amongst researchers, relevant information about meaning, purpose and scientific presuppositions of Spatial Statistics are often absent. Also, the scientific community has not yet established means to express and share this information in a systematic way, i.e. by making use of semantic ontologies and reference systems.

This thesis shows how to ease the tedious work of metadata creation and how to create metadata that is expressive regarding meaning and purpose. It presents the 'SpatialSemantics' package for R, which is a prototypical library and extension of the R environment for statistical computing. It exemplifies how during execution of an analysis a so-called spatio-temporal data derivation graph is generated in the background, which can be visualized, shared and reviewed, as it is a documentation of the analysis.

The metadata generation is semi-automated; on the one hand, the effort on the part of the user is minimal because the recording engine populates the graph automatically with semantics. On the other hand, users can specify which part of the analysis, they can apply semantic annotations and define checks for semantic consistency and validity. The user can interactively query semantics during the analysis session and receives warnings if a semantic check fails.

This work concludes about how to better communicate our work, and how to share it with those people we collaborate with.

Acknowledgements

I would like to thank Prof. Dr. Edzer Pebesma and Dr. Simon Scheider for supervising my work and spending their valuable time for discussions and feedback. That always helped me to get clarity on the topic and how to carry out my research. It was a pleasure to work together with such great minds.

My parents made a lot of that possible what I achieved in my life up till now. They supported me during all years of my studies, and therefore I am grateful with all my heart. Likewise, I want to thank all other family members, relatives, and friends for being by my side and giving me mental support during this period of my life.

The international student dormitory 'Volkeningheim' was my home in Münster during the past five years and while I was writing this thesis. I am entirely thankful for the supportive community in this place that taught me things about life that cannot be learned from scientific theory.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

DDG	**D**ata **D**erivation **G**raph
GI	**G**eographic **I**nformation
GIS	**G**eographic **I**nformation **S**ystem
HOL	**h**igher-**o**rder **l**ogic
IID	**i**nstance**id**entifier
LOD	**L**inked **O**pen **D**ata
MCO	**M**ars **C**limate **O**rbiter
NASA	**N**ational **A**eronautics **S**pace **A**dministration
OPM	**O**pen **P**rovenance **M**odel
OWL	**W**eb **O**ntology **L**anguage
RDF	**R**esource **D**escription **F**ramework
SPARQL	**S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
SpODT	**Sp**atial **O**blique **D**eciscion **T**ree
W3C	**W**orld **W**ide **W**eb **C**onsortium
XML	E**x**tensible **M**arkup **L**anguage

Chapter 1

Introduction

1.1 Preface

In December 1998, the Mars Climate Orbiter (MCO) was launched from Cape Canaveral and sent on a 416-million-mile journey to Mars. Its mission was to find water on the planet by tracking seasonal changes and movements of vapor and dust for one entire Mars year (687 Earth days). After about nine months, MCO arrived and started its Orbit insertion trajectory, but after 4 minutes the signal broke up. It was never recovered again. NASA lost the spacecraft worth 125 million dollars. The root cause of the accident was a failure in modeling velocity changes of the spacecraft. A ground software file used English units instead of metric units and then did not convert the output from pound-seconds (lbf-s) into Newton-seconds (N-s). As a result, all subsequent predictions were underestimated by the conversion factor 4.45. That happened despite that a Software Interface Specification (SIS) explicitly specified the output unit; the document was not followed (Stephenson et al., 1999).

Dr. Edward Weiler, NASA's associate administrator for space science said: "People sometimes make errors. The problem here was not the error; it was the failure of NASA's systems engineering, and the checks and balances in our processes to detect the error. That's why we lost the spacecraft (Isbell, Hardin, and Underwood, 1999)." Many investigations followed, questioning why the project failed. NASA itself studied the case intensely and published a report on lessons learned and recommendations for further projects (Griner and Keegan, 2000).

Other researchers picked up the topic later (Sauser, Reilly, and Shenhar, 2009) and argued that project failure often goes beyond technical reasons and management could have prevented the accident. Sauser et al. suggest a better upfront assessment of the program's uncertainty and complexity, and installing proper managerial systems that would detect errors ahead of time.

A contributing factor to the failure was NASA's policy of faster, better, cheaper, which was not implemented adequately; the project was carried out under high constraints of time and budget. One team leader said that "it was mandatory that we cut corners, primarily in the review and quality engineering process (Sauser, Reilly, and Shenhar, 2009)" (Stephenson et al., 1999).

As shown above, the NASA incident serves as an example to the importance of management. However, failures occurred at various levels: management, engineering, programming and interpersonal interaction. Eventually, it can be broken down to miscommunication. A look at current developments in data science suggests another, different resolution that complements the management approach: what if we implemented more reliable, more transparent workflows that carry out semantic reasoning over data processing? Most development environments and data processing applications will throw errors or warnings only when a program code is syntactically invalid, or when a task is not feasible. But many of those errors are detected even before compilation and execution. On the other hand, seldom applications throw errors when data manipulation violates semantic premises, especially if it does not comply with the underlying scientific assumptions. So-called domain-aware semantic applications can do so because they make use of domain-specific knowledge and concepts formalized as reference systems and ontologies. An error such as mixing up observations of different units such as pound-seconds and Newton can be detected as a semantic error by making use of an ontology of physical measurement units. Applications are called provenance-aware if they keep track of data generation and transformations and thus provide a record of all entities and procedures involved in the delivery of their output. Such provenance records have an added-value if they are annotated with domain-specific semantics. It enables posterior semantic validation without the need to re-execute the process as well as meaningful documentation of the same. Hence, the record could be shared with third parties, which may review the analysis independently. The use of provenance- and domain-aware software, in turn, would allow project managers to reasonably cut down the overhead in the test and review process - a significant advantage when budget and time are limited.

There is a particular demand for meaningful data analysis supported by domain-specific semantics in the field of Spatial Statistics. This is the motivation of this work. The following paragraphs will elaborate this problem and explain how this thesis addresses it.

1.2 Background

Modern data science not only addresses the handling and processing of data itself, but also descriptive information thereof that fosters understanding. Metadata tells the computer how to handle and access data structures (file formats), how to display and visualize it. It can refer to measurement units and reference systems, which enables meaningful interpretation of data and thus turning data into information.

An important research field in this context is provenance. According to the W3C Provenance Incubator Group, provenance and metadata are related but distinct terms. Provenance is often represented as metadata and thus defined as “a record that describes entities and processes involved in producing and delivering or otherwise influencing that resource (Gil, Cheney, et al., 2010)”.

The incubator group further came up with a listing of key provenance dimensions, from which the scope of provenance information becomes apparent (Gil, Cheney, et al., 2010). They structure listing into the following categories of management, content and use:

Content: Object, Attribution, Process, Versioning, Justification, Entailment

Management: Publication, Access, Dissemination, Scale

Use: Understanding, Interoperability, Comparison, Accountability, Trust, Imperfections, Debugging

These dimensions and requirements of provenance were elaborated in detail by the incubator group and later taken up by the W3C Provenance Working Group. This follow-up aimed to create a cross-disciplinary Web standard for provenance information and with this, defined the PROV-data model (Gil, Miles, et al., 2013; Missier, Belhajjame, and Cheney, 2013). The data model is implemented in well-established technologies on the Semantic Web (XML schema, OWL, RDF). PROV is domain-agnostic, meaning that it is not capable of holding domain-specific information by itself. For this reason, the authors encourage developers to create PROV-extensions according to the requirements of their applications and scientific disciplines.

Yet, before the development of PROV, the idea of semantic provenance emerged; Missier, Sahoo, et al. (2010) pointed out that most former research on provenance acquisition focused on causal relationships amongst data products and neglected the semantic character of these products. With Janus, they provided a proof-of-concept for a domain-aware provenance model that enhances provenance graph with domain-specific annotations. Janus was designed for bio-informatics and enables user-scientists to query the provenance of scientific workflows with domain-specific terminology, for instance:

"Amongst all genes that are known to perform a certain biological function, list those that are involved in a certain pathway," (Missier, Sahoo, et al., 2010).

Missier et al. proposed to embed provenance graphs as data on the Web of Data, thereby following the conventions of Linked Open Data (LOD). They demonstrated that this approach serves to answer an even wider range of queries; the query above generally cannot be solved if the provenance graph that is queried does not include the *concept* of the biological function of genes. The solution to this problem is to query a large public database for genes that occur in the provenance graph and for their function. This information is then used resolve the overall query. Hence, integration of provenance on the Web of Data allows executing complex, domain-specific queries supported by multiple sources of information.

Currently, there is no comprehensive provenance model for Spatial Statistics that is comparable with Janus, despite some ongoing research in the field of semantics (See Chapter 2). On the other hand, there is a concrete demand for domain-specific inquiries when such analyses are communicated. Roger et al. state in their book about "Applied Spatial Data Analysis in R" that "spatial data *analysis* is concerned with questions not directly answered by looking at the data themselves. These questions refer to hypothetical processes that generate the observed data. Statistical inference

for such spatial process is often challenging but is necessary when we try to draw conclusions about questions that interest us (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013, p. 1)". In the same chapter, they state that "Statistical inference is concerned with drawing conclusions based on data and prior assumptions. The presence of a model of the data generating process may be more or less acknowledged in the analysis, but its reality will make itself felt sooner or later." They criticize that frequently, "the prior assumptions are not made explicit, but is taken for granted as part of the research tradition of a particular scientific discipline. Too little attention typically is paid to the assumptions, and too much to superficial differences (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013, pp. 11-12)." Such statements hint at a lot of vagueness and uncertainty in the common research practice, yet they comprise a concrete appeal to pay more attention to research assumptions and models.

In a recent publication, C. Stasch, Scheider, et al. (2014) stressed the importance of observations in research: "Observations form the basis of empirical and physical sciences. They provide samples for a process of interest, enabling us to infer knowledge about this process and to evaluate assumptions and hypotheses. In order to infer knowledge or test hypotheses about a process, statistical models and procedures can be applied to observations." According to the authors, syntactical integration of observation in statistical modeling frameworks is not an issue, but the semantical integration is a challenge. In particular, determining which method is appropriate for which kind of data in an automated fashion is an open research question. Stasch et al. yet identified additional problems in this context; in interdisciplinary settings, large volumes of data from heterogeneous resources are often combined by researchers without specific domain knowledge. Furthermore, the distance between those who collect data and those who analyze it in general becomes larger. Thus there is a need for semantic metadata that bridges the knowledge gap between domain experts and data scientists.

Another point brought up in the paper is model interoperability and model reuse; it is said that, according to a NASA report from 2012, NASA scientists would spend about 60% of their time on making data and models compatible (C. Stasch, Scheider, et al., 2014). To this example, it may be added that abortive reuse of software from NASA's Ariane 4 program in 1996 led to the explosion of the first Ariane 5 rocket (Ariane V88), a loss of about 360 million US-dollars and a delay of the Ariane 5 program for one year (Le Lann, 1997). For such reasons, model- and data interoperability as well as semantic interoperability (see Chapter 2) are a particular concern of research.

Given this assessment of particular needs in Spatial Statistics and Geographic Information Science, Stash et al. proposed to formalize analysis procedures and introduced a novel notion of the meaningfulness of prediction and aggregation. This was specified in a semantic theory in higher-order logic (HOL). Procedures for observation, prediction and aggregation are formalized as functional types which are assigned to data sets. The theory was designed to enable meaningfulness checks when either prediction or aggregation functions are applied to data.

A subsequent paper from Scheider et al. (2016) built upon this theory

and drafted it as a generative algebra for modeling spatio-temporal information generation. The concept of this algebra will be explained in Chapter 3 because it forms part of the methodology and approach presented in this thesis.

According to the Scheider et al. current solutions provide insufficient descriptions about how data is generated. They argue that "in data analysis, we crucially depend on such meta-information which goes beyond the data type. [...] [W]e present a generative model of spatiotemporal information that precisely makes these distinctions, by describing how information is generated, including raw observations as well as derived products." In the following sentences they stress important aspects of the *meaning of data*, more precisely, about their *function and purpose* and on the advantages of making this information explicit; "If the who, how, and why of the entire production chain is known, meaningful analysis can be inferred and added on top of an existing data product and datasets can be queried on the basis of how they were derived or which products they could be turned into. This enables *smart data* as opposed to *smart applications* (Scheider et al., 2016, p. 1981)." Again, the authors aimed for domain-specific semantic queries, such as mentioned above. Apart from those queries, the model could help automating the description of a derivation process and thus enable semantic-aware software for Spatial Statistics. This was also expressed by the authors. They indicated that it is a challenge "to map tools to generation types, and to annotate datasets with their derivation graph. The latter should be automatized as much as possible in order to avoid work for data publishers. For example, a tool such as R (R Core Team, 2016) could generate a derivation graph automatically in the background as data producers generate datasets. (Scheider et al., 2016, p.2001)." This statement largely addressed in this thesis; it gives direction to one of the research objectives described in the following section.

In conclusion, the algebraic model presented by Scheider et al. (2016) can be used to address open research questions and demands in the fields of Spatial Statistics, as outlined in this section. This thesis aims to show *how* they can be addressed.

1.3 Research aim and objectives

This work represents applied research insofar that it aims to explore and suggest practical solutions for a common problem in Spatial Statistics: communication of meaning and purpose of spatio-temporal data analyses. The main focus lies on analyses that are carried out in software environments for Spatial Statistics, but also, users of Geographic Information Systems (GIS) could relate to this work. People that shall benefit from this research are those who use spatio-temporal analysis as a tool.

The thesis builds upon primary research on the subject of *Meaningful Spatial Statistics*[1](C. Stasch, Scheider, et al., 2014; C. Stasch, E. Pebesma, and Scheider, 2014; E. J. Pebesma and C. Stasch, 2014; Scheider et al., 2016). Notably, the publication from Scheider et al. (2016) is fundamental to this work because it defines an algebra for modeling spatio-temporal data generation

[1]Resources and publications about Meaningful Spatial Statistics are available at http://meaningfulspatialstatistics.org/.

and derivation using types of reference systems. This modeling algebra will be used throughout the thesis in order to demonstrate semantic modeling on spatio-temporal data analysis.

In order to carry the research aim into effect, a number of research objectives are declared in the following. A research objective is considered fulfilled if the requirements stated below the objective are met:

Objective 1: Design and implement a framework for the semantic provenance of spatio-temporal analysis

This work aims to provide a proof-of-concept to the existing algebraic model by implementing it in a prototypical extension package for the statistical computing and graphics environment R (R Core Team, 2016).

The prototype shall facilitate communication of the semantic provenance associated with spatio-temporal data analysis. Therefore it shall perform the following functionality:

- The prototype shall be a domain-aware application for Spatial Statistics.
- The prototype shall be documented and tested.
- The prototype shall generate a spatio-temporal derivation graph in the background while an analysis is carried out. It shall be possible to save and export the graph as appropriate data formats that enable sharing, visualization, and metadata analysis.
- The prototype may annotate data and objects automatically with semantic metadata if the required information is explicit or if implied by reasoning. It must be done manually if crucial information is missing or needs refinement.
- The prototype shall prompt semantics-related messages and warnings while an analysis is carried out. Users may also query for semantics and provenance any time during and after the execution.

Objective 2: Enable meaningful communication of spatio-temporal analysis

The meta-information that the prototype communicates shall facilitate a domain-specific understanding of spatio-temporal data analysis. The information shall emphasize *meaning*, *purpose*, and *function* of data and procedures that form part of the analysis. Precisely, the information shall encompass the following aspects:

- Research assumptions about the hypothetical process that generates observed data, in particular, *how* and *why* data was generated or transformed.
- The meaning of individual spatio-temporal datasets, in terms of their representation, function, and purpose.

- The validity of data generation and transformations in terms of *meaningfulness* by evaluating function and purpose of their input data. Validity checks may further compare the de facto semantics of an executed procedure with previous semantic assumption stated by the user-researcher, i.e. the expected semantics.
- User-notification about semantic inconsistencies and missing assumptions on observations and hypothetical processes, if an analysis cannot be assessed in terms of meaningfulness without uncertainties.

Objective 3: Demonstrate the solution on common analytical problems of Spatial Statistics

This work shall demonstrate the prototype by applying it to selected use cases that deal with common analytical problems of Spatial Statistics. These use cases shall consist of analyses derived from published scientific works. The use-case analysis shall be functionally equivalent to the approach presented in the publication, which assures that the use case has a certain degree of significance and approval by the research community.

For demonstration purposes and as a concession to the limited scope of this thesis, the use cases may be simplified and reduced to certain aspects of the originally published analysis.

Objective 4: Evaluation of the semantic framework

The overall findings of this work shall be evaluated against the research objectives according to the requirements set here. As a requirement to the research aim, the findings shall be summarized, and thus conclusions to the research questions shall be drawn.

1.4 Research Questions

Arising from the research aim and objectives, four prevalent questions become apparent; Addressing them is the function and purpose of this work.

1. How can the generative algebra introduced by Scheider et al. (2016) be possibly implemented in a software environment for Spatial Statistics?
2. How can research assumptions about observations and the hypothetical processes that generate them be made explicit in a spatio-temporal analysis?
3. How can it be ensured that spatio-temporal data generation and derivation operations are meaningful?
4. How can the meaning and purpose of spatio-temporal analysis and their components be communicated and shared?

1.5 Structure of this Work

Chapter 1 provides a background about the subject of this work. It determines problems and open research questions in this context and accordingly determines research objectives of this work, which are described regarding their requirements. It concludes with four research questions that are derived from the research objectives and steer this thesis in its general direction.

The following Chapter 2 introduces related works in addition to those works already mentioned in the previous chapter. Many of these works inspired this research. Hence they are referred to in the following chapters as well in order to illustrate or explain statements made by this work.

Chapter 3 explains the general approach taken by this work, including how the research objectives and -questions are addressed and which existing works, concepts and software are applied. The chapter also explains which alternative approaches were not taken and why.

Chapter 4 introduces the 'SpatialSemantics'-package in R, which is the aimed implementation of a framework for semantic provenance of spatio-temporal analysis. It thus corresponds to research objective 1.3. Also it corresponds to the research objective 1.3, because the implementation includes functionality that enables meaningful communication of spatio-temporal analysis.

Chapter 5 demonstrates the usage of the R package and applies it to three use case analyses that address common analytical problems of Spatial Statistics. The chapter shows that using the package yields useful results regarding semi-automated creation of semantic meta-information. Therefore the package is a contribution to meaningful communication of spatio-temporal analysis. The chapter corresponds to research objective 1.3 and expands upon the research objectives 1.3 and 1.3 by giving practical examples in this regard.

Chapter 6 evaluates the results of this work by comparing them with the research objectives 1.3, 1.3, and 1.3. Consequently, the chapter is divided into three sections that each correspond to one research objective. Each section matches the requirement of the objective with the results of this work and thus concludes on whether the particular research goal is considered fulfilled. It also mentions restrictions of this fulfillment if there are any. The chapter itself corresponds to research objective 1.3.

Chapter 7 addresses challenges of this work and expands upon the restrictions mentioned in Chapter 6. It discusses pros and cons of the current solution and provides new ideas related to the outcomes of this work that can be addressed by further works.

Chapter 8 draws the final conclusions of this work by addressing the research questions stated in the beginning. It expands upon the research objective 1.3 because the conclusions are drawn from the evaluation of the research results in Chapter 6.

Chapter 2

Related Works

This chapter introduces related works in the field of provenance and semantics. The reference to semantics is kept short because semantics and their particular requirements in the field of Spatial Statistics were already discussed in the background section 1.2. Section 2.2 emphasizes particular approaches to enable provenance tracking in R that inspired the development of the 'SpatialSemantics'-package presented in Chapter 4.

2.1 Provenance and semantics in data science

Many workflow management systems and workbenches already support the collection of provenance information, for instance, Kepler (Altintas, Barney, and Jaeger-Frank, 2006), VisTrails (Scheidegger et al., 2008), the Trident (Barga et al., 2010) and Taverna (Alper et al., 2013).

The Trident[1] is a scientific workflow workbench based on the Windows Workflow Foundation (WF), which is a workflow engine and library embedded in the Windows operating system. The Trident supports the provenance retrieval with the Open Provenance Model (OPM) (Moreau et al., 2011), which can be a predecessor of the PROV model introduced in section 1.2. The project is no longer active since 2013.

Taverna is an open source domain independent workflow management system consisting of a suite of various tools; the engine is written in Java. It can be used from the Taverna server, Taverna desktop workbench or as a command-line tool. Taverna also provides interfaces to other software and tools, including the R environment for statistical computing (R Core Team, 2016). In the past, support for the OPM model and a data model for the Janus provenance model (Missier, Sahoo, et al., 2010) were implemented. The Taverna-PROV plugin currently enables provenance recording using the PROV model (Gil, Miles, et al., 2013) and the Resource Description Framework (RDF) (Klyne and Carroll, 2004). Furthermore, the PROV model is extensible and can be enhanced with domain-related semantics (compare with section 1.2).

PROV and RDF are both standards recommends by the World Wide Web consortium (W3C). Implementing these standards allows publishing provenance metadata according to the principles of Linked Open Data (LOD) so that they are accessible from the Semantic Web. It also enables usage of other semantic technologies implementing W3C standards, so that provenance can be analyzed using the SPARQL query language (Group, 2013).

[1]The web site of the no longer active Trident project is available at https://tridentworkflow.codeplex.com/ (Last access on Sept. 19th 2016).

In deviation from how provenance is captured by Taverna and the Trident, Plale, Cao, and Aktas (2011) claim that provenance capturing is a standalone activity and presented Karma[2], a provenance system that is independent from workflow systems and free of the assumptions that workflow systems have. The Karma system has been applied to geoscientific workflows, notably by (Jensen et al., 2013). The system consists of four different layers: The *Provenance Creation Layer* (1) is an interoperability point to the application from which provenance is captured. It generates provenance events which are received by the *Capture layer* (2). The capture layer provides API and web services for capturing provenance and passes received provenance events to the *Representation layer* (3). The representation post-processes the data and stores it in an organized form. It also applies reasoning so that knowledge can be inferred from the data that is not directly apparent. The *Access Layer* (4) interacts with the representation layer and provides tools and an interface to the user. The Karma system has been applied in several cases to geoscientific workflows, notably by (Jensen et al., 2013).

2.2 Provenance in R

(Lerner and Boose, 2014) take a stand that opposes most of the aforementioned approaches: They argue that existing provenance systems pose a high technological barrier to scientists. Amongst other examples, they mention Kepler, Taverna, and Vistrails. The paper explains that in order to capture provenance, scientists have to adopt these additional tools and techniques that may not work well with the software that they are accustomed to use. For this reason, Lerner et al. propose to implement provenance collection capabilities in the tools that scientists are actively using. They present a set of tools made for collecting, visualizing and querying provenance data from commands or scripts executed in R. The toolset consists of an R package named *RDataTracker* and a standalone visualization tool called *DDG explorer*. The approach of provenance collection is semi-automated: users add calls to functions from the R package to their R script or type then interactively in the R command line interface. When these functions are executed, they evaluate the runtime-state of R, including variable name bindings and the R command history. The library further provides means to structure an R script into individual sections, called procedures. The recorded provenance is represented as data derivation graph (DDG). Nodes are either procedures, R expressions, or data that is read or written. Procedures are collapsible nodes that provide a layer of abstraction and enabling scientists to view the (manually within the script) annotated derivation graph in different levels of details. Lerner et al. oppose a fully automated retrieval of provenance information because the data may become voluminous if all intermediate values of the execution are saved. Instead, they propose to collect only data that has significant value to the scientists. Hence they seek a powerful payback for a small investment on the part of the scientist.

[2]Kadmandu, the successor of Karma, is W3C compliant and available online at http://d2i.indiana.edu/provenance_komadu (Last accessed on Sept. 19th, 2016).

Another effort to implement provenance capabilities in R was formerly presented by (Silles, 2014). In his work, he implemented provenance capabilities in CXXR, which is a variant of the R language and environment that is reengineered in C++.[3] Provenance is obtained during the execution using hooks in the call-interpreter-loop of R, i.e. the runtime state of R is evaluated each time after a new expression is entered in the R console. Users can interactively query the provenance of variables using the functions *provenance* and *pedigree*. When a user thus queries the provenance of a variable x, the corresponding function returns details about its current name binding and dependencies, including the expression by which it was created, the date and time when it was created, the variables corresponding to the data from which x was directly derived (parents) and the variables which were created in dependence of x (children). The pedigree-function returns a list of all those expressions which lead up to the creation of x so that x can be reproduced by re-executing this sequence of commands. Silles also demonstrated how to represent provenance data from CXXR by the W3C PROV model and how to export and serialize it in RDF.

[3]The successor of CXXR, named Rho, is actively developed at the moment and as a prototype available on https://github.com/rho-devel/rho (Last access on Sept 19th, 2016).

Chapter 3

Methodology

3.1 Overview

This work addresses a field of research with little has been addressed before, especially in a practical manner. It relates to semantic provenance for Spatial Statistics and the subject of Meaningful Spatial Statistics. Basic research on this topic has been carried out by Scheider et al. (2016) resulting in an algebra for modeling data generation and transformation processes in Spatial Statistics. At the current stage, the modeling algebra is considered incomplete by their authors. A formal proof of completeness does not exist yet and demonstrations were only done on an abstract level (Scheider et al., 2016, p. 2000). This thesis presents applied research that seeks to put the algebra into practice and thus solve communication problems when dealing with spatio-temporal data analysis. It is the first attempt to implement the algebra in a software environment for Spatial Statistics that also involves an automated generation of derivation graphs.

Because little has been done in this field, an unstructured exploratory approach is chosen that allows much flexibility in research. A structured mode of inquiry, for instance, a controlled experiment on communicating data analyses to researchers is rejected because of the limited scope of this master thesis and because such labor-intensive investigation approaches are more promising when the research fundamentals are fairly elaborated. The approach can be classified is qualitative work, because the research questions are not addressed with any quantitative analysis.

The remainder of the chapter is organized as the following: First, the approach is outlined and described how research objectives and questions are addressed in this thesis and why they are addressed in this way. In the next section describes a couple of alternative approaches and describes why they were rejected. Afterwards, the research foundations that are incorporated in this thesis are described, which in essence consist of the modeling algebra. The final section explains the choice of software that was used to implement the prototype

3.2 General approach

In order to address research question 1, an extension package for the R language and end environment for statistical computing (R Core Team, 2016) was created. This package consists of a software prototype for capturing semantics and provenance of spatio-temporal analyses and makes use of the modeling algebra introduced above.

During the development process a bottom-up strategy was followed:

(1) At first, R was extended with functional components that enable the collection of provenance information while an analysis is carried out over the R command line. From this information, a generic data derivation graph is computed that maps all data i.e. data sets, literals (e.g. strings, numbers, logical values), unevaluated functions and function arguments to *object nodes* and maps all calls, i.e. function calls and primitive operations to *call nodes*. Each call node is linked to in- and output with corresponding arrows so that the derivation history of each data piece can be retraced by traversing the directed graph upwards to its parent nodes.

(2) Second, enriching the graph with semantic annotations was enabled. At first, annotations were added manually to some example graphs, mainly corresponding to use-case 5.1. Then functional components were implemented that automatically infer as many of these annotations from the existing metadata, basically data attributes and data types. Afterwards, semantic information was identified that could not be inferred with the available data. As a consequence, functional components were implemented that facilitate providing user-annotations to data. This particular activities target research question 2 (i.e. how to make research assumptions explicit)

(3) The third step in the approach addressed research question 3 (how to ensure that particular operations are meaningful): a mechanism for detecting errors and missing assumptions were implemented. In order to do a semantic assessment, it first has to be assured that all required metadata the datasets involved in an analysis is available. If not, corresponding warnings are prompted to the user. It also was addressed that this missing or ambiguous information is reflected in the derivation graph. Provided that all semantic information is available, semantic validation can be carried out when an analysis is executed. Chapter 4 explains how this is done.

(4) Finally, the prototype was tested on various R scripts and on the use-cases (5). Bug-fixes and feature improvements where applied when this was necessary.

In order to evaluate the research activities and results, four research objectives were declared alongside with individual requirements on when an objective is considered fulfilled (see section 1.3). Therefore, the research findings were described and matched against the mentioned requirements (Chapter 6).

The evaluation in particular concerns the implementation of framework for semantic provenance of spatio-temporal analysis (technical requirements, research objective 1.3), enabling meaningful communication of spatio-temporal analysis (semantic requirements, research objective 1.3) and how the prototype is demonstrated (use-case requirements, research objective 1.3)

The fourth research objective 1.3 is the evaluation itself. As stated in the introduction, this goal is not fulfilled just by matching all requirements with findings but also requires drawing meaningful conclusions about all four research questions from those findings (see chapter 8). Most importantly, a conclusion should be drawn on question 4, on how to communicate and share meaning and purpose of spatio-temporal analysis and their components. The answer will be the overall résumé of this thesis.

3.3 Alternative approaches

Establishing a completely automated mapping from an analysis in R to a spatio-temporal derivation graph as specified by Scheider et al. (2016) exceeds by far exceed the scope of this thesis. The associated challenges and further work are described in chapter 7. In this regard, a bottom-up approach is the fastest strategy to produce valuable intermediate results. As shown in chapter 4 and 5, the prototype already offers much reusable functionality that can be applied to many R-scripts.

Instead of employing a bottom-up approach, the prototype could be implemented using a top-down of mixed bottom-up/top-down strategy. A top-down strategy would be to start with an abstract use-case that can be expressed with the modelling algebra, then to create a derivation graph and an R script that exactly match the abstract use-case, and then to build a software prototype that generates this exact derivation graph from the R script, starting from designing interfaces and the overall software architecture.

A top-down approach was generally rejected for various reasons: As the algebra has little been put into practice before, we do not exactly know what the algebra can express and what it cannot express. Therefore, starting with a practical use-case analysis that is not manipulated according to the algebra and as close as possible to common practice is the best strategy to explore patterns of meaning that cannot be expressed yet Findings can induce refinement and improvement of the modeling algebra. Starting the development with designing software components and interfaces is only mandatory for large software projects, but it can be neglected for development in this scope.

It is common practice in software engineering to do a detailed requirement analysis ahead of developing software. Such requirement analyses involve studying of use-cases or stakeholder-interviews. The objectives and requirements of this work are not deduced from such practices, but neither they are arbitrary: They are in accordance with problems and needs stated in the background chapter 1.2 that were previously expressed in accredited scientific publications. It is also shown in the related work chapter 2 that other scientists addressed similar problems in neighboring research fields. On the other hand, the use-cases exemplified in chapter 5 suggest an added value to the field of Spatial Statistics which in hint-sight also justifies the research objectives. Finally, a strict software engineering approach was rejected because this work does not represent a software project. Its major outcome neither is a software product. The prototype is a means to exemplify the modeling algebra in an environment for Spatial Statistics and the major outcomes are scientific conclusions to the research questions stated above.

Another approach to addressing the overall research and aim and especially research question 4 would be a structured user study on the meaningful communication of analysis. Even without a working software prototype, the software could be pen-and-paper based: For instance, participants would receive a printed R script compiled together with all feasible analysis results, i. e. plots and printed messages as they are rendered by Sweave or knitr (See section 2.2). Next, they would have to answer a questionnaire about this analysis. Some of these questions may concern the meaning and purpose of particular datasets or whether they consider the analysis itself

as meaningful. Participants would also answer self-assessment questions on how good they understand the analysis. The study would be evaluated regarding how many right answers the participants give and in terms of the participants' self-assessment. The setup would be manipulated by dividing the participants in to groups and handing some participants supplemental information material to the analysis, such as a data derivation graph. By comparing answers from participants who had supplements and with those who had none, conclusions could be drawn upon which kind of representation of analysis semantics improve an understanding and thus mediate information about meaning and purpose.

Such approaches were rejected because they do not address questions about how to implement the algebra by Scheider et al. (2016) and how to ensure that particular operations of an analysis are meaningful. Even if the study could be done without a prototype it would be beneficial to do research on these questions first. The setup is far more convincing if participants are handed derivation graphs that are real, not just in an abstract way, derived from an R script. A user study then could evaluate both, usability of the prototype and expressiveness of spatio-temporal derivation graphs.

Combining both approaches, that is developing a prototype and then carrying out a user study with the prototype, was rejected because it would exceed the scope of this thesis. As the discussion shows (chapter 7), this research had to address many technical and conceptual challenges. Reducing the development time in favor of user studies would lead to results of minor significance. As stated above, this research explores a problem that little has been addressed before. It aims to suggest a practical solution and thus provide well-elaborated intermediate results to be refined in further work, not to research one problem with the pretense of completeness.

3.4 Modeling data generation and transformation

This section explains the modeling algebra as specified by Scheider et al. (2016), as far as necessary for the comprehension of this work. It also explains how and why the algebra was chosen as a research foundation. For getting a throughout understanding of the algebra, please refer to the originally published work and the previous publications it is based on (Scheider, 2012; C. Stasch, Scheider, et al., 2014; C. Stasch, E. Pebesma, and Scheider, 2014; E. J. Pebesma and C. Stasch, 2014).

The introduced model describes the origins of data in terms of *how* they were obtained by means of conceptualized data generation and derivation procedures. Corresponding to the data's *purpose*, these procedures imply *why* data were obtained in terms of research assumptions. Corresponding to the data's *function* and based on the how and why of data generation, it can be inferred which possible transformations and conversions can be applied. Likewise. it can be inferred if an operation applied to the (annotated) data is *meaningful*.

Figure 3.1 shows the general approach taken by Scheider et al., which is explained in the following paragraphs. Data are sets of organized symbols. In terms of the algebra, data represent sets or list of tuples of referents. A referent is a concept that is specified by a basic reference system type, called basic type. Referents are created by *data generation functions* and can

be derived from other referents that are inputted to those functions. Data generations functions are constructed *derivation functions* and thus can be derived from other generation functions. Note that deriving one referent from another really means deriving one *concept of data* from another. The derivation and generation of *data* are modeled by *data generators*, which are functions that have generation functions as a conceptual input. Derivation functions and data generators are based on a set of primitive operations, which are the essential components of the algebra.

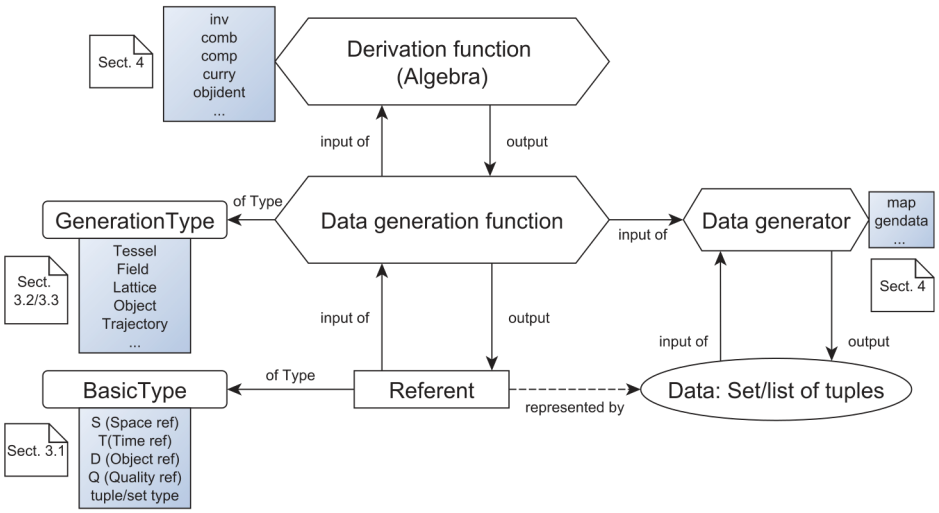


FIGURE 3.1: General approach to the algebraic model. Annotations denoting a document section refer to the original publication (Scheider et al., 2016, p. 1984)

Basic types

The algebraic model defines a set of basic reference system types, called basic types: S, T, D, and Q stand for types of possible spatial locations, moments, discrete entities and quality values respectively. The basic type $bool$ refers to logical or Boolean types and also expresses predicates.

Other basic types are constructed based on these notions. The suffix notion set denotes a type of collections of basic types, for instance, an $S\ set$ is a type of collections of possible spatial locations. The basic type of regions R is defined by an $S\ set$ because regions are bound by polygons, curves and/or collections of isolated locations. Likewise, a time interval I is defined by a $T\ set$. Some basic types are constructed for attributing referents: The type $Extent$ is defined by a tuple of a region and time interval ($R \times I$) and describes the extent of something. An $SExtent$ denotes a spatial extent. It is a special form of an $Extent$ without a time reference. $Occurs$ is defined by a set of spatio-temporal points ($S \times T\ set$) and conceptualizes occurrences of something, i.e. a footprint or support.

Basic types and their terminology are described extensively in this section because most of them will be used throughout the thesis for semantic modeling.

Generation functions

Generation functions are described by data *generation types*. Those types are sub-divided into *sets, selections, and partitions* on the one hand, and into *spatio- temporal generation types* on the other hand. The former group consists of are $Select$, $Tessel$, $Partition$ and $Qstat$. A $Select$, defined by $Select \Rightarrow S \times T$, selects a centroid or something alike from an extent. Similarly, the derivations $SSelect :: R \Rightarrow S$ and $TSelect :: I \Rightarrow T$ select centroids from regions and time intervals, respectively. The function $Tessel :: S \times T \Rightarrow Extent$ maps spatio-temporal locations to an Extent. It is also possible to map locations to regions and moments in time to Regions, which is done by $STessel$ and $TTessel$ respectively. The function $QPartition :: Q \Rightarrow Qset$ maps quality values to quality values and the function $Qstat$ summarizes quality values. $Qstat$ conceptualizes many statistical operations like for instance calculating the mean, median or variance of numeric values.

Spatio-temporal generation types are *field*, *inverted field*, *lattice*, *event*, and *object*. Fields create quality values in space and time where each tuple $S \times T$ maps to an observation Q, short $S \times T \Rightarrow Q$. Examples of fields are taking sensor observation in space and time and the interpolation of those values. While fixing space results in a temporal field ($TField$), from which time series data can be generated, a spatial field ($SField$) maps quality values from locations in space as time is fixed ($S \Rightarrow Q$). Lattices ($R \Rightarrow I \Rightarrow Q$) refer to data aggregations, either in Space and Time or with either of them fixed. They conceptualize for instance the generation of daily average temperatures over a region.

The general definition of an inverted field ($invField$) is $Q \Rightarrow Occurs$. An event is defined by $D \Rightarrow S \times Q$ and an object by $D \Rightarrow T \times S$. Some of these generation types are referred to in the remaining chapters, where they are further explained within the context.

Derivation functions, data generators and primitive operations

Derivation functions and data generators are only a few times referred to in this text, and if so they are explained explicitly. For instance, $curry :: ('a \times 'b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b \Rightarrow 'c)$ is a primitive operation used for converting an unary function with a tuples of n referents input into an n-ary function where each referent is an input. Note that $'a$, $'b$ and $'c$ are variables for arbitrary reference types. The operation $settop :: ('a\ set) \Rightarrow ('a \Rightarrow bool)$ converts a set of referents into a predicate.

Spatio-temporal derivation graphs

Figure 3.2 is an example of a derivation graph that was published by Scheider et al. (2016). It shows how a temporal field ($TField$) can be aggregated from a $Field$. A $TField$ is the conceptual foundation for generating time series data. Diamond-shaped nodes refer to functions based on operations from the algebra, i.e. derivation functions and data generators. Generation functions and referents are specified by elliptic nodes. Furthermore, nodes are colored white if they act as an input or output of a function and they are orange for those functions that are *called*. Arrows from inputs to a called

functions have empty arrowheads, while arrows from function to output have filled arrowheads.

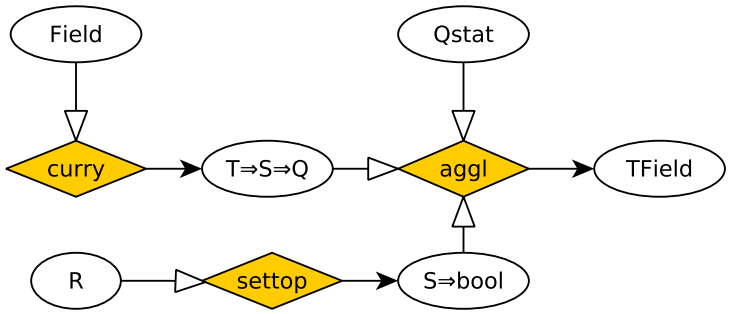


FIGURE 3.2: A derivation graph for generating a time series from a Field (Scheider et al., 2016, p. 1994). Base types are R, T, S, and Q. Derivation functions from the algebra are denoted by diamond-shaped nodes. The node labeled with R denotes a referent, all remaining nodes refer to generation functions.

3.5 Applied concepts of provenance

The main approach to capture provenance information is to record and analyze commands after they are executed. At the same time changes in the user's workspace are monitored, i.e. changes in the environment in which in- and outputs of the executed analysis are stored. After enabling provenance tracking, this process runs in the background without any required user-interaction.

The recorded commands often comprise nestled and concatenated function calls and primitive operations. For displaying them in a derivation graph, the commands are broken down to individual function calls and their in- and outputs, which each map to one node. For improved readability and in order to keep the graph simple, expressions that consist of only primitive functions and operations not broken down into individual parts, but treated as one call. For instance, the expression $4 * sin(x) + 2 < 5$ is kept together as one *call node*, because it employs only simple mathematical and logical operations.

The following paragraph describes how this work addresses challenges that arise when deriving provenance-information from either scripted source code or series of commands that are interactively executed from command-line interfaces:

3.5.1 Mapping unique identifiers to variable names

Most programming languages refer to objects or values by variable names (name binding). During program execution, the object that is bound to the variable may be modified and the initial binding of the variable might be overwritten with bindings to other objects. Therefore referring to actual

data within a provenance record by just a variable name is ambiguous, because this name refers to different objects of different states and properties, depending on the stage of execution.

It would be possible to assign randomly generated identifiers to an object and refer to data by these id's instead of the variable name. This solution is insufficient, however, because it does not take into account modifications of the same object. Also, it suggests itself to keep a reference to the variable name, because on the one hand it facilitates relating source code to derivation graphs, on the other hand, variable names are usually assigned with a specific intention of the developer and therefore communicate an implicit meaning of the data they refer to.

This work proposes to map *instance identifiers (IIDs)* to variable names, which persistently refer to the data bound by the variable name within one stage during the execution. This stage of execution is defined by the time period between two modifications of a variable binding. An IID consist of a variable name and an incremental version number. For instance, a variable *myVar* maps to the IIDs $myVar \sim 1$, $myVar \sim 2$, $myVar \sim 3$ et cetera based on the name binding and its modifications. IIDs are used in the derivation graph in order to unequivocally refer to data and their (semantic) properties during the execution of an analysis. Changes of the variable name bindings in respect of different of stages of program execution are summarized in a *variable version record*.

3.5.2 Detecting side-effects of function calls

This work retraces executions of data analyses by causal graphs which are directed, hierarchic and acyclic. Arrows connect input data to calls (function calls and primitive expressions) and these calls to their output. On the one hand, mapping function calls to these causal graphs suggests itself to be straightforward because function inputs are defined in the function signature or argument list and because functions return either one or no output by end of execution. On the other hand, side-effects of function calls pose a challenge:

Many programming languages permit functions to make use of external objects and also to call other functions that neither belong to the function's internals nor to the declared inputs. Functions may also create or modify new objects in an external environment. In R, side-effects also comprise printing messages/warnings and plotting/visualizing data during the execution of a function.

This thesis proposes to detect all those side-effects that are useful for comprehension of an analyses and to incorporate them as *hidden in- and outputs* in derivation graphs. This proposal is constricted because detecting all side-effects is not always useful, for instance, calls to internal functions which are intentionally hidden from the users. It is rather proposed to only detects side-effects that concern the the user's dedicated workspace.

Detecting all possible side-effects in a language like R by far exceeds the scope of this thesis, but work shows practical solutions to detect some of them:

- It can be heuristically detected whether a function call makes use of external variables or functions, which is done by analyzing the function body.
- It can be detected whether a function call creates new data in the workspace or overwrites existing variables with bindings to newly created objects, which is done by monitoring changes in the workspace.

3.5.3 Syntax add-ons for derivation graphs

The derivation graphs modeled in this thesis comply to the above-shown syntax by Scheider et al. (2016), but they are extended by additional syntactical elements in order to express provenance-related patterns of data analysis that are difficult or impossible to express otherwise (see section 4.2.3), for instance, side-effects of function calls (see previous section 3.5.2).

It can be argued that these modified graphs are downward-compatible and hence can be reduced to the basic syntax defined by Scheider et al. (2016), although this would implicate a loss of information. Further research can thus adopt the herein solutions of this thesis without necessarily incorporating the syntax add-ons. On the other hand, if they are found useful, the syntax add-ons suggested by this thesis could also propagate to further works.

3.6 Applied concepts of semantics

Semantic metadata is applied to the data derivation graph in a semi-automated way. The approach combines automation based on inference/reasoning and heuristics with user-defined annotations and validity checks.

3.6.1 Estimation of semantic types using heuristics

An analysis that does not incorporate semantic metadata still could be assessed in terms of semantics. This assessment comprises heuristic assumptions on the semantics that a resource or a call *could have* and makes explicit which semantic information is *missing*.

In order to achieve this, a heuristic mapping of semantic types is applied (see appendix A). If a data-set is not explicitly annotated, the semantic type of a dataset is estimated from its data type and other properties that can be queried (see appendix A). In cases where the mapping does not allow an assessment of meaningfulness because it is ambiguous or incomplete, the estimated semantic type is prefixed with a bracketed question mark $(?)$ and a warning is prompted to the user that suggests to manually annotate the data. This approach is consistent with the argumentation of Scheider et al. (2016, p. 1981), where the authors expressed that data analysis crucially depends on meta-information that goes beyond the data type.

3.6.2 Posterior signature estimation

Dynamically typed programming languages like R, Haskell or MATLAB do not formally define function signatures in terms of their data types. This enables a very flexible and dynamic mode of programming because functions can accept a variety of different input combinations.

Yet for the purpose of modeling and validating function semantics, it is important to determine the semantic types of all in- and outputs. There are two ways to achieve this: The first way is to force a pre-defined signatures on the function. The function itself would throw an error during a call and interrupt the execution if the inputs do not match certain semantic types. This approach is disadvantageous because it undermines the dynamic-typing paradigm of these languages and results in a loss of flexibility in exchange for semantic control.

This thesis suggests an alternative approach, which is less oppressive. It proposes to estimate the (implicit) function signature during runtime, i.e. when the function is called with real inputs. An advantage of this approach is that it favors posterior validation: Instead of enforcing semantic structures on the analysis and programming style, it leaves these structures unchanged and provides semantic provenance, including warnings, for posterior review and validation.

Within a similar context, Miles et al. (2007) suggests doing provenance-based validation of e-science experiments on the web, i.e. using a service oriented approach. In this approach, validation is also not considered *during execution*, but rather afterward. This posterior validation based on provenance data and a separately defined set of requirements and semantics. The provenance record can be shared with third-parties that can perform validation by themselves using different sets of requirements without the necessity to re-execute the workflow.

This thesis suggests two technical approaches in order to estimate function signatures during runtime:

- The first approach is to define wrappers around the functions that act as a proxy and analyze the input arguments and function output before the latter is returned. The wrapper would ideally have the same signature and name as the function it is dedicated to, so that it can be applied by existing code without much editing.

- The second approach is to add task handlers that are invoked before and/or after a command (or task) was executed. These handlers could also analyze the function call along with inputs and outputs that are available in the user's workspace. (The R prototype only uses posterior analysis after execution of a command)

Both of these approaches have been implemented as complementary mechanisms in the 'SpatialSemantics'-package for R (see chapter 4).

3.6.3 User-defined semantics and validity checks

While the above-mentioned approaches could be fully automated, this section addresses the insertion and customization semantic metadata by users and developers, which cannot be fully automated.

Semantics need to be customizable because they reflect the researcher's assumptions and viewpoints on data and analysis. In this work for instance, datasets that contain just georeferenced polygons are automatically mapped to the semantic type $R\,set$ and denote a set of regions. A user may want to use data in order to describe the area in which a research team localized colonies of a certain animal species. In such a case, the researcher would customize the semantics of this region as a spatial extent ($SExtent$), because it represents the observation of a temporal point pattern, i.e. observations of animal colonies. Yet the type $SExtent$ does not universally apply to the data because the forest area might as well referred to in other contexts.

In addition, spatio-temporal data is often not provided together with metadata about their semantics and provenance. The output of many functions used in data analysis is neither annotated. Therefore is beneficial if user-researchers who conduct a data analysis could insert this missing information in a simple and concise way. It suggests itself to provide semantic annotation capabilities in the environment he/she is normally using, which is software for Spatial Statistics.

For these reasons, this thesis proposes to facilitate user-defined semantics and validity checks as the following:

Annotation of datasets with user-defined semantics

It is proposed to implement functional components that allow the user to annotate data with their semantic type and with data generation functions involved in their creation. These assertions are then incorporated in the semantic provenance, notably in the spatio-temporal data derivation graph. It is further proposed to simplify this process by letting the user specify the name of the data generation function (for instance Field, Object, etc.), from which the semantic type of the data and the formal definition of the function is inferred.

Because spatio-temporal data can comprise many different attributes denoting subsets of observations, and because these subsets may all have individual semantics, it should be possible to specify semantics also per attribute rather than per dataset.

Customizable function wrappers

While the above-proposed function wrappers are used to estimate a function's implicit signature, herein they are proposed for additional purposes:

- In order to match the semantics of a function with the users assumptions and *expectations*, it is proposed to let him/her optionally specify the *expected call semantics* of a function call. If the call does not match these default semantics, a warning is prompted and the graph is marked by the keyword *INCONSISTENT*.

- In order to let the users and developers implement optional add-hoc checks for consistency or meaningfulness, it is proposed to let them define a validator, which is a Boolean function that has the functions inputs, outputs, call semantics and the expected call semantics (if any)

as arguments. It is invoked by the function wrapper after each function call. It may throw errors or warnings during the execution based on custom requirements. If the validator returns *FALSE*, a default warning is prompted and the graph is marked by the keyword *INVALID*.

- In order to annotate the output of a function, it is proposed to let the user/developer specify a post-processor to a function that is invoked after by the function wrapper after each function call. The purpose of this post-processor is to add predefined annotations to the function's output. In order to simplify this process, a default-postprocessor may be used that annotates the runtime semantics of the function as a semantic procedure associated with output's semantic pedigree.

3.7 Software approach: R and Graphviz

The R language and environment

The 'SpatialSemantics'-package introduced in chapter 4 is implemented in the R language and environment for graphics and statistical computing (version 3.3.1) (R Core Team, 2016)[1]. R was chosen because it has established itself as a widely used tool across many scientific communities. It has a large community of users and developers that gather on dedicated conferences like the *UseR!*. Users and developers frequently communicate over subject-specific mailing lists.[2,3] Notably the *R-SIG-geo* addresses a *Special Interest Group on using Geographical data and Mapping*.

The R environment is completely open source so that anyone can review the source code of any of its components. The language itself is well-documented (R Core Team, 2000; Wickham, 2014) and anybody can write extensions in the form of R packages. The book *'Extending R'* (Chambers, 2016) describes extensively how this can be done. Analyses carried out in R are generally reproducible by the R script, although it requires sharing also the input data. If an analysis is transferred from one software installation to another it also has to be ensured that the used R version and all library versions are compatible.

Many works and publications address specifically Spatial Statistics in R, for instance, Baddeley, Rubak, and Turner (2015), Hengl (2009), and E. J. Pebesma (2004). The sp-package (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013) comprises many classes and functions for spatial data, providing an infrastructure for handling spatial polygons, grids, points and lines that is re-used by many other packages and which is also incorporated in the use-cases of this thesis 5. Thanks to the R-community and their achievements, it was possible to base the use-cases in directly on published research in Spatial Statistics that was carried out with R.

[1] R software can be obtained online on https://www.r-project.org/ (Last access on Sept. 14, 2016).

[2] An analysis of the R-help mailing list by David Smith (2009) is online available at http://blog.revolutionanalytics.com/2012/08/an-analysis-of-the-r-help-mailing-list.html (Last access on Sept. 13, 2016).

[3] A listing of official R mailing lists is available at https://www.r-project.org/mail.html (Last accessed on Sept. 11th, 2016)

Recent work on the subject of *meaningful Spatial Statistics* was also prototyped in R (C. Stasch, Scheider, et al., 2014) [4] and resulted in the 'mss' package [5].

R packages and components for capturing provenance

Some specific components and additional packages were incorporated in the software prototype (Chapter 4) for the purpose of capturing provenance. R task handlers (R Core Team, 2016, p. 550-554) [6] are incorporated because they allow defining custom callback functions which are invoked after each top-level task (command) entered into the R command line. Once the callback is activated, the callback is executed invisibly in the background while the analysis is carried out. This mechanism is crucial to the provenance recording of the prototype. One disadvantage of these task handlers is that they currently do not work in batch mode or when executing scripts with the demo or source - functions. This poses considered only a minor issue to the implementation, as it does not encumber the purpose of this thesis.

The implementation also makes use of the R packages codetools (version 0.2-14), CodeDepends (version 0.4-2), stringr (version 1.1.0), graph (version 1.50) and Rgraphviz (version 2.16.0).

The package CodeDepends (Lang, Peng, and Nolan, n.d.) provides tools for code analysis and is used to determine inputs and outputs of a particular command. The codetools package (Tierney, 2015) also provides code analysis tools and was specifically used to detect global variables and functions referred to in function bodies. The stringr package (Wickham, 2016) is used for string manipulation because it provides simple and consistent wrappers around common string operations.

The graph package (Gentleman et al., 2016) provides data structures to handle different types of graphs in R. Rgraphviz (Hansen et al., 2016) bridges between R and Graphviz and builds upon the graph package. It provides the 'Ragraph' format, which extends the existing graph data structures with graphical attributes and plotting capabilities. Graphs can be exported in various data formats, notably the 'dot/gv' format utilized by Graphviz.

Graphviz for graph visualization and export

Graphviz (Gansner and North, 2000)[7] is an open-source software for visualising graphs. The visualization of pedigree with R and GraphViz has been formerly demonstrated by Zhao (2006), associated with research on genetics and population analysis. Graphviz makes use different algorithms for

[4] Tools on 'Meaningful Spatial Statistics' are referred to on http://meaningfulspatialstatistics.org/tools/ (Last access on Sept. 11, 2016).

[5] The 'mss' package is available on GitHub: https://github.com/edzer/mss (Last access on Sept. 11, 2016).

[6] An exhaustive description of the task callback mechanism is online available in 'Top-level Task Callbacks in R' by Duncan Temple Lang (2001): https://developer.r-project.org/TaskHandlers.pdf (Last access on Sept. 11, 2016).

[7] Resouces and throughout documentations of Graphviz are available online at http://www.graphviz.org/ (Last access on Sept. 14, 2016).

graph layouts. The visualization is customizable by a large set of graphical attributes. The 'dot/gv' used to serialize Graphviz graphs can be converted into various common graphics formats like pdf, png, svg and jpeg. Graphviz can be used with many programming language bindings and various graphical user interfaces. [8]

[8] I recommend using the XDot viewer for Ubuntu for basic interactive visualization, zooming and panning of dot-graphs. Instructions are online at https://apps.ubuntu.com/cat/applications/quantal/xdot/ (Last access on Sept. 15, 2016).

Chapter 4

The 'SpatialSemantics'-package for R

'SpatialSemantics' is a prototypical R package for capturing the semantics and provenance of spatio-temporal data analyses. It extends R towards a provenance-enabled, semantic-aware system for Spatial Statistics and therefore implements the concepts introduced in the sections 3.5 and 3.6 of the methodology in chapter 3.

The package records provenance through task-callbacks as commands are executed on the R-console. Part of this provenance information is a version history maintained for each variable that serves to answer important provenance-related questions, namely where, when and how an object has been created or modified. Based on the collected information a spatio-temporal data derivation graph is constructed that describes all objects, operations, calls and their parameters involved in the execution.

The graph is enriched with semantic annotations in compliance with Scheider et al. (2016) (see section 3.4). These annotations enable an interpretation of data in respect of their meaning and allow an assessment of spatio-temporal analyses in terms of meaningfulness and semantic consistency. The basic graph-syntax introduced by Scheider et al. was extended in order to adequately represent provenance information (see section 3.5.3).

Spatio-temporal data derivation graphs can be visualized by the Rgraphviz-package. Using Graphviz-capabilities, it is possible to export graphs in many data formats, notably dot/gv, pdf, svg, png and jpeg (see section 3.7.

This chapter gives a comprehensive overview of the package and how it is used. It shows how the graph-syntax was extended in respect of Scheider et al. (2016) and elaborates the technical realization of the prototype.

4.1 Package installation

The package is included in the attached CD. [1]

The package can be installed from online sources by using the following code in R code.

LISTING 4.1: Online installation of the 'SpatialSemantics' package

```
1 # install  dependencies not hosted on CRAN
2 install .packages("devtools")
```

[1] The package is also hosted online, where it is actively developed at the moment: https://github.com/MatthiasHinz/SpatialSemantics (Last access on Sept. 15th, 2016).

```
devtools:: install _github('duncantl/CodeDepends')
source("https://bioconductor.org/biocLite.R")
biocLite("Rgraphviz")
#optional command, these dependencies should install automatically from
    CRAN:
 install .packages(c("stringr","codetools")
# install 'SpatialSemantics' from GitHub
devtools:: install _github("MatthiasHinz/SpatialSemantics")
```

Alternatively, the packages and dependencies can be obtained installed offline using the install.packages function according to the below following schema. The devtools-package (Wickham and Chang, 2016) is not needed in this case.

```
    install .packages(<path/package.zip>,repos=NULL,type="source")
```

For reproducing the code examples in this chapter, it is also necessary to install the sp-package (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013).

```
    install .packages("sp")
```

4.2 Package overview

The 'SpatialSemantics'-package currently exposes 15 functions as an interface to the user. These functions can be categorized into functions for provenance, utility functions and functions for semantics. They are explained in the following sub-sections together with the related functionality.

4.2.1 Provenance functionality and utilities

Figure 4.1 illustrates the basic mechanism underlying the provenance recording functionality and semantic inference in a typical sequence. It is based on task callbacks (R Core Team, 2016, p. 550-554), which are invoked during the read-eval-print loop that forms the basic command line interface of R. Technical details about the loop are explained in the R language definition (R Core Team, 2000, p. 9, p. 45): When a user types in a command, the text input is read until a complete expression is available. This expression is internally parsed into a *call*. This call is then evaluated. During this evaluation process, R typically reads and manipulates data in the user's workspace (workspace:environment), which comprises the global environment and all subordinated environments (R Core Team, 2000, p. 20) [2]. The result is an evaluation value that is printed to the console (if it is not *invisible*). Before the task is completed and the parser awaits subsequent expressions, a task call back is invoked. The callback (callback:SpatialSemantics) is a previously registered function defined in the 'SpatialSemantics' package. The function analyzes the previously evaluated expression and also keeps track of changes in the user's workspace. It actively annotates data in the workspace with provenance- and semantic-related attributes and stores metadata in the metadata-store, which is a separate environment within

[2] R stores data and functions locally in hierarchies of distinct environments. Examples are the global environment (which is the root of the user's workspace), package-environments and the internal environment that a function encloses

the internals of the 'SpatialSemantics' package. Keeping metadata in a separate environment has the advantage that they are kept hidden in the background and persistently stored during the session unless the provenance-record is purposely deleted. Later on, the metadata can be retrieved and visualized as a data derivation graph.

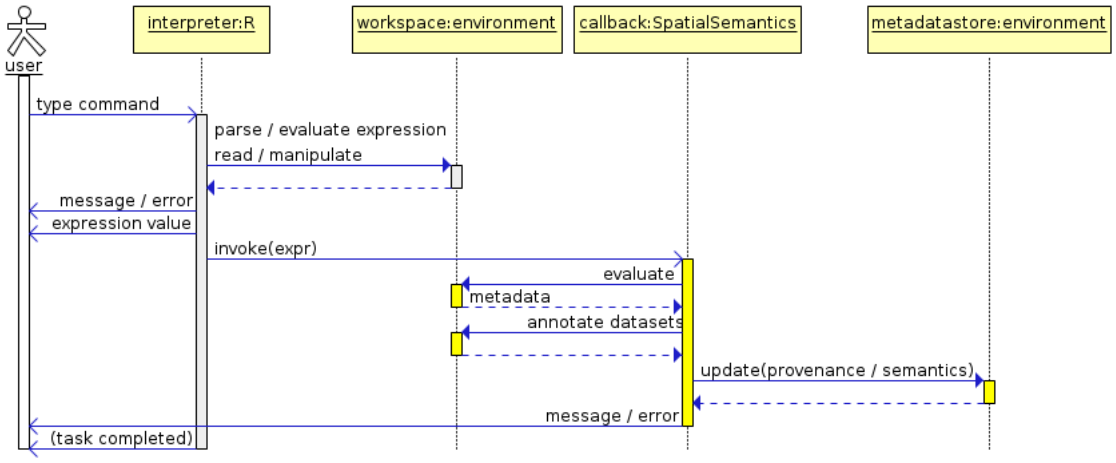


FIGURE 4.1: Sequence of provenance recording and semantic inference

Table 4.1 lists all functions useful for recording, querying and retrieving provenance as well as some utility functions. They will be explained in the following paragraphs.

TABLE 4.1: Functions for provenance and utilities defined by the 'SpatialSemantics'-package

Function Name	**Description**
enableProvenance disableProvenance	Enable or disable provenance tracking
getScriptGraph	Retrieve derivation graph for export and visualization
getVersions	Get history of a variable's name bindings (IID mapping)
provenance_history	List of all recorded commands
reset_provenance	Deletes all internally recorded provenance and sets the tracker back to default state
rewriteReplacementFunction	Utility function that converts calls to replacement functions
spsem_internals	Utility function for retrieving package internals

Provenance recording

Listing 4.2 shows a simple example of how to record provenance with 'SpatialSemantics'.The code retrieves a spatial dataset named *meuse* from the sp-package (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013) (line 3). It

then sets up the data as a spatial data set by defining the x and y columns of the given data frame as coordinates (line 4). Finally, a new attribute/-column named $lzinc$ to the data by computing natural logarithms from the existing numeric attribute $zinc$ (line 4). The function *enableProvenance* (line 2) registers the callback function that executes provenance recording and initializes the provenance-record, if necessary. The function disableProvenance (line 6) simply removes the callback from the register of callbacks. Hence, all commands between the calls *enableProvenance()* and *disableProvenance()* (lines 2 and 6) are recorded. Commands before or after this do not become part of the provenance record. This has the advantage that users can restrict the recording to those parts of the executions that form part of the analysis. As shown here, it is recommended to pre-load libraries (line 1) or define new functions from outside the recording-block. Also, the retrieval of provenance and semantics is normally not of interest (line 7).

In line 7, the recorded provenance graph is retrieved by a call to *getScriptGraph()*. The graph is directly passed to the *toFile* function, which is defined by the Rgraphviz package. *toFile* exports the graph as a dot-file that can be externally visualized or converted int other formats using Graphviz. The output of *getScriptGraph()* can also be directly visualized using the plot-function of R, but some details of the graphs cannot be displayed, including the semantic annotations. This derivation graph associated with Listing 4.2 is explained in section 4.2.3 of this chapter (Figure 4.3).

LISTING 4.2: Simple example of provenance recording

```
1 library(SpatialSemantics);library(Rgraphviz);library(sp)
2 enableProvenance() #recording starts here
3     data("meuse")
4     coordinates(meuse) <- c("x","y")
5     meuse$lzinc = log(meuse$zinc)
6 disableProvenance() #recording ends here
7 toFile(getScriptGraph() , layoutType="dot", filename="myDerivationGraph.dot"
      , fileType="dot")
```

Query provenance

After recording provenance, the user may query certain provenance information of the analysis. The history of all recorded commands can be retrieved by the function *provenance_history* (Listing 4.3).

LISTING 4.3: provenance_history: List all recorded commands

```
1 > provenance_history()
2  [[1]]
3 data("meuse")
4
5  [[2]]
6 coordinates(meuse) <- c("x", "y")
7
8  [[3]]
9 meuse$lzinc = log(meuse$zinc)
```

A user may also want to query information about certain variables. In the above-shown example, the variable *meuse* has been modified three

times. Changes and modifications of a variable, precisely on its name binding, can be revealed by calling the *getVersions* function with the variable name as an argument (Listing 4.4). The displayed information can be read like a change- or version history of the variable. It contains information about the 'class' and 'semantics' of the data that was associated with the variable during one stage of execution. All provenance records refer to these data by instance identifiers (IID) (See section 3.5.1), which are newly created for each change of the name binding. The listing shows that the variable *meuse* changed its class as well as semantics during the execution. It also shows the commands by which it was changed.

Furthermore, the change history attributes to each row a timestamp, showing the exact time when the binding was changed and it attributes a record number (rec_num), which relates the change to the history of recorded commands. The latter two attributes are not displayed in the listing in order to keep the overview simple.

LISTING 4.4: getVersions: Display changes of name bindings

```
 #[c(−1,−6)] conceals the  attributes  'rec_num' and 'timestamp':
> getVersions(meuse)[c(−1,−6)]
      IID                    class      semantics                              command
1    meuse                data.frame          Q set                        data("meuse")
2  meuse~2 SpatialPointsDataFrame (?)S x Q set coordinates(meuse) <− c("x", "y")
3  meuse~3 SpatialPointsDataFrame (?)S x Q set      meuse$lzinc = log(meuse$zinc)
```

provenance information is added to the existing metadata when enableProvenance() again. If a user wants to clear all recorded provenance data in order to start recording a new analysis he can do so by calling the *reset_provenance* function:

```
reset_provenance()
```

It was decided to expose two utility functions to the users, which can be handy sometimes. However they are not mandatory to the package usage: *spsem_internals* can be used to inspect the packages internal environment, in which metadata and internal functions are stored. The function *rewriteReplacementFunction* is used by the parser internally. It is used to transform calls to replacement functions into an equivalent, more parser-friendly syntax, for instance:

```
> rewriteReplacementFunction(quote(coordinates(meuse) <− c("x", "y")))
meuse <− ‘coordinates<−‘(meuse, c("x", "y"))
```

Detecting side-effects of function calls

As the previous section 3.5.2 in the methodology chapter states, side-effects of function calls can pose a challenge to obtaining provenance-information. Nevertheless, the provenance recording mechanism incorporates two approaches to detect some of these side-effects.

(1) The first approach makes use of the codetools package (Tierney, 2015). The package defines a function called findGlobals that inspects a function's definition and finds global functions and variables invoked or used in the methods body. Global and functions and variables in this context are those

that do not belong to the environments enclosed by the function, i.e. those objects that are not local, but external. The callback function that is utilized for provenance recording breaks expressions (the user input) down into distinct function calls. It then retrieves the definitions of the called functions (if available, which is not the case for primitive functions) and then determines the globals using *findGlobals*. These globals are then incorporated as *hidden* inputs and function calls in the provenance record.

(2) The second approach relies on monitoring changes in the workspace: the metadata-store of the package keeps a temporary list of all variables stored in the workspace (as returned by the R function call $ls()$). When the callback is invoked, it compares the *previous* workspace content with the *current* and determines those variables which are newly created. As another means, the callback annotates each object when it is observed for the first time with an attribute called *'isTracked'*. Among those variables which were previously apparent in the workspace, it can be detected that they have been overwritten or modified if the *'isTracked'*-attribute is missing. Finally, those modifications that were detected by workspace monitoring are compared with those modifications that can be inferred directly form the expression which was previously evaluated. Those modifications which do not become apparent from the expression are considered side-effects. They are Incorporated as *hidden* outputs in the provenance record.

It is important to note that both approaches are estimations. They are not capable of detecting all side-effects but rather some of them. The second mechanism is also imprecise insofar that the origin of a side-effect can be narrowed-down to the top-level expression, but it cannot determine the associated function call if the expression comprises multiple nestled calls. Yet it is shown in this thesis that the derivation graphs are more precise thanks to this functionality.

4.2.2 Semantic functionality

There are six functions exposed to the users which facilitate semantic functionality. Part of the semantic functionality works without any user-interaction. Even without explicit metadata available, semantic types can be propagated using heuristics and inference (see sections 3.6.1 and 3.6.2). Section 3.6.3 explained that semantics, however, needs to be customizable. This is not only because crucial semantic information is often missing, but also because semantic metadata reflect on the researcher's individual assumption of the data and analysis. Therefore, the functions *captureSemantics<-*, *functionalType<-* and *addSemanticPedigree* facilitate user-defined semantics and validity checks, as explained in section 3.6.3. The remaining three functions are used to query for existing semantics.

Inspecting and defining object semantics

The semantics of an object can be inspected by using either *getSemanticObject*, which returns a semantic type denoting *what* the data represents, or the function *getSemanticPedigree*, which returns the semantic pedigree of an object, an ordered list of all semantic procedures directly involved in the creation of a dataset. The record also shows if only part of the data was modified, by optionally referencing the attribute. Users can also retrace how

TABLE 4.2: Functions for semantics defined by the 'SpatialSemantics'-package

Function Name	**Description**
addSemanticPedigree	Add semantic pedigree to data
captureSemantics<-	Create semantic function wrapper
functionalType<-	Add a functional type to the semantic pedigree of data
getDefaultCallSemantics	Pre-defined semantics of a function call
getObjectSemantics	Estimate semantics of a given object
getSemanticPedigree	Get semantic pedigree

the semantics were modified because the records include the corresponding R command (see listing 4.10, explained below). The latter information is not available if provenance recording was not activated while applying semantics. Apart from this constriction, however, data can be annotated both inside and outside of recording block.

In code listing 4.2 which was explained in the previous section, none of the functions for semantics were incorporated yet, but the associated listing 4.4 showed that the variable *meuse* is already annotated with semantics. The output derivation graph (Figure 4.3) also shows that almost all nodes already contain semantic annotations. However, during the execution of the example, a warning appears after line 4, which is the following:

'No semantic annotation available for the object meuse. Assumend semantics will be (?)S x Q set'

Code Listing 4.5 shows how the (current) object semantics of *meuse* can be retrieved. The semantic type $S \times Q\ set$ is prefixed by a question-mark, which means that the type is estimated based on heuristics and that crucial meta-information about this object is missing.

LISTING 4.5: Get object semantics

```
1 > getObjectSemantics(meuse)
2 [1] "(?)S x Q set"
```

It is possible to replace this estimation by explicitly annotating the data with an attribute of name *'semantics'* (Code listing 4.6). This will overwrite any previously assigned semantic types too (which still can be reviewed using getVersions and/or getSemanticPedigree). However, it is not recommended to set semantics of complex spatio-temporal datasets in this way, because information about underlying spatio-temporal would still be missing.

LISTING 4.6: Setting semantic types by attribute

```
1 > attr(meuse,"semantics") <- "'a set"
2 > getObjectSemantics(meuse)
3 [1] "'a set"
```

Instead, it is advised to use either use the replacement-function *functionalType* (listing 4.8) or the function *addSemanticPedigree* (listing 4.7 in order to annotate datasets in respect of their generation procedures.

Code listing 4.7 shows how to add the semantics of a log-transformation to the semantic pedigree of a dataset. Often this is not necessary, because semantics are inferred manually, at least for simple expressions such as a log-transformation. The function *addSemanticPedigree* is made for defining pedigree in a custom and flexible way: Users can define the name of the generation procedure (name) and the signature of this procedure (procedure). The function argument result_semantics describes the output of the procedure in conjunction with a data generator. Often, a semantic procedure just modifies an attribute of the dataset instead of the whole dataset, e.g. the zinc-attribute of the meuse dataset, which is a Q set. In such cases, two semantic attributes have to be specified, which are result_semantics and parent_semantics. The former describes the targeted attribute only, while the former describes the complete dataset which is modified. The user can optionally specify the modified attribute (attr-argument). If it is not specified, it will be assumed that the procedure modifies all attributes of the dataset.

LISTING 4.7: Setting semantic pedigree manually

```
meuse = addSemanticPedigree(obj = meuse,name = "log", attr = "lzinc",
    procedure = "S x Q set -> Q set", result_semantics = "Q set", parent_
    semantics = "S x Q set")
```

The function *functionalType* is dedicated to specify spatio-temporal generation types: The notion of functional types traces back to C. Stasch, Scheider, et al. (2014) and E. J. Pebesma and C. Stasch (2014), who described spatial- and spatio-temporal data in terms of functions that expresses their meaning. Within the context of this work, a functional type is defined as the type of the spatio-temporal generation function which causes one dataset to be in its current semantic representation. This work refers to the spatio-temporal generation type definitions by Scheider et al. (2016), which slightly defer from functional types as described in the above mentioned sources.

If functional types are assigned, they become part of the semantic pedigree of the data. The functionalType-function applies a straight-forward mapping from the type name and definition to the arguments of addSemanticPedigree (i.e. result semantics and the optional parent dataset semantics), which is described in B). The mapping will be further discussed in chapter 7. Listing 4.8 shows how to set functional types either per attribute or for a complete dataset.

LISTING 4.8: Set functional type

```
> #setting the functional type for certain attributes
> functionalType(meuse, attr = "zinc") <- "SField"
> functionalType(meuse, "zinc")
[1] "SField"
> functionalType(meuse)
NULL
> #setting the functional type for all data
> functionalType(meuse) <- "SField"
> functionalType(meuse, "zinc")
[1] "SField"
> functionalType(meuse)
[1] "SField"
```

In order to add missing semantic to in code-listing 4.2, it suffices to specify the spatio-temporal generation procedure with *functionalType*. Code listing 4.9 is a modified version of listing 4.2, where user-defined semantics are applied. As mentioned above, a warning is printed after line 4. Therefore the required semantics are supplied by the command in line 5.

LISTING 4.9: Minimal example of provenance recording with semantics

```
1 library(SpatialSemantics);library(Rgraphviz);library(sp)
2 enableProvenance() #recording starts here
3     data("meuse")
4     coordinates(meuse) <- c("x","y") ## warning occurs here
5     functionalType(meuse) <- "SField" ## add functional type here
6     meuse$lzinc = log(meuse$zinc)
7 disableProvenance() #recording ends here
8 toFile(getScriptGraph() , layoutType="dot", filename="myDerivationGraph.dot"
      , fileType="dot")
```

The semantic pedigree of a dataset can then be explored by *getSemanticPedigree*. Listing 4.10 demonstrates this with respect of listing 4.9 on *meuse*. It can be retraced that the unclear semantics of the initial dataset (result-semantics-attribute in the first row) were re-redefined the functionalType-function (command-attribute in the second row). A procedure called 'log' (third row) then added or modified the attribute 'lzinc' of the overal dataset.

LISTING 4.10: Inspect semantic pedigree

```
1  > getSemanticPedigree(meuse)[c("procedureName","procedure","result_attribute"
       )]
2    procedureName                          procedure result_attribute
3  1 coordinates<- Q set -> Q set -> (?)S x Q set               ALL
4  2          SField                              S -> Q         ALL
5  3             log                 S x Q set -> Q set        lzinc
6
7  > getSemanticPedigree(meuse)[c("result_semantics","parent_semantics")]
8    result_semantics parent_semantics
9  1      (?)S x Q set            <NA>
10 2             Q set      S x Q set
11 3             Q set      S x Q set
12
13 > getSemanticPedigree(meuse)[c("rec_num","command")]
14   rec_num                              command
15 1       2 coordinates(meuse) <- c("x", "y")
16 2       3 functionalType(meuse) <- "SField"
17 3       4      meuse$lzinc = log(meuse$zinc)
```

Defining semantic function wrappers

The following paragraph explains how to define and semantic function wrappers. Function wrappers in this work serve multiple purposes. They are (1) posterior estimation of implicit function signatures (see section 3.6.2) (2) to apply user defined consistency- and validity checks, and (3) to annotate function outputs with pre-defined semantics (see section 3.6.3 for user-defined semantics and validity checks).

Figure 4.2 illustrates a call to a function wrapper by a sequence diagram: The semantic wrapper (wrapper:function) acts as a proxy to the user. It is called by the same name and arguments as the wrapped function (main:function). The proxy passes function all inputs directly to the *main*-function, which processes the data and computes the output. The wrapper then estimates the implicit function signature by inspecting the semantics of all inputs and outputs using the *getObjectSemantics*-function. Then all data, including the estimated semantics, are passed to the postprocessor (postprocessor:function) this postprocessor is either user-defined or a standard generic function that annotates the output with semantic pedigree. The semantics are then re-evaluated by the wrapper and matched against pre-defined default semantics, if any. If the de facto semantics of the call are conflicting with any of these defaults, a warning is prompted, because the call semantics are *inconsistent*. All data and semantics are then passed to a validator, which is an optional component. Validity of the operation is then checked according to user-defined requirements. If the operation is graded invalid, another warning is passed to the user. if provenance recording is enabled, the wrapper passes all collected provenance and metadata to the internal environment of the 'SpatialSemantics'-package, where they are stored and incorporated in the data derivation graph. Finally, the annotated output is returned from the function wrapper.

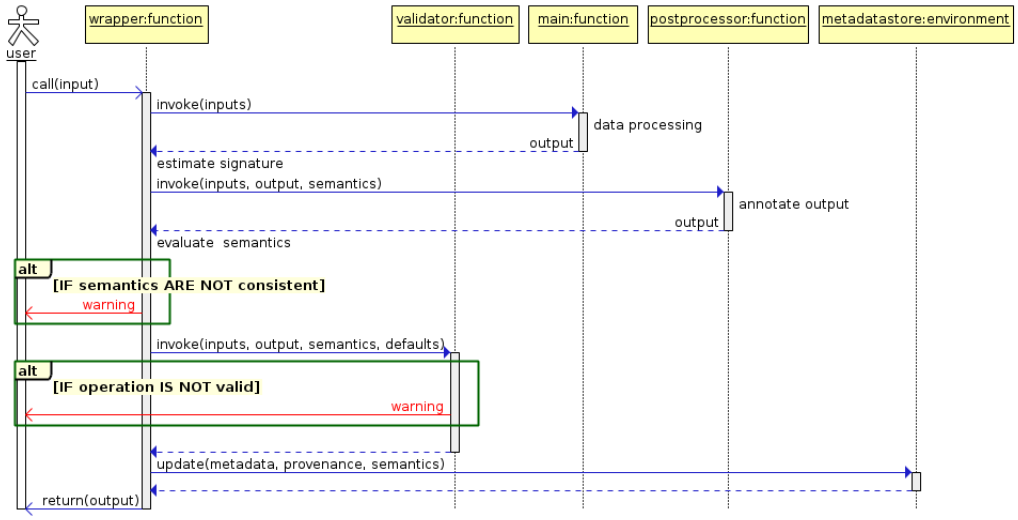


FIGURE 4.2: Sequence diagram of a call to a semantic function wrappers

In order to demonstrate the usage of function wrappers on a simple example, the following code listings are defining wrappers of a simple log-function. The log-function, as defined in the base-package of R, by default computes the natural logarithm of a given argument x. *Log* is a primitive function in R. Because primitive functions behave slightly different than standard functions, the prototype does not allow to wrap them directly at the moment. This is why listing 4.11 starts by defining a standard function around *log*.

Standard functions can be wrapped by just applying the replacement

function *captureProcenance*<-. This function creates a new function of identical name and signature in the user workspace and acts as a proxy to the original function. Wrapping can be reverted by either deleting the wrapper (provided that the wrapped function *does not* belong to the global environment or by calling *captureProcenance(fun)*<- *FALSE*, which will cause the wrapper to replace itself with the wrapped function.

Code listing 4.11 first applies a wrapper to the log-function that is not parametrized (line 5). This wrapper basically estimates the implicit function signature annotates the output by adding the function itself and its signature as a generation procedure to the output's semantic pedigree. Function wrappers may have predefined default-semantics (expected semantics). They can by queried by invoking *getDefaultCallSemantics* (lines 6 and 7). For the firstly defined wrapper, the function returns "dynamic", which means that no default semantics are defined and the signature estimation may vary during runtime. Using the *'semantics'* argument of *captureProvenance*, a user can specify one or more default signatures using semantic terminology. This is exemplified by the lines 9 till 11.

LISTING 4.11: Define semantic wrapper with and without default semantics

```
1  > library(SpatialSemantics)
2  > log = function(x){
3  +   return(base::log(x))
4  + } #direct wrapping or primitive functions is currently not supported
5  > captureSemantics(log) <- TRUE
6  > getDefaultCallSemantics(log)
7  [1] "dynamic"
8  > captureSemantics(log) <- FALSE
9  > captureSemantics(log, semantics = c("Q -> Q", "Q set -> Q set")) <- TRUE
10 > getDefaultCallSemantics(log)
11 [1] "Q -> Q"          "Q set -> Q set"
```

It is also possible to define default semantics per function call. Therefore, function wrappers have an 'semantics'-argument as well. Code listing 4.12 exemplifies this in line 7. If the semantics are specified per-call then any previously defined default semantics are ignored for this call. The listing also exemplifies consistency checks: if the estimated semantics during runtime does not comply with any of the pre-defined default-semantics, warnings are prompted (lines 3-6 and 7-9). Within the associated data derivation graph, such calls are marked by the keyword *INCONSISTENT*. In the listing, the first warning is prompted because the input does not comply with default semantics (call in line 3) and the second is prompted, because the call does not comply with the per-call semantics (line 7).

LISTING 4.12: Consistency checks on log-function

```
1  > input = 1
2  > attr(input, "semantics") <- "'a"
3  > result=log(input)
4  Warnmeldung:
5  In log(input) :
6    Inconsistent function semantics, given is 'a -> 'a but expected was one of
       the following: Q -> Q, Q set -> Q set
7  > result=log(input, semantics = c("Q set -> Q set"))
8  Warnmeldung:
```

```
In log(input, semantics = c("Q set -> Q set")) :
  Inconsistent function semantics, given is 'a -> 'a but expected was one of
      the following: Q set -> Q set
```

Further capabilities of semantic wrappers are postprocessors and validators (Figure 4.2). Both are functions which can be specified or customized by users and/or developers. Code listing 4.13 exemplifies this on the example of the log-function: A postprocessor (lines 5-7) has the formal arguments *args* (a named list of all inputs), output (the wrapped functions output data) and semantics (a string denoting the estimated function signature). The function returns the processed output, which is intended to be the wrapped functions output enriched with semantic annotations.

If a validator (lines 9-15) is specified, it is called *after* the postprocessor. It has the same arguments, except that the estimated signature are updated if semantics of the output were re-defined by the latter function. In addition the validator receives the default call semantics (as returned by the function *getDefaultCallSemantics*). The validator is a Boolean function that returns TRUE/FALSE if the operation is valid/invalid respectively. The function may prompt individual warnings if the overal operation is not valid by the user's requirements. In any case of invalidity, a generic warning is prompted by the function wrapper and the function call that was executed by the user is marked by the keyword *INVALID* within the data derivation graph. The validator in the example checks if the wrapped function's input consists of one or more quality values Q. If this is not the case, the call $log(input)$ is not meaningful and therefore invalid.

LISTING 4.13: Defining simple postprocessors and validators

```
1  log = function(x){
2    return(base::log(x))
3  }
4  #define postprocessor:
5  postprocessor = function(args, output, semantics){
6    addSemanticPedigree(obj = output, name = "log", procedure = "Q -> Q", result
         _semantics = "Q set")
7  }
8  #define validator:
9  validator = function(args, output, defaultSemantics, semantics){
10   in_sem = getObjectSemantics(args$x) # get semantics of input 'x'
11   valid = in_sem %in% c("Q", "Q set") # requirement
12   if(!valid){
13     warning(paste0("Invalid input of type ",in_sem,"! Expected Q or Q set"))
14     return(FALSE)
15   }
16   return(TRUE)
17 }
18 #create parametrized function wrapper:
19 captureSemantics(log, validator = validator, postprocessor= postprocessor) <-
       TRUE
```

Code listing 4.14 puts the above create function wrapper of log into action. Line 1 computes the natural logarithm of 1. The result (which is 0) is then queried for semantic pedigree that was defined by the postprocessor (line 2). The corresponding record is displayed in line 3 and it can be seen

that the result data was defined Q set, i.e. a (singleton) collection of quality values. Of course the descriptor can be fine-tuned to assign just Q in case the input has length 1, but this definition suffices demo-purposes.

In lines 6-8, the log-function is again called with the argument 1, but in this case the validator prompts a warning. Line 7 assigns the semantic type D to the input value. D is a basic type used to refer to discrete *identifiers*, mainly to identify objects and events. While it is generally assumed that numbers refer to quality values Q (see Appendix A), computing natural logarithms from identifiers D is not meaningful.

LISTING 4.14: Applying simple postprocessors and validators

```
1  > result=log(1)
2  > getSemanticPedigree(result)[1:4]
3    procedureName procedure result_attribute result_semantics
4  1           log     Q -> Q              ALL            Q set
5  >
6  > input = 1
7  > attr(input, "semantics") <- "D"
8  > result=log(input)
9  Warnmeldungen:
10 1: In validator(args, output, semantics, call_semantics) :
11   Invalid input of type D! Expected Q or Q set
12 2: In log(input) : Post-validation of function call failed!
```

4.2.3 Graph visualization and syntax

This section describes how the 'SpatialSemantics' package visualizes data derivation graph and how they are interpreted. The basic graph syntax and semantic terminology by (Scheider et al., 2016) have been explained in section 3.4 of this work. The following section 3.5 explained the approach to handle provenance information within this work. It closes with the proposal of syntax add-ons to the existing graph representation in order to incorporate provenance information and to keep references to the underlying source-code, with the intention to improve comprehension.

Figure 4.3 refers to code listing 4.2 in the beginning of section 4.2. It is the visualization of the therein shown example of automated provenance recording. It appears in this section in order to exemplify the graph syntax. A curiosity of this graph is that *meuse* is referenced four times in nodes that denote data (white nodes), but only the latter three occurrences refer to a variable named meuse. The first occurrence refers to a literal value that *is* the expression 'meuse'. The subsequent nodes labeled meuse, meuse 2 and meuse 3 refer to an actual dataset, which has been modified three times during the execution. The meuse-dataset is created by a call to the function *'data'*, which does not returns the dataset explicitly but rather creates it via a side-effect (dashed red arrow). Expressions from the command line interface are generally mapped to the graph by breaking the down into function calls and data according to its parse tree. Nevertheless, it can be observed that some nodes do not refer to atomic expressions: the expressions *c("x",y")*, *log(meuse$zinc)* and *meuse$lzinc* are not broken down into individual parts, despite not being atomic. Expressions that only consist of

primitive functions or operations are kept as one in order to keep the graph simple and readable.

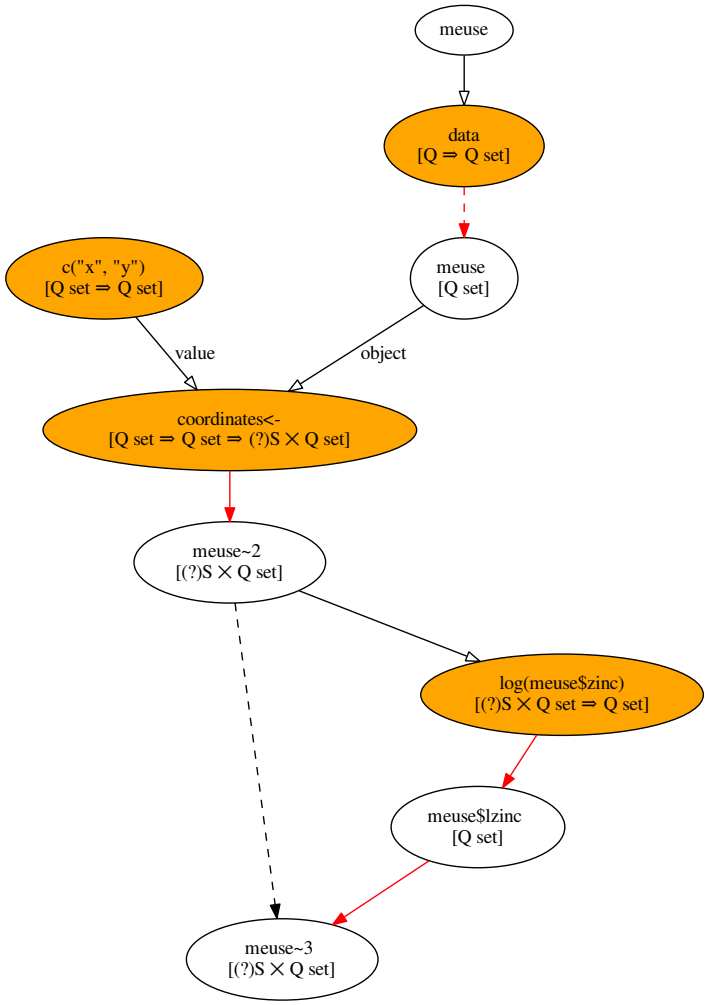


FIGURE 4.3: Simple derivation graph resulting from the dot-file created by the code in listing 4.2

The following paragraphs explain the extended graph syntax as far as it differs from (Scheider et al., 2016). It shall be pointed out that the syntax is still downward-compatible and can be converted to the basic syntax, as shown below.

Figure 4.4 shows possible appearances of graph nodes. In general, function calls or primitive operations are represented by orange nodes, while data is represented by white nodes. In cases were function inputs are atomic literals and not referred to by a variable (1), they are displayed *as is*. Any data that is referred to by a name binding (2) is referred to by an instance identifier (IID), which is composed of a variable name and an incremental version number (see section 3.5.1. In the second line of the label always appear the semantic annotation in brackets, except for case (1), in which it

is omitted. Data may also be referred to by a simple expression (2), denoting a *subset* of data (compare with Figure 4.6). In this case no explicit IID reference is necessary because the parent dataset is referenced in conjunction as an input to the data. Function calls (3) are labeled with the name of the called function and their semantic signature in brackets. In case the node represents a call that consists of one primitive operations, the node is labeled by the complete expression. The second column of the figure shows how each of the representation can be converted/reduced to the basic syntax.

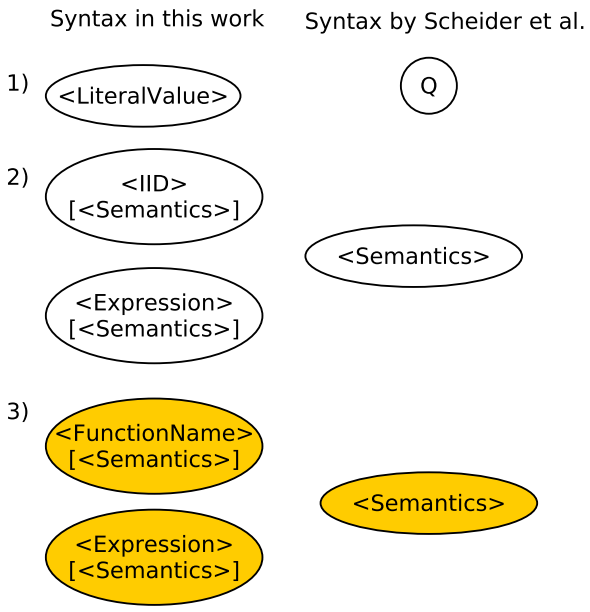


FIGURE 4.4: Syntax of graph nodes

Figure 4.5 shows possible appearances of edges. While Scheider et al. (2016) just differentiate between input- and output- relation, the graphs in this work have a finer distinction of these relations. First of all, any inputs, outputs or function calls that are side-effects of a corresponding function-call are indicated by dashed arrows, which indicate that the relation between the nodes are not explicit by the code, but inferred or estimated.

Regular inputs (1) to functions or expressions are represented identical to the basic syntax, except that they are annotated with the argument name (<argName>) as defined by the function's argument list, if available. If the function is present as an object in the user's workspace, derivation graphs also specify a relation from the function to the call that invokes this function (2), represented by a blue arrow. In terms of Scheider et al. (2016), this relation would correspond to a simple input, for example a generation function that is input to a data generator or a derivation function. Output- relations in this work (3) are represented by red arrows. This modification was simply applied for visual purposes to better distinguish the edges from each other.

In R it is common to modify or read not one dataset as a whole, but a fraction of it, i.e. a subset, which can be accessed by primitive operators

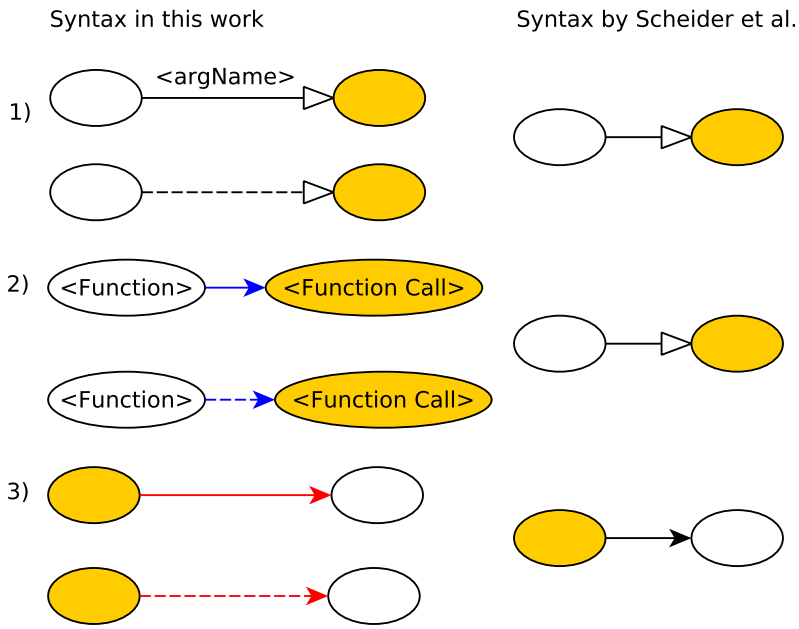


FIGURE 4.5: Syntax of graph nodes

such as $, [, [[and @. For representing these common cases of reading and writing a subset, this work uses two patterns, which are displayed in 4.6. In case of reading or retrieval (1), the parent dataset is a direct input to the subset-expression - a shorthand for explicitly representing the data-access operation as a call. (2) In case a subset of an operator is modified, the modified data is an output of the subset-expression with an implicit relation of the original dataset to its modified version. In basic syntax, the subset and original dataset would be concatenated to a new dataset.

The latter pattern is also apparent in Figure 4.3: The graph branches off and joins together where subsets off the *meuse*-dataset are processed.

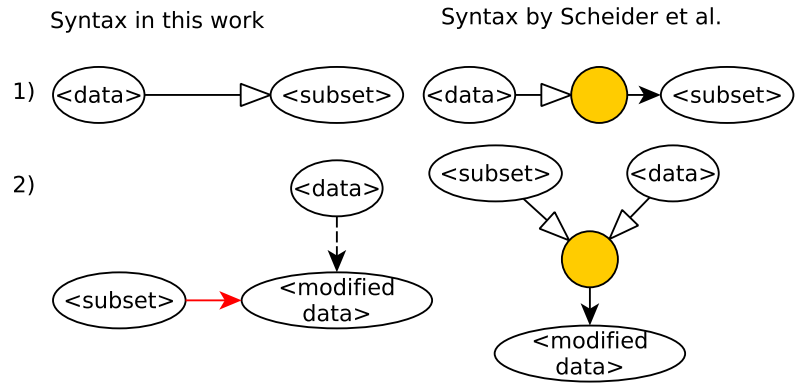


FIGURE 4.6: Syntax of reading/writing subsets

Chapter 5

Use Cases

This chapter applies the R prototype described in chapter 4 based on the methodology described in Chapter 3 on common analytical problems of Spatial Statistics. It shows how this work can contribute to better communicate spatio-temporal data analysis. Therefore, this chapter presents three use cases in each of the following sections. Each use case is derived from scientific work that is already published.

The following sections all share a similar structure. First, they giving an overview over the use case by describing the analytical problem, how this problem is addressed by spatio-temporal analyses and which data and utilities are used. It follows a subsection describing how the analysis is realized in R and how it is interpreted in terms of semantics. The third subsection describes in a detailed walk-trough how the 'SpatialSemantics'-package can be applied to the use case. The use cases are evaluated in the third section of Chapter 6.

5.1 Spatio-temporal aggregation of bird counts

5.1.1 Overview

A common analytical problem in Spatial Statistics is the aggregation of values over space and/or time. Such problems comprise for instance summarizing daily mean temperatures from different measurement stations to monthly averages or summarizing electoral results from electoral districts to counties.

The herein presented problem requires aggregation over both, space and time. The analytical approach was first published by R. S. Bivand, E. Pebesma, and Gómez-Rubio (2013, pp. 156-161) and therein also demonstrated in R. The data to be analysed origins from the North American Breeding Bird Survey[1]. and contains observation counts of the Eurasian Collared Dove (*Streptopelia decaocto*), which is an invasive bird species in northern America. Since the 1980s, this bird occurs in the US-state of Florida. For this state we would like to know how much the bird population increased in 2 year periods between 1989 and 1999.

The dataset was first published by by Cressie and Wikle (2011). It consists of two tables that can be downloaded from the book web site[2]. The table ECDovelatlon.dat contains 253 locations were birds have been observed, represented by latitude / longitude coordinates. Figure 5.1 shows

[1] North American Breeding Bird Survey: https://www.pwrc.usgs.gov/bbs/ (Last access on Sept. 19th, 2016

[2] ftp://ftp.wiley.com/public/sci_tech_med/spatio_temporal_data/

those observation locations which are in or nearby Florida. Nevertheless, observations are distributed over a larger area in northern America.

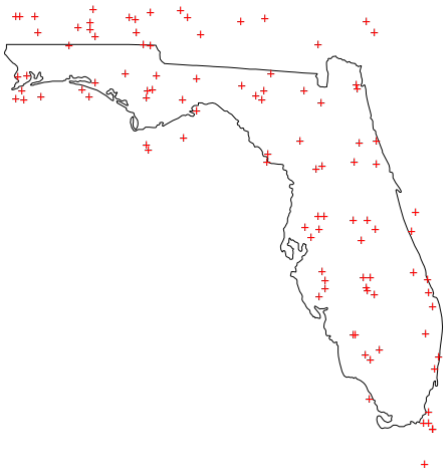


FIGURE 5.1: Bird observation locations near Florida

The table ECDoveBBS1986_2003.dat contains 18 columns and 253 rows. Each row refers to an observation location, while the columns refer to the years in which the observations were taken, i.e. from 1986-2003.

Hence, the overall dataset contains yearly bird counts per location over 18 years and 253 locations. In order to determine the population count for the state of Florida in 2-year periods, it must be determined which observations fall within the state borders and to sum them up for every 2 years.

Table 5.1 shows the simplified result for the analysis as presented by R. S. Bivand, E. Pebesma, and Gómez-Rubio (2013). It can be observed that the population Eurasian Collard Dove steadily increased, and that the relative growth rate over 2 year periods is exponential.

LISTING 5.1: Use case 1 - spatial prediction results

	counts	timeIndex
1989−01−01	3	1
1991−01−01	5	2
1993−01−01	92	3
1995−01−01	176	4
1997−01−01	860	5

5.1.2 Approach

The analysis makes use of the packages sp (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013) for spatial data handling, spacetime (E. Pebesma, 2012) for spatio-temporal data handling, xts (Ryan and Ulrich, 2014) for representing time series data, maps (Richard A. Becker, Ray Brownrigg. Enhancements by Thomas P Minka, and Deckmyn., 2016) for retrieving the boundaries of the US-state Florida and maptools (R. Bivand and Lewin-Koh, 2016) for converting the boundary data into the SpatialPolygons-class from the sp-package.

Both tables from the North American Breeding Bird Survey are imported into R by the read.tables function. They are then processed and combined to a a spatio-temporal full lattice data frame (STFDF), consisting of three independent slots for time, sp, and data.

A target geometry is built as a spatio-temporal full lattice (STF), which has the same structure as the STFDF, except that it contains no data frame. The target geometry consists of the Florida state boundaries, represented by a spatial polygon and a time series consisting of dates from January 1989 to 1997, marking the beginning dates of each 2-year period. The data these datasets are inputted to a function named *aggregate*, which performs an aggregation over the target-geometry by summing up values over space and time. The result is a time series dataset consisting of summed up bird population counts for each 2-year period in respect of the US state Florida (Listing 5.1).

A challenge is to describe the components of this analysis regarding their meaning. According to the terminology by Scheider et al. (2016), they can be interpreted as the following: Bird counts per year and location are represented by a MarkedEvent, formally defined by $D \Rightarrow S \times T \times Q$. The dataset itself consists of sets of point locations S, moments in time T, given by year and bird counts as quality values Q. They are combined to a set of tuples $S \times T \times Q$. A discrete identifier D that refers to the event (of observation) is not explicit in the data.

The target geometry for the aggregation is simply a tuple of a region (Florida boundaries) with a set of time intervals (2-year periods) $R \times I\ set$. No functional type needs to be assigned.

The time-series data that results from the aggregation is generated from a temporal lattice $TLattice :: I \Rightarrow Q$, while the resulting dataset is a collection of tuples of time intervals and quality values (bird counts), i.e. a $Q \times T\ set$.

The technical approach is outlined in Chapter 3, but will be illustrated by this example for better comprehension. First, a to a data derivation graph is recorded. It is then automatically populated it with heuristic semantics a mapping from data properties (see appendix A). This process is further automated by semantic inference (e.g. estimation of function signatures). While the graph is created in the background, a user receives warnings from the command-line. The derivation graph signals semantic ambiguities which prevent an assessment meaningfulness of spatio-temporal data generation with question marks. This helps the user to subsequently assign semantic annotations where necessary and thus to refine the graph.

Writing semantic wrapper-functions with postprocessors can further automate the process because the functions' output are thus annotated during the function call. These functions also can be re-used for other purposes. Alternatively, the user can annotate data manually by setting the functional type (functionalType-function) to any spatio-temporal data sets that represent observations. Setting the semantics-attribute is suitable for all other data that shall not be attributed regarding their purpose.

5.1.3 Implementation

The fully annotated R script referring to this analysis is included in appendix C. For simplicity, details about the data preprocessing part of the

script are omitted in this text. The derivation graph also abstracts over these procedures by starting recording directly when the spatio-temporal dataset is created from locations (ecd.locations), a collection of dates (ecd.years), and the observed data (ecd.data).

This use case demonstrates how warnings prompt to the user to revise the semantics of the derivation graph by adding annotations. Figure 5.2 shows the warnings that occur when doing provenance recording without applying semantic annotations. The user is prompted to review the semantics of the spatio-temporal dataset (bird counts, named ecd.st) and the semantics of the target space-time geometry (named target.st). Figure 5.2 shows the derivation graph resulting from this execution. Question-marks appear for the aformentioned datasets as well as in the signatures of the functions STFDF and STF (constructors for spatio-temporal objects), as well as for the aggregate-function.

LISTING 5.2: Use case 1 - recording without annotations

```
> #Analysis starts here:
> enableProvenance()
>   #Create space-time object of all bird counts
>   ecd.st <- STFDF(ecd.locations, ecd.years, ecd.data)
Warnmeldung vom toplevel task callback '1'
Warnmeldung:
In getObjectSemantics(var0) :
  No semantic annotation available for object ecd.st. Assumend semantics will
      be: (?) S x T x Q set
>   #Create target-geometry to aggregate over Florida-state area
>   # in 2-year periods
>   target.years = ecd.years[c (4,6,8,10,12) ]
>   target.st <- STF(FL, target.years)
Warnmeldung vom toplevel task callback '1'
Warnmeldung:
In getObjectSemantics(var0) :
    No semantic annotation available for object target.st. Assumend semantics
        will be: (?) R x (T set)
>   #execute aggregation
>   ts = aggregate(ecd.st, target.st, sum, na.rm = TRUE)
> disableProvenance()
```

Listing 5.3 shows changes that were subsequently made on the R-script in order to revise the initial semantic provenance. These commands are executed *before* the analysis is recorded. In this case it is not necessary to place any annotation-functions directly inside the analysis-part. Instead, some of the existing functions are replaced by wrappers: The STFDF-constructor is wrapped by a function called STFDF.MarkedEvent that annotates outputs with the functional type $MarkedEvent$ (the parameter parent = FALSE causes that the discrete identifier D is not replicated in the dataset-representation). This is achieved by the captureSemantics-function parametrized with a defined postprocessor. Unfortunately, the captureSemantics-function cannot be applied to the aggregate-function due to technical issues (compare with 7, first section). Instead, a simple wrapper is written manually that annotates the output as $TLattice$ data. A similar wrapper is written for the STF-constructor, with the difference that it annotates only the time-slot of its output according to *input*. The semantics of an STF-object can be determined if the semantics of both slots (sp and time) are known (the semantics

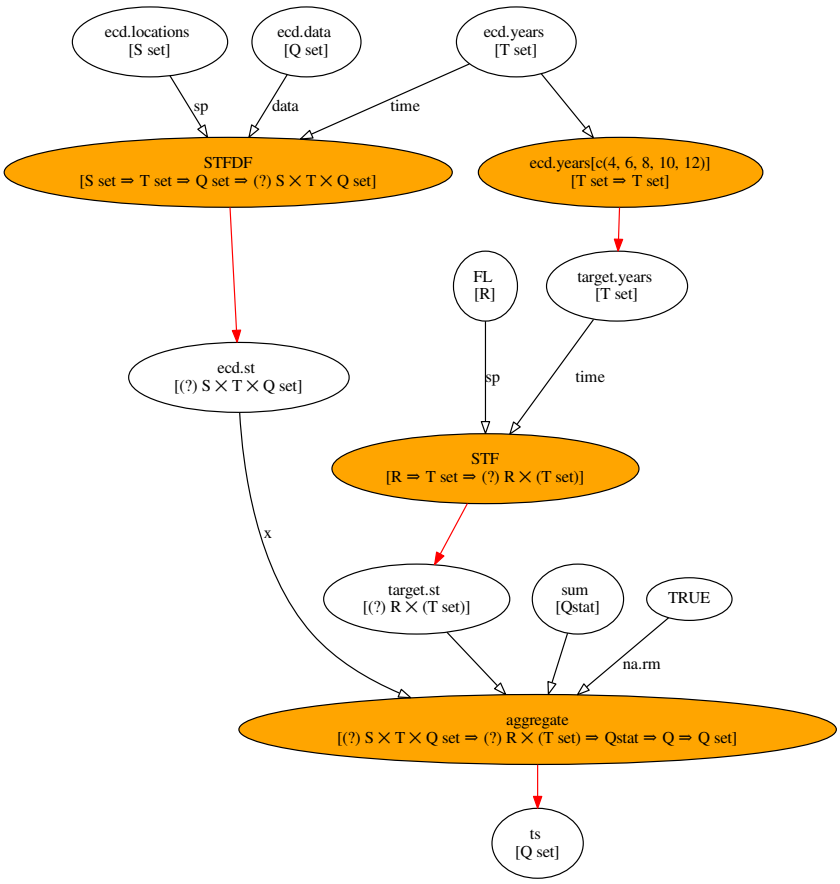


FIGURE 5.2: Derivation graph without user-defined semantic annotations

from the sp-slot are preserved from the input). Finally, it is necessary to write a simple function named getTimeIntervals, which not only samples time intervals from a given collection of dates, but also annotates the output according to their meaning.

These functions are then incorporated in the analysis-part of the R-script. The final, correctly annotated graph is displayed in Figure 5.3. Code listing 5.4 exemplifies the output of getSemanticPedigree when querying semantics of the individual datasets, in this case the generated time series object ts.

LISTING 5.3: Use case 1 - Definitions of semantic function wrappers

```
# Create function wrapper for STFDF constructur
postprocessor = function(args, output, semantics){
  functionalType(output, parent = FALSE) <- "MarkedEvent"
  return(output)
}
STFDF.MarkedEvent = STFDF #give constructor a more informative name
```

```
captureSemantics(STFDF.MarkedEvent, postprocessor = postprocessor) <-
    TRUE
# Create manual wrapper function arround aggregate (direct wrapping currently
    not supported)
aggregate.st <- function(x, by, FUN, na.rm){
  output = aggregate(x, by, FUN, na.rm = na.rm)
  functionalType(output) <- "TLattice"
  return(output)
}
#encapsulate intervall-subsetting as a semantic function
getTimeIntervals = function(T_set, breaks){
  output = T_set[breaks]
  attr(output, "semantics") <- "I set"
  return(output)
}
#encapsulate STF-constructor in a function that preserves semantics
STF.sem = function(sp, time){
  output = STF(sp, time) #sp semantics are preserved already
  attr(output@time, "semantics") <- attr(time, "semantics")
  return(output)
}
```

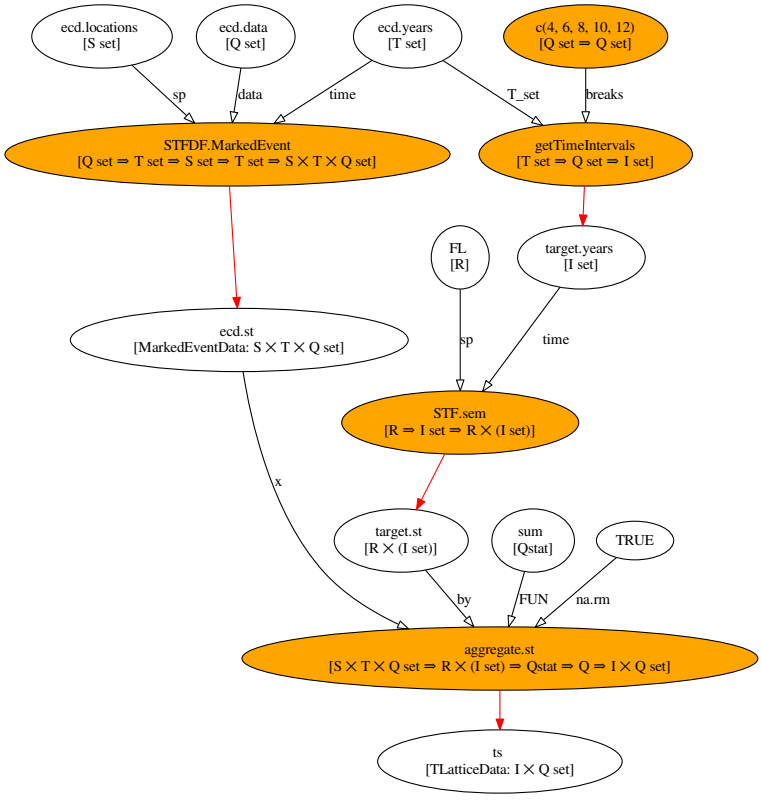


FIGURE 5.3: Derivation graph that includes user-defined semantic annotations

LISTING 5.4: Use case 1 - query of semantics of the generated time series

```
> ##Query semantics of ts (aggregation result)
> getSemanticPedigree(ts)[1:3]
  procedureName procedure result_attribute
1       TLattice      I -> Q              ALL
> getSemanticPedigree(ts)[4:5]
  result_semantics parent_semantics
1            Q set          I x Q set
> getSemanticPedigree(ts)[6:7]
  rec_num                                                    command
1       4 ts = aggregate.st(ecd.st, target.st, sum, na.rm = TRUE)
```

5.2 Spatial prediction on the meuse river floodplain

5.2.1 Overview

The meuse river data was collected during fieldwork by Rikken and Van Rijn (1993). It contains top-soil heavy metal concentrations of the meuse floodplain, taken from locations in a study area near the village of Stein in the Netherlands. Considering that heavy metal concentrations are just known for punctual locations, it suggest itself to spatially predict the concentration over the complete area.

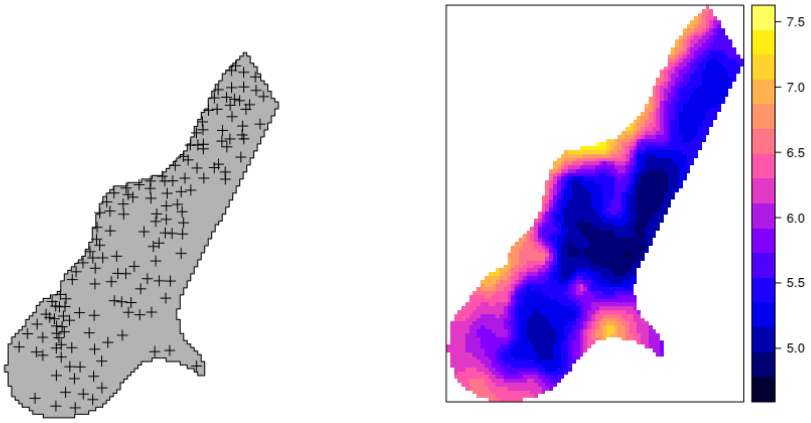
The dataset was compiled by Edzer Pebesma and the description extended by David Rossiter. It is included in the sp-package and used to exemplify Spatial Analysis, including prediction, by R. S. Bivand, E. Pebesma, and Gómez-Rubio (2013). The dataset contains a set of 155 observation locations that are described by 12 different variables, amongst others top-soil concentrations of cadmium, copper lead and zinc. Further variables refer to soil and landscape properties. Supplemental data include a prediction grid for the study area (muese.grid, cell spacing 40×40 m) and an outline of the study area (meuse.area), given as a spatial polygon.

Bivand et al. suggested different techniques to spatial prediction in this context. This use-case is based on an ordinary kriging prediction based on a variogram model, as it is described in the book (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013, pp. 224-233). Figure 5.4 shows the study area and observation locations on the left, and the result of the kriging-prediction on zinc concentration on the right, which results from the analysis as shown in the following.

5.2.2 Approach

The analysis makes use of the packages sp for (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013) for representing and handling spatial data as well as the packages gstat (E. J. Pebesma, 2004) for variogram modeling and spatial prediction. Furthermore it was decided to include the prototypical package mss (C. Stasch, Scheider, et al., 2014) in the use case implementation, which facilitates meaningful spatial statistics. The mss-package is not included in the original approach, but the analysis is functionally equivalent.

The analysis is conducted by fitting a linear variogram-model to the point observations. Variogram models represent assumptions about the



(A) Ground sampling locations

(B) Interpolated log-transformed zinc concentration

FIGURE 5.4: Zinc concentration prediction at the meuse floodplain

spatial process that generates the data. In particular, they reflect on the extent of spatial auto-correlation of observations as well as on the nugget variance, which incorporates measurement errors and / or micro-variability of the data.

This model is inputted to an interpolation function, which then predicts values on a given target geometry, i.e. a spatial grid. (Optionally it can also compute prediction variances, giving an uncertainty measure for the predicted values, respectively the range in which the predictions deviate from observations)

The semantic assumption on the observed heavy metal concentrations as well as on the predicted values is that they are generated from a spatial field ($SField$), formally defined by $S \Rightarrow Q$. It is a special case of a $Field$ $S \times T \Rightarrow Q$ which describes phenomena over space S and time T. An $SField$ describes phenomena over locations within a fixed time.

The mss-package includes a class *SField*. Objects of this class represent are data that were generated from a spatial field ($SField$) process. They have a slot named *observations*, which keeps the observed data, and a lot named *domain*, which refers to the spatial extend ($SExtent$) in which the spatial phenomenon was observed. The latter also causes interpolation-procedures to interpolate only in this area, because the $SExtent$ implies that it is unknown whether the phenomenon exists outside.

When the analysis is executed in R, the meuse data set (meuse) is reduced to log-transformed observations of zinc concentrations. The subset and meuse study area (meuse.area) are then combined by an mss::SField container (zincPointData). The zinc point observations are represented by an $S \times Q$ set. This is a collection of tuples of spatial locations and quality values, generated from an $SField$ by the gendata-function (data generator) from the algebra. The SField-container is represented as a tuple of SField data and its spatial extent, i.e. $(S \times Q\ set) \times SExtent)$. Similarly, the target geometry of the prediction (locInterest) is constructed as $(Sset) \times SExtent$, which represents the spatial locations targeted by the prediction and the extent of the phenomenon. The interpolated data (intZincPointData) has

the same representation and semantics as the observed data it is based on (zincPointData).

The variogram model is fitted to the observed zinc concentration (zincPointData) by a function called modelSemivariogram. The model is passed on to a function called getInterpolator, which returns a so-called closure. A closure is a function that inherits data from the environment in which it is created, i.e. it is a function that is written by another function and contains data (Wickham, 2014, p. 183). In this way, a function called interpolator is created. The interpolator is parametrized with locations of interests. Therupon, the function interpolates the targeted zinc concentrations (intZincPointData) for a given set of locations (locInterest).

Spatial semantics are assigned to the observed data with the functionalType-function and to the interpolated data by modifying the interpolator in a way that it assignes the functional type during execution. Also, all functions are wrapped by the captureSemantics-function, which enables semantic parametrization (postprocessor, validator, default signatures) and partially improves the inference of signatures (e.g. default values are included which are not explicit inputs by the function call).

During the developement of this use case, it was discussed that the interpolator must have the signature $S \Rightarrow S \times Q$, respectively $Sset \Rightarrow S \times Qset$. Hence, these semantics were assigned as the default semantics to the interpolation. However, in reality and due to the representation of the SFieldcontainer the *de facto* signature of the interpolator-call is $(Sset) \times SExtent \Rightarrow (S \times Qset) \times SExtent$. This mismatch is detected as a semantic inconsistency and accordingly reflected in the derivation graph.

5.2.3 Implementation

One pragmatic challenge to this use-case is that the overall derivation graph is way too large to be depicted within this text (Figure 5.5). This problem is addressed by only recording provenance from that part of the analysis that is essential and then displaying the reduced derivation graph (Figure 5.6). This demonstrates that the user can be selective about the information that is recorded.

The complete R script referring to this use case is depicted in appendix C. The code listing 5.5 displays only the analysis part of this script. It responds to the derivation graph depicted in Figure 5.5, which is way to large to be displayed herein with all details. A workaround is to record provenance only from line 8 on, i.e. by repositioning the enableProvenance() -function in line 7. The resulting sub-graph (Figure 5.6) represents the core of the analysis, i.e. model-fitting by modelSemivariogram, construction of a model-based interpolator and the interpolation itself. It can be observed that the interpolator-call is marked by the keyword *INCONSISTENT*, because the de facto semantics of the call, which is estimated by inference from inputs and outputs, are not consistent with the default-semantics, indicated by the semantics-parameter of the function.

It can be further observed that the function called *modelSemivariogram* internally calls a function *init_model*, which is also present in the user's workspace. This is one of the side-effects that the prototype can detect. It thus can produce derivation graphs of greater detail and accuracy.

LISTING 5.5: Use case 2 - analysis part of R script

```
enableProvenance() # Run analysis
#  load meuse data from package sp in current session:
demo(meuse, ask=FALSE, echo=FALSE)
functionalType(meuse) <- "SField"
functionalType(meuse.grid) <- "SField"
meuse$lzinc = log(meuse$zinc)
zincPointData = SFieldData(meuse["lzinc"], meuse.area)
interpolator  = getInterpolator(modelSemivariogram(zincPointData),
    zincPointData)
locInterest  = SFieldData(geometry(meuse.grid), geometry(meuse.grid),
    cellsArePoints = TRUE)
intZincPointData = interpolator( locInterest ,  semantics = "S  set  -> S x Q set")
disableProvenance() #end of analysis
```

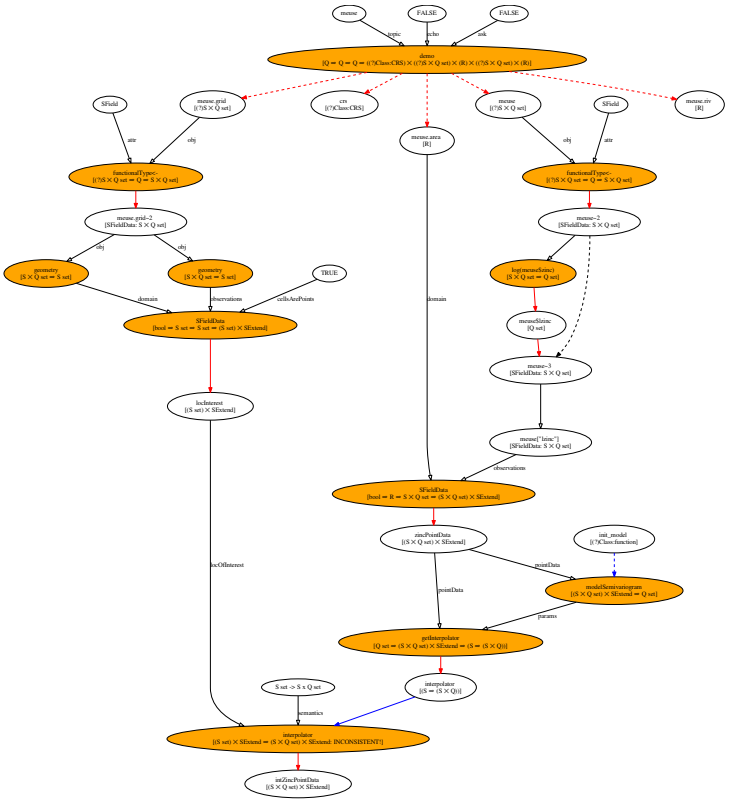


FIGURE 5.5: Meuse prediction data derivation graph

Listing 5.6 shows the semantic pedigree of the interpolated data: The dataset as a whole is annotated with the interpolator as a data generation procedure. In addition, a message is displayed that pedigree is available for the observations-slot. An inspection of the slot indicates that the data therein was generated from a spatial field.

LISTING 5.6: Use case 2 - pedigree of the interpolated zinc point data

```
> getSemanticPedigree(intZincPointData)[1:3]
Information: Semantic pedigree is available for the following slot(s):
    observations
  procedureName                                    procedure result_attribute
```

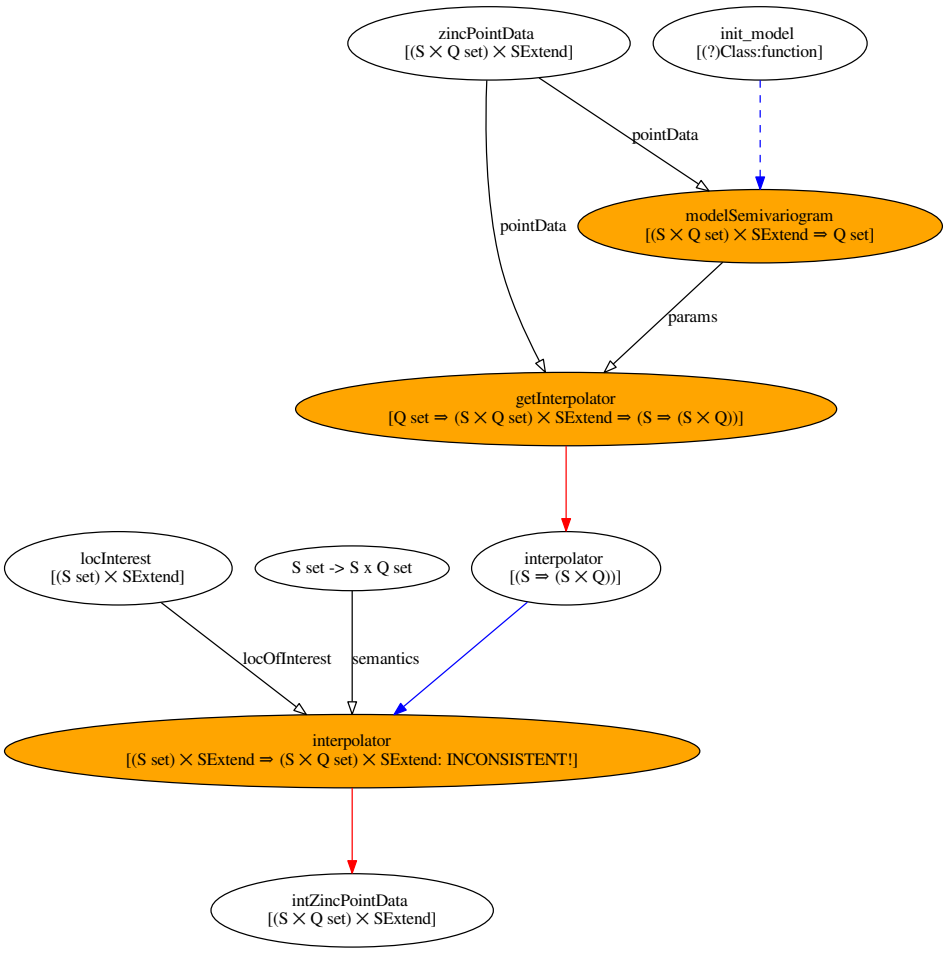


FIGURE 5.6: Meuse prediction data derivation sub-graph

```
1  interpolator  (S  set)  x SExtend −> (S x Q set) x SExtend          ALL
>
> getSemanticPedigree(intZincPointData@observations)[1:3]
   procedureName procedure result_attribute
1          SField      S −> Q                ALL
```

5.3 Clustering malaria episodes in Bandiagara, Mali

5.3.1 Overview

In a study that was first published by Coulibaly et al. (2013), incidents of malaria episodes were recorded amongst children that were aged under 6 years. The study was carried out in Bandiagari, Mali during a transition season, precisely from June 2009 until May 2010. From 300 children that participated, 296 clinical malaria cases were recorded.

A common problem in epidemiology is to detect spatial clusters in an environment; are there areas in which the risk of malaria is higher or lower than in the the neighborhood? This question was addressed by Gaudart et al. (2015), who demonstrated a novel algorithm for spatial partitioning on the data. The algorithm computes so-called spatial oblique decision trees

(SpODT), which are used to determine boundaries between zones of different epidemic risks. Figure 5.7 shows the result of the algorithm as implemented for this use case. The analysis is functionally equivalent to the code example at page 9 of the paper.

Guardart et al. also proposed the SPODT-package for R, which compiles functions for spatial partitioning and also includes the dataset of the above mentioned study by the name dataMALARIA. The data is structured by a spatial points data frame, which contains 168 locations referring to the geo-referenced households in which the children lived. The data table contains two variables: loc represents the identifiers of the registered households and z are the number of malaria episodes per child for each household.

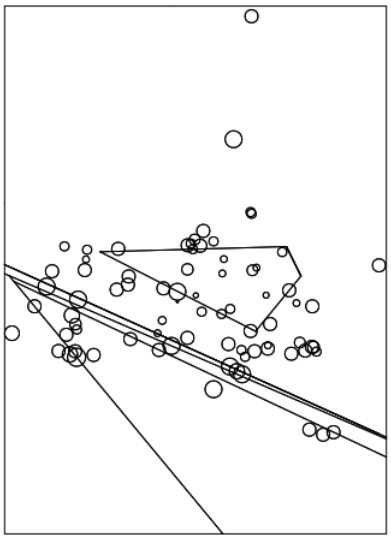


FIGURE 5.7: Spatial partitioning of malaria episodes. The plot is identical to Figure 4 in Gaudart et al. (2015, p. 10)

5.3.2 Approach

The SpoDT algorithm starts with computing risk indicators (probabilities) from the dataset and searches for a boundary that splits the study area into two areas where the risk indicators are as different as possible. The line is defined as a linear function of x and y coordinates. The algorithm then estimates boundaries lines that divide each of the two areas by the same criteria and continues this recursive splitting until pre-defined stopping criteria are met.

The algorithm is implemented in an R function of name *spodt*. It performs the classification of the dataset and computes a decision tree that is an object of class *spodt*. It includes all results from the algorithm including the splitting coefficients of each line and for each split the identifiers of the locations belonging to either of the two spatial classes. A function named *spodtSpatialLines* takes the epidemic data and the spodt-object as inputs and computes a spatial lines object.

According to the terminology by Scheider et al. (2016), the dataset dataMALARIA was generated from spatial marked events, which are defined

as $SMarkedEvent :: D \Rightarrow S \times Q$. Therefore, the data describes phenomena that occur at locations in space, which are malaria episodes of children in a household. The phenomenon is described over a fixed time, i.e. when the study was carried out.

In other terminology, the data can also be classified as spatial marked point pattern (C. Stasch, Scheider, et al., 2014), which equals to either spatial marked objects or spatial marked events. Which in compliance with Scheider et al. have the same formal definitions, i.e. $D \Rightarrow S \times Q$. It may be argued hence, that the latter definition also applies, as the point data refers to households, which are better described as objects. Nevertheless, it was decided in this work to use the notion of 'event' because the describes malaria occurrences, which are phenomena that are instantaneous in space and time, contrary to objects that can undergo change. The households are not the subject of the data, but rather the spatial reference of these occurrences. It must be added that Scheider et al. neither defined spatial marked events nor spatial marked objects *explicitly*, but rather they result from adding a quality domain to an $SEvent$ (event locations) or fixing the time domain of a Marked Object respectively.

The dataset dataMALARIA is typed as a $D \times S \times Qset$. It relates to a spatial marked event as the follows; the discrete event identifier D is replicated in the data by the variable loc (identifiers of the housholds' locations) and the quality value Q is represented by the variable z, which denotes the average numbers of malaria episodes per child in each household.

The analysis is carried out as follows: The SpODT-algorithm takes the malaria data as an input and internally computes risk estimates. It then provides a mapping from these risk estimates to spatial regions in which these estimates apply. The computed decision tree *is* this mapping. As a model, the tree thus fulfills the criteria of a spatial inverted field, defined by the expression $SInvField :: Q \Rightarrow R$. The *spodtSpatialLines*-function constructs spatial lines from the decision tree and the malaria data. It is remarkable that the resulting data is technically a collection spatial lines, but in terms of semantics it represents a collection of regions.

It shall be noted that the original analysis also includes a subsequent hypothesis test that validates the classification that is represented by the decision tree. This validation is *not* included in this use-case as the focus of this work lies on modeling data generation and transformation.

5.3.3 Implementation

For communicating the above-mentioned semantics and for applying the 'SpatialSemantics'-package to the R script, the following modifications are applied to the original code: spodt-function is enclosed by a wrapper (spodt.malaria) that takes the data and the list-compiled parameters of the spodt-function as inputs. The wrapper annotates its output, i.e. the decision tree as s spatial inverted field ($SInvField$) and includes a validator. The validator checks whether the input has the functional type $SMarkedEvent$ (i.e. if the data was generated from a spatial marked events). If not, the wrapper prompts a warning that indicates that the input does not match with the intended analysis. For general purpose, the validator may be extended to also accept other functional types.

The function spodtSpatialLines is also wrapped and extended with a postprocessor. This postprocessor annotates the output of the function, i.e. the spatial lines, with the *functional type* $SInvField$, from which also the *semantic type* representation of the data, i.e. an $R\ set$, is inferred. Code listing 5.7 shows these definitions in R code.

LISTING 5.7: Use case 3 - definitions of function wrappers, a postprocessor and a validator

```
validator = function(args, output, semantics, assumtions){
  data = args[["data"]]
  pedigreeData = getSemanticPedigree(data)
  valid = "SMarkedEvent" %in% pedigreeData$procedureName
  if(!valid){
    message= "Spatial partitioning with the given input data was not intended.\
        nExpected were SMarkedEventData, "
    if(is.null(functionalType(data)))
      message = paste0("but functional type of data is unknown.")
      else
    message = paste0(message, "but given were ",functionalType(data),"Data.")
    warning(message)
  }
  return(valid)
}
spodt.malaria <- function(params, data){
  args = append(params, list(data=data))
  output = base::do.call(SPODT::spodt, args)
  attr(output, "semantics") <-"SInvField"
  return(output)
}
captureSemantics(spodt.malaria, validator=validator) <- TRUE
postprocessor = function(args, output, semantics){
  functionalType(output, parent=FALSE) <- "SInvField"
  return(output)
}
captureSemantics(spodtSpatialLines, postprocessor=postprocessor) <- TRUE
```

As in the former use-cases, the provenance recording is limited to the core-part of the analysis. It leaves out for instance the pre-processing of the malaria data to a projected SpatialPointsDataFrame as well as the vizualization of the computed decision tree and risk areas. Code listing 5.8 shows the recorded commands and Figure 5.8 shows the spatio-temporal derivation graph that results from the recording. This graphs not only depicts the semantics of each data an function call, but also the complete parameter list of the spodt-function. In this way it is different from all other graphs presented in this work. Displaying lists of parameters is possible because the 'SpatialSemantics' package visualizes expressions like *list(param=..., param2=) as is*, provided that they do not contain calls to non-primitive functions.

LISTING 5.8: Use case 3 - core analysis part

```
enableProvenance()# start recording provenance
params = list(formula = z ~1, graft = 0.13, level.max = 7, min.parent = 25, min
    .child = 2, rtwo.min = 0.01)
spodt.results <- spodt.malaria(params = params, dataMALARIA)
#create spatial lines that divide the study area into regions of higher / lower
    malaria risk
```

```
SSL.result <- spodtSpatialLines(spodt.results, dataMALARIA)
disableProvenance()
```

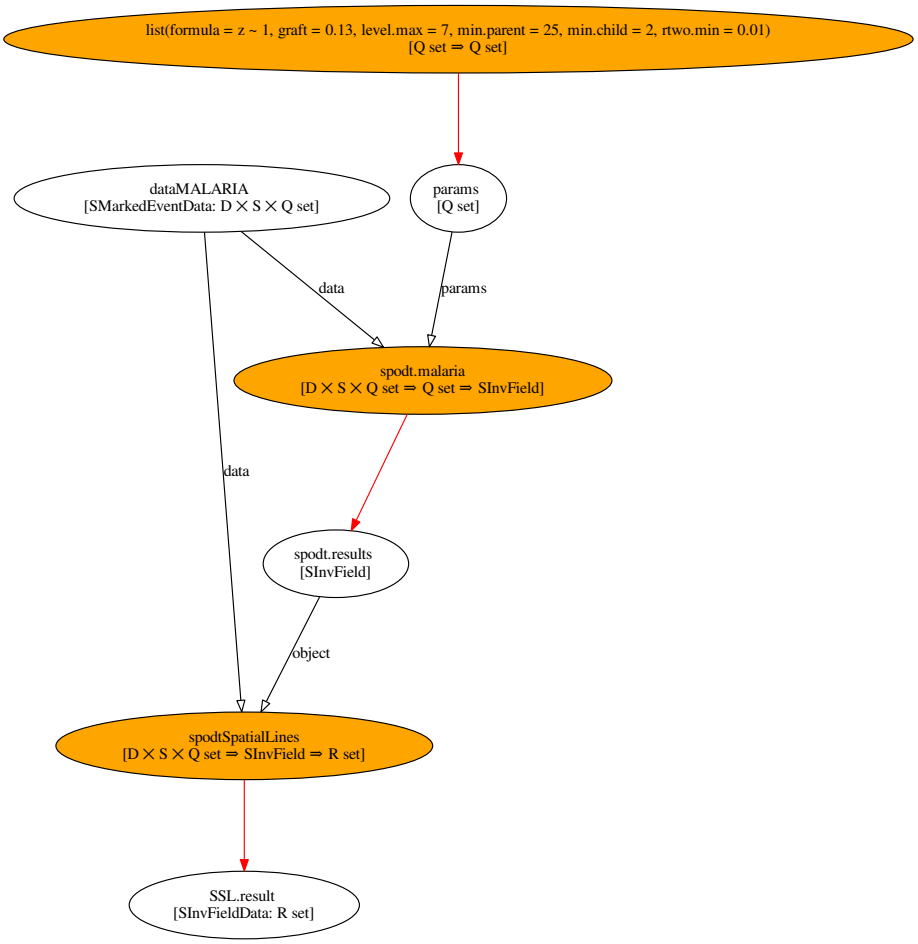


FIGURE 5.8: Derivation graph of the spatial partitioning analysis

In code Listing 5.9, the validator that is integrated in the spodt.malaria - function is tested against the meuse dataset section 5.2). The function indeed computes a classification tree from the meuse ground samples, but a warning indicates that this is not the intended usage of the function. The algorithm is designed for point-pattern analysis, but the data is generated from a spatial field. Thus, applying the function is not in compliance with the data's purpose.

In a corresponding derivation graph, the function call would consequently be marked as *INVALID*.

LISTING 5.9: Use case 3 - Test validator on meuse data

```
> #test validator against meuse data set
> library(sp)
> demo(meuse,echo = FALSE, ask = FALSE)
> functionalType(meuse) <- "SField"
> params = list(formula = zinc ~1, graft = 0.13, level.max = 7, min.parent = 25,
      min.child = 2, rtwo.min = 0.01)
```

```
> meuse.results = spodt.malaria(params,meuse)
Warnmeldungen:
1: In validator(args, output, semantics, call_semantics) :
  Spatial partitioning with the given input data was not intended.
Expected were SMarkedEventData, but given were SFieldData.
2: In spodt.malaria(params, meuse) :
  Post-validation of function call failed!
```

Chapter 6

Evaluation and results

This chapter evaluates the findings of this work in respect of the research objectives and their pre-defined requirements (compare with section 1.3): In compliance with the first objective, the proposed R package is a framework for the semantic provenance of spatio-temporal data analysis and implements the algebraic model by Scheider et al. (2016) (section 6.1). Matching the the second objective, this work fosters meaningful communication of spatio-temporal statistics (section 6.2. According to the third objective, a selection of three use cases of common analytical problems of Spatial Statistics puts the R package into action and demonstrates the usefulness of the implemented prototype and the underlying scientific approach (section 6.3).

This Chapter addresses the forth research objective (evaluation), which is fulfilled by finally drawing meaningful conclusions to the research questions from the research results (Chapter 8).

6.1 Implementation

The 'SpatialSemantics'-package for R is a domain-aware application for Spatial Statistics because it makes use of the algebraic model by Scheider et al. (2016) and thus describes spatio-temporal data and -procedures by of types of reference systems.

Like most R extensions, the package is easy to install, using just a few lines of R code (4.1). The package manual includes basic descriptions of each R function exposed to the user. Furthermore, the Chapter 4 contains a throughout documentation of the package's capabilities and usage.

Although the package is still in a prototypical state and did not yet undergo usability studies or a regular testing and review process, it already performs with much stability. It works together with many reproducible code examples that are presented herein, notably the three complete R scripts that perform spatio-temporal data analyses according to the use cases in Chapter 5 (see Appendices C, D, and E). The R package includes more runnable scripts in the demo folder.

Using function calls corresponding to the package's provenance functionality (section 4.2.1), a user can enable and disable provenance tracking during an R session. While provenance tracking is enabled, task callbacks perform the provenance recording invisibly in the background. The package engine generates a spatio-temporal derivation graph from all expressions that are entered in the R command line and the coincident runtime-state of the R environment. The graph is stored in the internal package environment and enhanced whenever a callback records new provenance data. Users can retrieve and export the graph using the Graphviz-format

dot/gv, which can also be converted into common data formats like pdf, png, jpg and svg. Dot-graphs can be visualized and visually analyzed in various GUI-applications. (Section 3.7)

The section 3.6 of this thesis explains the approach of handling semantics. Section 4.2 explains the corresponding semantic functionality implemented in R: The meaning of an analysis is determined automatically as far as possible. With the help of heuristics and semantic inference, data derivation graphs are populated with semantic annotations, even if no or little semantic information is explicit. On the other hand, the framework allows the user to define missing information manually. Section 5.1 exemplifies the interplay between heuristics and semantic inference with user-defined semantics in the context of the first use case.

Appendix A shows a heuristic mapping from data properties, such as the data type, to semantic representations of data (object semantics). It is important to note that this mapping is not definite; data types in reality correspond to many possible semantic representations and vice versa. The mapping tries to estimate amongst all semantic expressions that *possibly* apply that expression which is most *likely* to apply. The mapping could be regarded as an automated *content assist* for semantic annotations. Determining semantics is further automated by inferring the implicit signature of a function call or of an expression during runtime. This signature is estimated from the semantics of inputs and outputs. Another mapping applies semantic inference when a user assigns functional types (B) to spatio-temporal data: If the spatio-temporal generation type of a dataset is known, the semantic representation of the dataset itself is straightforward. It also depends on an unknown data generator, though. Nevertheless, this gap of knowledge is marginal within the context of this work: All spatio-temporal dataset associated with the use cases directly relate to just two data generators from the algebra: the *map*-function and the *datagen*-function.

The package's semantics-engine prompts semantics-related messages during an R session when appropriate. In particular, it prompts warnings if a function call is semantically inconsistent or invalid (i.e. if the call did not pass the pre-defined checks), as well as if relevant semantic information (i.e. annotations) of a spatio-temporal dataset is missing.

At any time during an R session, users can interactively query for the semantics of any resource in the user's workspace. The four basic queries refer to: (1) the semantic representation of an object, (2) the semantic pedigree of an object, (3) the functional type of an object, and (4) the default call semantics of a function. If necessary, the user can define or refine all this information manually by setting object attributes or using the corresponding setter-functions (section 4.2.2). Furthermore, a user can query changes of a variable binding (version history). The thus retrieved information also reflect on the semantics of the data that are bound by a variable during one stage of the execution (section 4.2.1).

The findings described in this section suggest conformity of this work with research objective 1 and its requirements (*Design and implement a framework for the semantic provenance of spatio-temporal analysis*). Therefore, the particular research goal is considered fulfilled.

The current implementation still lacks of data formats for exporting and sharing semantic provenance, though. Although the Graphviz-format dot/gv is a mighty format for graph visualization and subsequent visual evaluation of the same, it provides no means to store provenance records and semantics in an organized way. This fact will be discussed in Chapter 7.

6.2 Meaningful communication

The semantic attribution of data and objects may appear confusing at first glance but regarding meaningful communication it is straightforward:

The 'semantics'-attribute of an object (i.e. *object semantics*) defines its semantic representation (as it appears in the data derivation graph). In other words, the 'semantics'-attribute describes *what* an object or a resource represents. For instance, a list of dates can represent either a set of moments in time ($T\ set$), or a set of time intervals ($I\ set$). In the former case, the dates are interpreted as distinct moments, whereas in the latter case the dates are interpreted as beginnings and endings of time intervals. This example illustrates that semantics are not always implied by how data is organized. Semantic annotations are thus means to communicate the implicit modeling assumptions of resources.

Object semantics or the 'semantics'-attribute do not always relate to basic types and their derivations (referents); some resources may also represent data generators and generation functions. Furthermore, it is not always clear that resources that are *technically* objects or datasets also represent a referent in terms of the algebraic model; the third use case analysis (5.3) incorporated a decision tree from which data of an inverted spatial field can be generated. The decision tree is represented by data, but the data represents a scientific model, in particular, a spatio-temporal generation function. The tree is labeled accordingly as an $SInvField$. The R prototype allows to communicate the meaning of such resources and scientific models by using object semantics as an attribute.

The *semantic pedigree* of an object is a record of all semantic procedures that are directly involved in the creation of the object or resource. Hence, a user that queries semantic pedigree receives a list of generation types and functions (with their estimated signature) that generated or modified the dataset either as a whole or parts of it (some attribute). The semantic pedigree is a resource attribute that describes in an abstract way *how* the resource was created.

This work uses the term 'functional type' for an attribute that corresponds to spatio-temporal datasets. It refers to the spatio-temporal generation function that was involved in the creation of the resource, but it has a prominent meaning amongst all other procedures in a resource's semantic pedigree. The functional type denotes that generation function that rendered the data into its current semantic representation (strictly speaking, in conjunction with a data generator - see Appendix B). Moreover, the functional type stands for the hypothetical process that generates the observed data. Because spatio-temporal generation types are well defined in terms of reference systems, the functional type allows relating the data back to the real-world phenomenon it describes, for instance, events, fields, objects or

trajectories. It also links the data to research assumptions of observations, because the type definition describes how the phenomenon is modeled. For instance, if an event is modeled by a mapping from a discrete identifier to a location, the underlying modeling assumptions are communicated by the definition $Event :: D \Rightarrow S \times T$. The functional type of data is incorporated in the data derivation graphs as a prefix to the semantic representation (object semantics) of a dataset. For instance, a collection of spatial locations and quality values that was generated from a spatial field is labeled $SFieldData : S \times Q\,set$). In conclusion, the 'functional type' attribute relates spatio-temporal data to its *purpose*. It explains *why* data is generated.

R functions (if they are wrapped) can be attributed with default call semantics, a validator, and a postprocessor. While the postprocessor basically serves the automation of annotating semantics, the former two have direct impact on meaningful communication: The default call semantics hint to the *purpose* of a function. It allows the user to specify the expected semantics of a function's signature by a list of permitted signatures. For instance, the second use case analysis (section 5.2) makes use of the geometry-function of the sp-package (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013). This function receives a spatial data frame as an input and returns only the spatial components. Therefore, the default call semantics may be for instance $R \times Q \Rightarrow R$ or $S \times D \times Q\,set \Rightarrow S\,set$. A user can specify these semantics per function call or per function in order to express what he intents the function to do. Hence, the default call semantics refer to a function's *purpose* in the context of an analysis. If the *de facto* signature that is estimated during runtime does not comply with the default semantics, the call is be marked and thus communicated as an inconsistent regarding the user's or developer's *intentions* and how the analysis is carried out in practice.

Because a functional type communicates the purpose of a spatio-temporal dataset, it can also be inferred what *literally* is its function, i.e. what can be done with it. For instance, it is known that interpolation of spatial field data is meaningful, as well as the a aggregation of point-patterns (i.e. events or objects) (E. J. Pebesma and C. Stasch, 2014; C. Stasch, Scheider, et al., 2014). A function's validator hence provides means to implement ad hoc checks for *meaningfulness* of an operation, as this is exemplified by the third use case (section 5.3). Such a check is realized by letting the validator retrieve the functional type of the function's inputs and matching it with the required type. In case of a mismatch, the function call is marked as invalid and thus communicated as *not meaningful*.

In conclusion, a function's call semantics reflect on its purpose purpose and alows to match its *de facto* semantics with the user's intentions. A function's validator allows users to check whether an operation that is carried out on spatio-temporal data is *meaningful* in the sense that it complies with the data's *purpose*, which is communicated by its functional type (compare with E. J. Pebesma and C. Stasch (2014)).

A user can be notified about missing semantics and assumptions about observations and the underlying hypothetical processes in two ways: The mapping in Appendix A estimates not only the semantic representation of an object based on heuristics, it also determines whether the user should be warned if semantic annotations of a certain data set are missing. This is mechanism is demonstrated in the first use case (section 5.1) (section 5.3). A

user is not only notified if an input's functional type does not comply with the requirements but also if the type is missing.

All these information is reflected in spatio-temporal data derivation graphs as well information directly prompted to the user or which a user can interactively query. The data derivation graphs incorporate the basic graph syntax proposed by Scheider et al. (2016). This syntax is enhanced by additional syntactical elements (compare with the sections 3.5.3 and 4.2.3) in order to communicate provenance-related specifics that otherwise could not be communicated, for instance, the names of variables that bound data or functions. Such information allow the user to retrace how the analysis is carried out and to relate semantic information back to the underlying R script and expressions as well as to technical particulars of the execution.

Users can be selective about the information they want to communicate, because provenance tracking can be flexibly enabled and disabled when an analysis is executed. All three use cases in Chapter 5 demonstrate how this can be done: The depicted data derivation graphs display only that part of the associated execution that is the core of the analysis, i.e. data generation and transformation procedures. Particulars that are not of direct interest for comprehending these procedures and thus would only complicate the graph are purposely excluded from the visualization - for instance, data preprocessing and calls to plot-functions.

The findings of this section suggest conformity of this work with research objective 2 and its requirements (*Enable meaningful communication of spatio-temporal analysis*). Several approaches are described for communicating the meaning and purpose of datasets, scientific models and function calls that form part of an analysis. Furthermore, research assumptions on the same components can be communicated and checks for validity and consistency applied. Users are notified by warnings if critical semantic meta-information or research assumptions are missing. It is possible to relate back semantic information from a data derivation graph to technical particulars of how the analysis is carried out and users can be selective about the information to be communicated by deciding which part of an analysis is tracked.

In some aspects, the current solution compromises between pragmatism and formalism in regards of the algebraic model. This is because some aspects of an analysis, notably function calls and their internals, cannot be related to formal definitions by the model yet. This will be discussed in the following chapter 7.

6.3 Demonstration

This work demonstrates the prototypical R package on three use case in Chapter 5. Each use-case is different from another regarding the data that is analysed, and the analytical problem that is addressed. Hence the approach is demonstrated under variable preconditions, which shows the versatility of the approach. General applicability of the prototype is self-evident from the use case chapter: For each use case, the meaning and function of the analysis is explained first, together with the semantics that

should be applied for the therein explained reasons (overview, approach). The implementation section that is common for all use cases shows that a spatio-temporal derivation graph can be generated that incorporates this exact meaning end semantics. The graph is visualized and explained by a detailed walk-through of how it is generated. The graphs can also be reproduced using the R-scripts from the appendices C, D, E.

Each use case also highlights different aspects and advantages of the implemented prototype, which are summarized in the following:

Spatio-temporal aggregation of bird counts

The first use case (section 5.1) shows that the R prototype can be applied to a spatio-temporal point pattern analysis, in which point observations that refer to events (bird counts per year and observation location) are aggregated over two-year periods and a spatial region (the boundaries of Florida). The aggregation results in a time series. Implicitly, the analysis carries out a derivation of a temporal lattice $TLattice$ from marked events ($MarkedEvent$), which manifests itself in the input and output data, in the graph labeled with $MarkedEventData$ (bird count data) and $TLatticeData$ (the aggregated bird counts). It is found that this derivation process can be described in terms of it in-and outputs and in terms of provenance, but the derivation itself still lacks a formal definition from the algebraic model.

It is found that the assignment of a functional type to the bird-count is not definite, because the data can be interpreted in two different ways based on the given knowledge: On the one hand, it can be interpreted bird counts are a marked interval-event $D \Rightarrow S \times I \times Q$, whereas the bird counts are the aggregated observations over the period of one year. On the other hand, the functional type could be a regular marked event, if the year date associated with the counts represents a moment in time (specified by limited accuracy, though). The latter interpretation was favoured for the use case.

The implementation walk-through highlights that a spatio-temporal derivation graph can be populated with heuristic semantic information even if no user-defined annotations are provided and that warnings are prompted in case some of this missing annotations is crucial to the analysis.

Spatial prediction on the meuse river floodplain

The second use case (section 5.2) shows that the R prototype can be applied to the analysis of point observations sampled from a continuous spatial field. It also exemplifies spatial prediction in this context. The analysis does not incorporate a derivation procedure, as input data and generated data have the same functional type, an $SField$. The core of this analysis is a variogram-model that is enclosed by the interpolator-function. This variogram models the assumption of the spatial field that generated the point observations that are the input of the analysis. That is why it is possible to estimate unknown zinc concentrations flood-plain. The interpolator-function generates spatial field data from this model for given locations of interest.

It was found that modeling the exact behaviour of the interpolator-function is difficult based on the existing framework, because the function internals consists of a spatio-temporal generation function (the variogram

model), from which spatial field data *can* be generated, and a data generator that takes locations of interests and generates the data from the model. This could problem could be resolved, however, by either modelling the functions' internals as a sub-graph, or by rewriting the R script in such a way that the variogram model stands for itself. Note that the internal variogram model *technically* is not a function in R, but an object. Regarding semantics, nevertheless, it fulfills the formal criteria of a spatio-temporal generation type.

The interpolated zinc data is annotated accordingly with an $SField$ as a functional type. And although a user cannot exactly pinpoint that the underlying modeling assumptions have origin from the variogram model enclosed by the interpolator, it becomes apparent form the derivation graph and semantic pedigree which function call generated the data. It suggest itself that this function call somehow must incorporate a model, even this model unfortunately is not made explicit in this case.

The implementation walk-through highlights in that function calls are checked for semantic consistency: It is shown that the user is warned and the interpolation graph inconsistent regarding the call of the interpolator if the *de facto* semantics of the call, which are estimated during runtime, do not comply with the pre-defined default call semantics. It is also demonstrated that users can be selective about the information that is communicated: While the first graph is too complex and too large to be displayed in this thesis in full detail, the second graph is reduced to the essential part of the analysis and thus communicates the meaning and purpose of the use case analysis much more clearly to the reader of this thesis. Note that this reduction is not done by posterior editing of the graph, but by pre-defining at which part of the analysis provenance shall be recorded.

Clustering malaria episodes in Bandiagara, Mali

The third use case once again demonstrates the prototype on a point pattern (malaria episodes per child per household), but in this case in this case it deals with a spatial point pattern of events that occurred over a fixed time period. The analysis in this case applies a spatial partitioning algorithm that estimates areas to which different risks (or risk ranges) of malaria incidents apply. The underlying data derivation procedure manifests itself in the spodt-function that derives an inverted spatial field, manifested in the generated spatial opaque decision tree, from marked event locations. Similar to the first use case, this derivation process lacks a formal definition from the algebraic model. Similarly, the function spodtSpatialLines generates spatial regions from the decision tree and the given dataset of malaria-incidents, but it is not clear how this is done internally, as the data generator does not work as straightforward as the map- or the datagen-functions from the algebra.

The use case implementation furthermore highlights how postprocessors can automate the annotation of datasets during an execution and how validators implement ad hoc checks for meaningfulness of a function call. The validator is applied to the spodt-function that generates the spatial opaque decision tree. In a negative-demonstration on spatial field data, it is shown that the validator warns that this is not the intended usage of the function. It would also warn if the functional type of the input, which

stand for crucial research assumptions necessary to asses meaningfulness, are missing. The exact specifics of such a validator would have to be defined by domain-experts that implement analytical functions. Main purpose of the demonstration is to show that such specifications are generally enabled by the framework.

The use case also exemplifies that complete lists of parameters that are applied to a function can be displayed *as is* in a spatio-temporal derivation graph using the existing framework. Although this 'effect' is enforced by customizing the R-script, it is an outcome that should be expanded on in further developments.

In conclusion, the findings of this section suggest compliance with the third research objective and its requirements (*Demonstrate the solution on common analytical problems of Spatial Statistics*). The use cases are diverse and replicate analytical approaches that were previously published. It can be seen that many aspects of meaning and purpose become apparent using the proposed framework.

Nevertheless, the evaluation of the use cases suggest that there is still room for improvement: Function calls should to be related more closely procedures formally defined by the algebra. Possibly it would be beneficial to also model function internals using sub-graphs. However, formal definitions of the corresponding derivation procedures are still missing, for instance, how a spatial inverted field derives from marked event locations. Such an expansion of the existing algebra, in turn, is not within the scope of this thesis.

Chapter 7

Discussion

This work handles two significant challenges at a time: a provenance-challenge and a semantic challenge. The provenance-challenge arises from the necessity to implement the algebraic model by Scheider et al. (2016) in R: Current provenance trackers do not provide means to incorporate semantics and they could not be used to generate those spatio-temporal data derivation graphs shown in this work without a considerable amount of modification. The herein presented provenance-tracker is novel to R and has not been implemented yet in this particular way.

The semantic challenge arises from the fact that this work attempts to implement a semantic model that has never been implemented before and tries to establish a meaningful relation between the abstract concepts to technical particulars of R, i.e. expressions, functions, variable bindings and more. The research in this area is far from being complete. In order to provide useful intermediate results, this work compromises between pragmatism and formalism regarding these relations.

These considerations are discussed in the following sections.

7.1 Capturing provenance in R

Provenance tracking in R is a big challenge because of the complexity of the language side-effects that are hard to detect. Abstracting over R language requires a deep understanding of the syntax and meaning of R expressions, as well as how R parses them within the read-eval-print loop (R Core Team, 2000) in order to do *computing on the language*. Therefore, the 'SpatialSemantics'-package fundamentally relies on non-standard evaluation (Wickham, 2014, p. 259): the quote- and substitute-functions in R allow capturing any expression that is executed in R, to deparse it and to access its parse tree. The parse-function allows to create custom expressions from strings and the eval- and do.call functions enables the execution of such custom calls.

The herein presented data derivation graph provide an abstract view of function calls and data that results from deparsing, sub-dividing and converting complex R expressions in which multiple function calls and variables are nestled. This shall be illustrated by a code line from the script in Appendix D. The abstract view of this code is reflected in Figure 5.6

```
1 interpolator = getInterpolator(modelSemivariogram(zincPointData),
    zincPointData)
```

This expression assigns the output from a call to 'getInterpolator' to a variable 'interpolator'. The call to 'getInterpolator' is evaluated after

evaluation of the expression 'modelSemivariogram(zincPointData)'. The 'getInterpolator'-function has two input arguments 'params' (first argument) and 'pointdata' (second), but the argument names are not visible from this expression. The argument list can be retrieved using the formals-function (R Core Team, 2016, p. 201) (Wickham, 2014, p. 71):

```
> formals(getInterpolator)
$params


$pointData


$semantics
[1] NA
```

This call reveals not only the parameter names but also default values if any. Obviously, the getInterpolator-function has a third argument with a default value that is not apparent from the code line above. The 'SpatialSemantics'-package internally standardizes function calls using the match.call function (R Core Team, 2016, p. 309) as the following:

```
> match.call(getInterpolator, quote(getInterpolator(modelSemivariogram(
    zincPointData), zincPointData)))
getInterpolator(params = modelSemivariogram(zincPointData), pointData =
    zincPointData)
```

Primitive function in R cannot be standardized in this way. The functions 'match.call' and 'formals' do not support it. Also, R allows placing a special argument called . . . (Wickham, 2014, p. 88) that matches any arguments not matched, i.e. it is a wildcard for any unspecified list of arguments.

Knowing a function's arguments, on the other hand, is essential for the construction of semantic wrappers. The current implementation of the *captureSemantics*-function attempts to create a generic function proxy that mimics the behavior of the wrapped function. But this generic wrapping fails, if no definite argument list can be retrieved. Therefore it is necessary in some cases to manually define a function that has a definite argument list and thus calls the function to be wrapped. This function can then be applied to the 'captureSemantics'-function that creates another wrapper with semantic capabilities (compare with use case 1 in section 5.1).

There are many ways to assign variables and manipulate objects in R, which all have to be captured by the provenance tracker in order to provide an accurate record. R defines the assign-function and the assignment operators =, <-,≪-, -> and -≫. (R Core Team, 2016, p. 35-38). The prototype evaluates these assignment operators and accordingly defines nodes and edges of the derivation graph. However, the current implementation only recognizes a limited number of such expressions, which of course can be amended in further developments. However, some of these assignments are hard to track. First of all, because R provides means to create use-defined assignment operators. Second, because assignments can be done from and to different environments (Wickham, 2014, p. 123), so that for instance, a variable in the users' workspace can be manipulated from inside a function using the ≪--operator or assign-function.

Third, assignments can also be done using so-called non-standard evaluation, which is almost impossible to capture from code analysis.

Finally, assignments can be done in R using replacement-functions (Wickham, 2014, p. 91). Replacement functions are acting like they are directly modifying an attribute or property of a given in place, but in reality they first modify a temporary copy of the object and then replace the object by this copy. An example is the *coordinates*-function from the sp-package (R. S. Bivand, E. Pebesma, and Gómez-Rubio, 2013), but also the *captureSemantics* and *functionalType*-functions from the 'SpatialSemantics'-package are replacement functions. A typical call to the coordinate-function looks as follows:

```
coordinates(meuse) <- c("x", "y")
```

The 'SpatialSemantics' package internally standardizes such calls so that they can be parsed like any other expression (which is also how the R-parser works internally):

```
> rewriteReplacementFunction(quote(coordinates(meuse) <- c("x", "y")))
meuse <- `coordinates<-`(meuse, c("x", "y"))
```

Some assignments that are side-effects of function calls can be retraced using the in this thesis proposed detection mechanisms (sections 3.5.2 and 4.2.1). Also, the approach is taken to only record provenance from the user's workspace, which narrows down the range of assignment operations to be detected.

The main alternative to track provenance in R is the RDataTracker by Lerner and Boose (2014). Figure 7.1 shows a data derivation graph (DDG) as presented in the paper. It can be seen that the taken approach spares at least some of the language computing: The underlying script only consists of simple expression without nesting of operations (which is not common for R scripts). Those expressions are not itemized but kept as they are. In this way, the provenance of an R session is represented as a directed graph of sequential expressions and data that is manipulated or read.

However, such an approach is not applicable to the algebraic model, because an infinite number of R expression can be formed, but the number of functions in R is finite. Hence, relating functions and function calls to semantic procedures is a more a more feasible task than assigning semantics to complete expressions. Besides, the herein proposed approach supplies the user with more detailed information about how an analysis is carried out (by itemizing expressions). A compromise would be to require R scripts to have one function call per line, but this would mean large burden on the user's side.

Technically, the RDataTracker is different from the 'SpatialSemantics' package in that way that the former only evaluates the runtime state of R when a function from the library is called; the 'SpatialSemantics'-package evaluates (provided that tracking is activated) the R runtime state continuously in the background after each top-level task (command) executed from the R command line.

The works of Silles (2014) on a provenance-aware CXXR provides provenance capabilities for the CXXR interpreter of the R language, but no means to compute a data derivation graphs comparable with the RDataTracker or SpatialSemantics. Besides, there currently exist no stable release that is

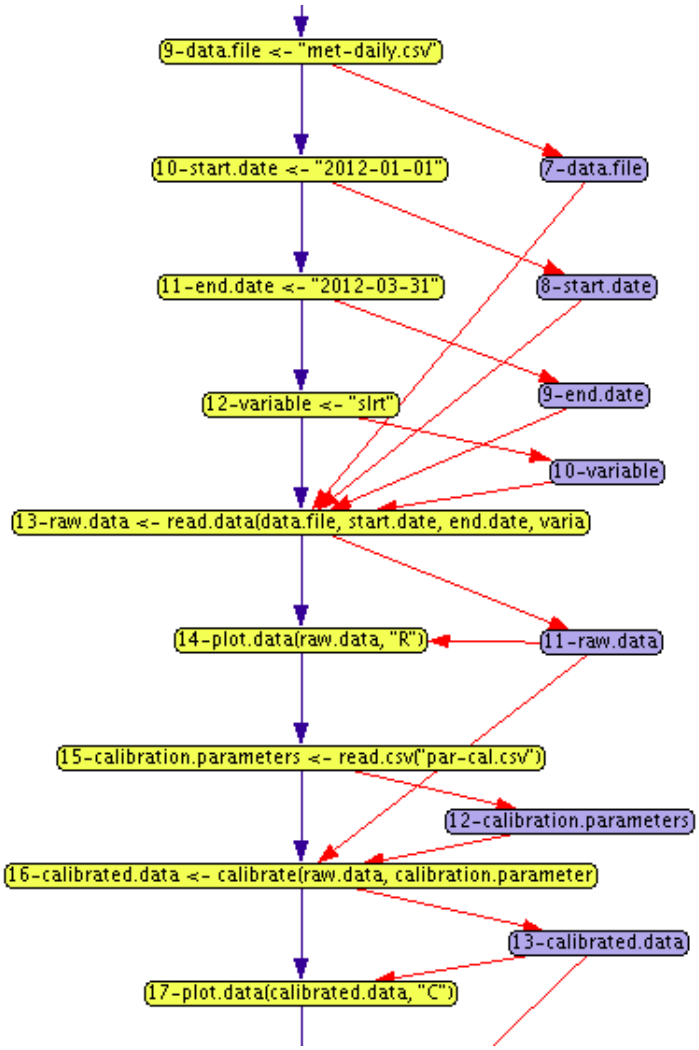


FIGURE 7.1: Data Derivation Graph (DDG) presented by Lerner and Boose (2014)

compatible with the R-packages used in this work for spatio-temporal analysis. Neither Silles (2014) nor Lerner and Boose (2014) considered incorporating domain-specific semantics in their frameworks. In this aspect, the approach taken by this work is unparalleled by existing solutions in R. In conclusion, the complexity of R and the lack of pre-existing provenance tracking solutions are constricting the implementation of the 'SpatialSemantic'-package. But this work shows also that the associated challenges can be resolved and that doing so yields much innovative potential.

7.2 Conceptual challenges regarding semantics

A general constraint to the 'SpatialSemantics' framework is that particulars of an analysis cannot be related to formal definitions from the algebraic model. While all spatio-temporal datasets can be defined in terms of reference systems, this is not the case for function calls: The calls are informally described by the name of the function or the expression that was used and

a 'semantic signature' that specifies its in- and outputs. However, in order to assess semantics to its full scope, it would be necessary to define these calls by types of reference systems and primitive functions from the algebra. The same applies to the underlying derivation processes as in the third use-case 5.3: A formal definition of how a spatial inverted field derives from event locations does not exist (nor can it be computed at the moment). Nevertheless, these constraints do not mean that the research goals are not fulfilled because research in first place seeks to implement a framework for the existing model by (Scheider et al., 2016), not to expand on the model with additional definitions. Besides, the framework provides with the concept of 'semantic pedigree' means to capture those formal definitions, independent of how the analysis is carried out technically: The currently the pedigree record is filled with both, formal spatio-temporal derivation types and *informally* defined functions that are involved in the creation of an object. If the corresponding definitions were provided, the record could be filled with the *formal* semantic procedures used underlying the functions internals and all preceding operations from the algebra leading up to the creation of a resource. The definitions could be automatically annotated by the function's postprocessors. The result would be a record of all semantic procedures leading up to the creation of a research, from which, in turn, a spatio-temporal derivation graph could be computed that has a *higher level of abstraction*: While the herein presented derivation graphs often compromise between technical constraints and semantics, those derivation graphs would display semantics independent from (technical) provenance. These graphs, on the other hand, would be difficult to be related back to how the analysis is carried out, i.e. to the source code, variables, function parameters and more. An alternative would be to model function call semantics as a sub-graph of the node that refers to the function call. This would be the most comprehensive solution in terms of provenance *and* semantics.

In this work, the approach is taken to assign semantic annotations to *all* data present in the analysis. In some cases, it is questionable if some data should be annotated at all. The algebra includes no guidelines for this, though: Literal function arguments are generally mapped to quality values, according to the heuristic mapping (A), but some of these arguments do not represent any kind of observation, but sometimes arbitrary parameters that influence the functions *behavior*, for instance, if messages should be printed or not. Such parameters do not represent referents nor any other defined part of the algebra. Whether and how such particulars of an analysis should be annotated has to be assessed by domain experts. The proposed framework, however, has the capability to assign any kind of semantic type to any part of the analysis. The decision about such scientific particulars is up to the user.

7.3 Further works

The heuristic mapping in Appendix A is not a definite mapping from data types to semantics, but an estimation of the semantics that is considered *likely*. This mapping would have to be extended and refined in future developments. Specifics like when and how a user should be notified about automated estimations also need to be determined. In future developments, the

mapping should also be made customizable by the user, while currently, it is hard-coded within the package. The mapping can be seen as an automated *content assist* for users, and this work proposes to expand on this idea: By assessing not only all semantics that are *probable* but narrowing down the range of annotations that are *possible* regarding data structures, the mapping could become the basis of an interactive content-assist application for spatial semantics. While recording provenance, the user would be notified about missing semantics of a certain resource. From a commented list of suggested annotations, the user would then select the type that applies to the data. The application would then automatically refine the provenance record, the R script, and also annotate the data.

This thesis exemplifies the scientific approach on a package for R, but it takes the stand that the same approach could be expanded on scripting languages like Python. Python scripts can be used for Spatial Statistics as well - the Python extension Arcpy, for instance, enables automated mapping and analyses in ArcGIS (Zandbergen, 2013). The algebraic model of Scheider et al. (2016) may also be implemented in workflow management systems like Taverna (Alper et al., 2013) or the ArcGIS model builder (Allen, 2011). Those solutions could then be compared with the implementation in R and thus a refinement of the overall approach could be undertaken.

The current implementation is restricted by the lack of a data model that stores provenance and semantics in an organized way. Such a data model would ideally implement the web standards recommended by the W3C, like the Resource Description Framework (RDF) (Klyne and Carroll, 2004) and the PROV model for provenance representation (Missier, Belhajjame, and Cheney, 2013; Gil, Miles, et al., 2013). Such a data model would require the development of a compatible vocabulary or an ontology that captures the semantic provenance of the herein presented data derivation graph. It suggests itself to comply with the W3C Web Ontology Language (OWL) (McGuinness and Van Harmelen, 2004) for this purpose. This extension of the presented framework would enable publishing spatio-temporal derivation graphs on the web, according to the principles of Linked Open Data (LOD). It furthermore would enable advanced queries over those graphs using the SPARQL query language (Group, 2013). Such queries could be used for example to query the derivation of intermediate results from an analysis and to thus to extract subgraphs from a larger provenance graph.

Furthermore, it may be considered to embed the prototype in a larger provenance system such as Karma or its successor tool Komadu (Suriarachchi, Zhou, and Plale, 2015). These systems already provide data models, storage and reasoning capabilities, web services and user interfaces that facilitate capturing provenance and semantics. Komadu also supports the current W3C PROV standard. According to the authors, Komadu can collect provenance information from any application. It should be assessed if the API and implementation of this tool are flexible and enough to capture and represent spatio-temporal derivation graphs.

Chapter 8

Conclusions

The following paragraphs derive meaningful conclusions from this work to the research question 1, 2, 3 and 4 in Chapter 1.

The main contribution of this work is a specification of guidelines for communicating meaning and purpose of spatio-temporal data analysis within an environment for Spatial Statistics. The conceptual basis of this specification is the existing model for spatio-temporal data generation and derivation by Scheider et al. (2016). The approach is exemplified by a prototypical extension for R but it can be expanded on any software environment that utilizes spatio-temporal analysis as a tool.

The requirements for implementing the model are provenance recording capabilities in the target environment. This thesis also specifies how these capabilities recording can be implemented for functional scripting languages that rely on a read-eval-print loop similar to R. The main approach is to implement handlers that are executed in the background and record provenance every time a new command is executed. Provenance information is obtained from evaluating the runtime state of the software environment and from analyzing and itemizing the executed command. The provenance recording tool builds up a generic data derivation graph of the executed analysis that can be populated with semantic annotations in compliance with the algebra. The provenance metadata is persistently stored in the background during the analysis and won't be deleted or overwritten unless the user does so intentionally. The user can retrieve and export the graph in an appropriate data format (in this work exemplified by the dot/gv format) that allows the graph to be visualized, reviewed and shared with third parties. The assignment of semantic annotations can be automated using post-processor functions that annotate the output of functions during runtime. Semantic inference and as heuristics provide means to estimate semantics that are not made explicit by the user.

Based on these capabilities, research assumptions on the analysis can be made explicit by assigning semantic annotations to the provenance record: The semantics-attribute (object or resource semantics) expresses what a resource represents in terms of the model. This could be either a referent, a generation type or a data generator. While observed data corresponds to referents, scientific models or function from which data can be generated are generation types. It thus can be inferred what a resource represents and what it is meant to express. Spatio-temporal datasets are also attributed with a *functional type* that communicates the *purpose* of the data and assumptions about the hypothetical process that generates the observed data. The functional type relates a data set a spatio-temporal generation type that

defines the observed phenomenon (e.g. object, field, trajectory). The generation type also defines how this phenomenon is modeled by a generation function. Thus it can be inferred how the semantics of the dataset (object / resource semantics) are rendered in conjunction with a data generator.

The purpose of a function or function call can be communicated by assigning default call semantics to the function that expresses how the function is supposed to behave regarding the semantics of its inputs and outputs.

The meaningfulness of a data generation or derivation operation can be checked by assessing the functional type of the input data, because knowing the purpose of data, i.e. why data has been generated, the function of data can be inferred, i.e. what can be done with the data. It is proposed to implement meaningfulness checks by a validator function that is executed after a function call and has the functions inputs, outputs, and semantics as arguments. Domain experts could define these validators that are attached to the functions and thus define requirements on which inputs are permitted for the function call. The validator could also check the output in terms of if the results do also have the expected semantics. The validator would throw warnings or errors in case that the operation is not meaningful. This would then be recorded and reflected by the data derivation graph.

The communication between the environment for Spatial Statistics could be enhanced by letting the user interactively query particular aspects of the stored semantics and by letting the system print semantics-related messages, warnings, and errors on the user-interface.

The communication from the statistician to fellow researchers finally can be improved by sharing the spatio-temporal derivation graph and using it as an illustration for explaining how an analysis was conducted and why it was conducted in this way.

Appendix A

Heuristic mapping of semantic types

TABLE A.1: The heuristic mapping for estimating unknown of semantic types, used by the 'SpatialSemantics'-package

Classes in R	**Semantic type**	**Warning**
numeric, character, factor, symbol, name, expression (base-package)	Q (for atomic vectors) $Q\ set$ (if length > 1)	No
logical(base-package)	$bool$ (for atomic vectors) $bool\ set$ (if length > 1)	No
SField[1] (mss package)	$'a \times SExtent$[2]	Depends on 'a
SpatialLines, SpatialPixels, SpatialPoints, SpatialMultiPoints (sp-package)	S or $S\ set$	No
SpatialLinesDataFrame, SpatialPixelsDataFrame, SpatialPointsDataFrame, SpatialMultiPointsDataFrame, SpatialGridDataFrame (sp-package)	$(?)S \times Q\ set$	Yes
SpatialGrid, SpatialPolygons (sp-package)	R or $R\ set$	No
SpatialPolygonsDataFrame (sp-package)	$(?)R \times Q\ set$	Yes
Spatial (sp-package, includes unknown subclasses)	$(?)S\ set$	Yes
Date, POSIXlt, POSIXct (base-package), xts (xts-package)	T or $T\ set$	NO
STF (spacetime-package)	$'a \times\ 'b$	Depends on 'a (time-slot) and 'b (sp-slot)
STFDF (spacetime-package)	$(?)S \times T \times Q\ set$	Yes
STI (spacetime-package)	$(?)S \times T\ set$	Yes
STIDF (spacetime-package)	$(?)S \times T \times Q\ set$	Yes
STS (spacetime-package)	$(?)S \times T\ set$	Yes
STSDF (spacetime-package)	$(?)S \times T \times Q\ set$	Yes
(other classes)	*(?)Class:<class name>*	No

Appendix B

Functional type mapping

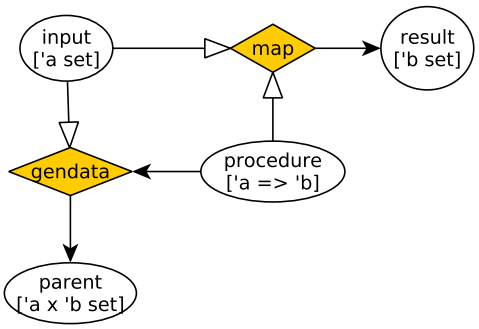


FIGURE B.1: The Data derivation graph underlying the functional type mapping: result semantics are estimated by mapping the generation procedure the inputs on the generation procedure by *map* result semantics originate from applying inputs and procedure to *datagen*. *Map* and *datagen* are both data generators from the algebra.

TABLE B.1: Functional type mapping - If the procedure name is known, the following semantics and data generations can be inferred based on Figure B.1 (Listing is not exhaustive)

Provedure Name	**Procedure definition**	**Result semantics**	**Parent semantics**
Field	$S \times T \Rightarrow Q$	$Q\ set$	$S \times T \times Q\ set$
TField	$T \Rightarrow Q$	$Q\ set$	$T \times Q\ set$
SField	$S \Rightarrow Q$	$Q\ set$	$S \times Q\ set$
InvField	$Q \Rightarrow Occurs$	$Occurs\ set$	$Q \times Occurs\ set$
SInvField	$Q \Rightarrow R$	$R\ set$	$Q \times R\ set$
TInvField	$Q \Rightarrow T$	$T\ set$	$Q \times T\ set$
Lattice	$R \Rightarrow I \Rightarrow Q$	$Q\ set$	$R \times I \times Q\ set$
SLattice	$R \Rightarrow Q$	$Q\ set$	$R \times Q\ set$
TLattice	$I \Rightarrow Q$	$Q\ set$	$I \times Q\ set$
Event	$D \Rightarrow S \times T$	$D \times S \times T\ set$	$D \times S \times T\ set$
MarkedEvent	$D \Rightarrow S \times T \times Q$	$S \times T \times Q\ set$	$D \times S \times T \times Q\ set$
SMarkedEvent	$D \Rightarrow S \times Q$	$S \times Q\ set$	$D \times S \times Q\ set$
MarkedTrajectory	$T \Rightarrow S \times Q$	$S \times Q\ set$	$T \times S \times Q\ set$
MarkedObjects	$D \Rightarrow T \Rightarrow S \times Q$	$S \times Q\ set$	$D \times T \times S \times Q\ set$

Appendix C

Space-time aggregation - R script

LISTING C.1: Space-time aggregation - complete R script

```
library(xts)
library(sp)
library(spacetime)
library(maps)
library(maptools)
library(SpatialSemantics)
library(Rgraphviz)

## Data preparation
## download "ECDovelatlon.dat" and ECDoveBBS1986_2003.dat from

#read and prepare observation locations as spatial points:
ecd.locations <- as.matrix(read.table("ECDovelatlon.dat", header = FALSE))
ecd.locations <- SpatialPoints(ecd.locations[,c(2,1)])
proj4string(ecd.locations) <- CRS("+proj=longlat +datum=WGS84")
#observation years (1986-2003):
ecd.years <- as.Date(paste0(1986:2003, "-01-01"), "%Y-%m-%d")
#read and prepare bird count data as single-column data frame:
ecd <- read.table("ECDoveBBS1986_2003.dat", header=FALSE)
ecd[ecd == -1] <- NA
ecd.data = data.frame(counts = as.vector(as.matrix(ecd)))
#retrieve Florida state boundaries:
m <- map("state", "florida", fill = TRUE, plot = FALSE)
FL <- map2SpatialPolygons(m, "FL")
proj4string(FL) <- proj4string(ecd.locations)

#plot location observations near Florida
plot(FL)
points(ecd.locations, pch="+", col="red")

## Prepare semantics

# Create function wrapper for STFDF constructur
postprocessor = function(args, output, semantics){
  functionalType(output, parent = FALSE) <- "MarkedEvent"
  return(output)
}
STFDF.MarkedEvent = STFDF #give constructor a more informative name
captureSemantics(STFDF.MarkedEvent, postprocessor = postprocessor) <-
    TRUE
```

```
# Create manual wrapper function arround aggregate (direct wrapping currently
    not supported)
aggregate.st <- function(x, by, FUN, na.rm){
  output = aggregate(x, by, FUN, na.rm = na.rm)
  functionalType(output) <- "TLattice"
  return(output)
}
#encapsulate intervall-subsetting as a semantic function
getTimeIntervals = function(T_set, breaks){
  output = T_set[breaks]
  attr(output, "semantics") <- "I set"
  return(output)
}
#encapsulate STF-constructor in a function that preserves semantics
STF.sem = function(sp, time){
  output = STF(sp, time) #sp semantics are preserved already
  attr(output@time, "semantics") <- attr(time, "semantics")
  return(output)
}

#Analysis starts here:
enableProvenance()
  #Create space-time object of all bird counts
  ecd.st <- STFDF.MarkedEvent(ecd.locations, ecd.years, ecd.data)
  #Create target-geometry to aggregate over Florida-state area
  # in 2-year periods
  target.years = getTimeIntervals(ecd.years, c(4,6,8,10,12) )
  target.st <- STF.sem(FL, target.years)
  #execute aggregation
  ts = aggregate.st(ecd.st, target.st, sum, na.rm = TRUE)
disableProvenance()

#plot and export derivatin graph
g = getScriptGraph()
plot(g)
toFile(g, layoutType="dot", filename="Florida.dot", fileType="dot")
system(command = "dot -Tpdf Florida.dot -o Florida.pdf")
#query semantics
getSemanticPedigree(ecd.st)
getSemanticPedigree(ts)
getSemanticPedigree(target.st)
#rset provenance to its default state
reset_provenance()
```

Appendix D

Spatial prediction - R script

LISTING D.1: Spatial prediction - complete R script

```
library(mss)
library(sp)
library(gstat)
library(Rgraphviz)
library(SpatialSemantics)

# define helper functions
#
    ------------------------------------------------

init_model = function(pointData) {
  range = sqrt(sum(apply(bbox(pointData@observations), 1, diff)^2)) / 6
   sill  = var(pointData[[1]])
  vgm(2 * sill  / 3, "Sph", range, sill  / 3) #  initial  variogram model
}
captureSemantics(init_model) <-TRUE
modelSemivariogram = function(pointData) {
  n = names(pointData@observations)
  if  (length(n) > 1)
    warning("taking first  attribute  variable")
  f  = as.formula(paste(n[1], "~1")) # which variable to model? take first .
  init  = init_model(pointData)
  fit .variogram(variogram(f, pointData@observations), init)
}
captureSemantics(modelSemivariogram) <-TRUE
getInterpolator  = function(params, pointData) {
  if  (!is(params, "variogramModel"))
    warning("getInterpolator: params should be of class variogramModel")
  out=function(locOfInterest)  {
    n = names(pointData@observations)[1] #take first variogram model
    f  = as.formula(paste(n, "~ 1"))
    out=interpolate(f, pointData, locOfInterest, model = params)
    functionalType(out@observations) <- "SField"
    return(out)
  }
  captureSemantics(out, semantics = "S -> (S, Q)") <-TRUE
  attr(out, "semantics") <- "(S -> (S x Q))"
  return(out)
}
captureSemantics(getInterpolator, procedureName="getInterpolator") <-TRUE
captureSemantics(geometry, procedureName="fst") <- TRUE
SFieldData <- SField #rename constructor to avoid ambiguities
captureSemantics(SFieldData, procedureName = "SFieldData",
  postprocessor = function(args, output, call_semantics) {
```

```
43      attr(output@observations, "semanticPedigree") <- getSemanticPedigree(
            args$observations)
44      attr(output@observations, "semantics") <- attr(args$observations,"semantics
            ")
45      return(output)
46    }
47  ) <- TRUE
48
49  enableProvenance() # Run analysis
50      # load meuse data from package sp in current session:
51      demo(meuse, ask=FALSE, echo=FALSE)
52      functionalType(meuse) <- "SField"
53      functionalType(meuse.grid) <- "SField"
54      meuse$lzinc = log(meuse$zinc)
55      zincPointData = SFieldData(meuse["lzinc"], meuse.area)
56      interpolator  = getInterpolator(modelSemivariogram(zincPointData),
            zincPointData)
57      locInterest  = SFieldData(geometry(meuse.grid), geometry(meuse.grid),
            cellsArePoints = TRUE)
58      intZincPointData = interpolator( locInterest ,  semantics = "S  set  -> S x Q set
            ")
59  disableProvenance() #end of analysis
60
61  spplot(intZincPointData@observations["var1.pred"])
62  plot(zincPointData) #plot data
63  # Export / visualize  derivation  graph
64  g = getScriptGraph()
65  plot(g,  main="Derivation Graph")
66  toFile(g ,  layoutType="dot", filename="meuse-prediction.dot", fileType="dot")
67  system(command = "dot -Tpdf meuse-prediction.dot -o meuse-prediction.pdf
        ")
68
69  reset_provenance()
```

Appendix E

Malaria Clustering - R script

LISTING E.1: Malaria Clustering - complete R script

```
library(SPODT)
library(tree)
library(SpatialSemantics)
library(Rgraphviz)
#prepare semantics and wrappers
validator = function(args, output, semantics, assumtions){
  data = args[["data"]]
  pedigreeData = getSemanticPedigree(data)
  valid = "SMarkedEvent" %in% pedigreeData$procedureName
  if(!valid){
    message= "Spatial partitioning with the given input data was not intended.\
        nExpected were SMarkedEventData, "
    if(is.null(functionalType(data)))
      message = paste0("but functional type of data is unknown.")
      else
    message = paste0(message, "but given were ",functionalType(data),"Data.")
    warning(message)
  }
  return(valid)
}
spodt.malaria <- function(params, data){
  args = append(params, list(data=data))
  output = base::do.call(SPODT::spodt, args)
  attr(output, "semantics") <-"SInvField"
  return(output)
}
captureSemantics(spodt.malaria, validator=validator) <- TRUE
postprocessor = function(args, output, semantics){
  functionalType(output, parent=FALSE) <- "SInvField"
  return(output)
}
captureSemantics(spodtSpatialLines, postprocessor=postprocessor) <- TRUE

#data preparation
data("dataMALARIA")
coordinates(dataMALARIA) <- c("x", "y")
proj4string(dataMALARIA) <- "+proj=longlat +datum=WGS84 +ellps=WGS84"
dataMALARIA <- spTransform(dataMALARIA, CRS("+proj=merc +datum=
    WGS84 +ellps=WGS84"))
functionalType(dataMALARIA) <- "SMarkedEvent"

enableProvenance()# start recording provenance
    params = list(formula = z ~1, graft = 0.13, level.max = 7, min.parent = 25,
        min.child = 2, rtwo.min = 0.01)
```

```
42      spodt.results <- spodt.malaria(params = params, dataMALARIA)
43      #create spatial lines that divide the study area into regions of higher /
            lower malaria risk
44      SSL.result <- spodtSpatialLines(spodt.results, dataMALARIA)
45  disableProvenance()
46
47  ##PLOT RESULTS
48  # classification tree
49  spodt.tree(spodt.results)
50  plot(SSL.result)
51  #adding each location
52  points(dataMALARIA, cex = log(dataMALARIA@data$z*10))
53  ### END OF ANALSIS
54
55  #visualize / export derivation graph
56  g = getScriptGraph()
57  plot(g, main="Derivation Graph")
58  toFile(g , layoutType="dot", filename="SPODT-use-case.dot", fileType="dot")
59  system(command = "dot -Tpdf SPODT-use-case.dot -o SPODT-use-case.pdf
        ")
60
61  #delete internal provenance record
62  reset_provenance()
63
64  #test validator against meuse data set
65  library(sp)
66  demo(meuse,echo = FALSE, ask = FALSE)
67  functionalType(meuse) <- "SField"
68  params = list(formula = zinc ~1, graft = 0.13, level.max = 7, min.parent = 25,
        min.child = 2, rtwo.min = 0.01)
69  meuse.results = spodt.malaria(params,meuse)
```

Bibliography

Allen, David W (2011). *Getting to Know ArcGIS ModelBuilder*. Esri Press.

Alper, Pinar et al. (2013). "Enhancing and abstracting scientific workflow provenance for data publishing". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. Association for Computing Machinery (ACM), pp. 313–318. DOI: 10.1145/2457317.2457370. URL: http://dx.doi.org/10.1145/2457317.2457370.

Altintas, Ilkay, Oscar Barney, and Efrat Jaeger-Frank (2006). "Provenance Collection Support in the Kepler Scientific Workflow System". In: *International Provenance and Annotation Workshop*. Springer. Springer Science + Business Media, pp. 118–132. DOI: 10.1007/11890850_14. URL: http://dx.doi.org/10.1007/11890850_14.

Baddeley, Adrian, Ege Rubak, and Rolf Turner (2015). *Spatial point patterns: methodology and applications with R*. CRC Press.

Barga, Roger S et al. (2010). "Provenance for Scientific Workflows Towards Reproducible Research." In: *IEEE Data Eng. Bull.* 33.3, pp. 50–58.

Bivand, Roger S., Edzer Pebesma, and Virgilio Gómez-Rubio (2013). *Applied Spatial Data Analysis with R*. New York: Springer. DOI: 10.1007/978-1-4614-7618-4. URL: http://dx.doi.org/10.1007/978-1-4614-7618-4.

Bivand, Roger and Nicholas Lewin-Koh (2016). *maptools: Tools for Reading and Handling Spatial Objects*. R package version 0.8-39. URL: https://CRAN.R-project.org/package=maptools.

Chambers, John M (2016). *Extending R*. CRC Press.

Coulibaly, Drissa et al. (2013). "Spatio-temporal analysis of malaria within a transmission season in Bandiagara, Mali". In: *Malaria journal* 12.1, p. 82. DOI: 10.1186/1475-2875-12-82. URL: http://dx.doi.org/10.1186/1475-2875-12-82.

Cressie, Noel and Christopher K Wikle (2011). *Statistics for Spatio-Temporal Data*. John Wiley & Sons.

Gansner, Emden R. and Stephen C. North (2000). "An open graph visualization system and its applications to software engineering". In: *SOFTWARE - PRACTICE AND EXPERIENCE* 30.11, pp. 1203–1233. DOI: 10.1002/1097-024x(200009)30:11<1203::aid-spe338>3.0.co;2-n. URL: http://dx.doi.org/10.1002/1097-024x(200009)30:11%3C1203::aid-spe338%3E3.0.co;2-n.

Gaudart, Jean et al. (2015). "SPODT: An R Package to Perform Spatial Partitioning". In: *Journal of Statistical Software* 63.16. DOI: 10.18637/jss.v063.i16. URL: http://dx.doi.org/10.18637/jss.v063.i16.

Gentleman, R. et al. (2016). *graph: graph: A package to handle graph data structures*. R package version 1.50.0.

Gil, Yolanda, James Cheney, et al. (2010). "Provenance XG final report". In: *Final Incubator Group Report*. URL: http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/ (visited on 08/25/2016).

Gil, Yolanda, Simon Miles, et al. (2013). "PROV model primer". In: *W3C Working Draft, 11th December*. URL: http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/ (visited on 08/25/2016).

Griner, CS and WB Keegan (2000). "Enhancing mission success–a framework for the future: a report by the NASA chief engineer and the NASA integrated action team". In: *National Aeronautics and Space Administration*. URL: http://history.nasa.gov/niat.pdf (visited on 08/25/2016).

Group, W3C SPARQL Working (2013). "SPARQL 1.1 Overview." In: *W3C recommendation*. URL: https://www.w3.org/TR/sparql11-overview/ (visited on 09/21/2016).

Hansen, Kasper Daniel et al. (2016). *Rgraphviz: Provides plotting capabilities for R graph objects*. R package version 2.16.0.

Hengl, Tomislav (2009). *A practical guide to geostatistical mapping*. Vol. 52. Hengl.

Isbell, Douglas, Mary Hardin, and Joan Underwood (1999). "Mars Climate Orbiter team finds likely cause of loss". In: *NASA news release*. URL: http://mars.nasa.gov/msp98/news/mco990930.html (visited on 08/25/2016).

Jensen, Scott et al. (2013). "Provenance capture and use in a satellite data processing pipeline". In: *IEEE Transactions on Geoscience and Remote Sensing* 51.11, pp. 5090–5097. DOI: 10.1109/tgrs.2013.2266929. URL: http://dx.doi.org/10.1109/tgrs.2013.2266929.

Klyne, Graham and Jeremy J. Carroll (2004). "Resource Description Framework (RDF): Concepts and Abstract Syntax". In: *W3C Recommendation*. Ed. by Brian Mcbride. URL: http://www.w3.org/TR/rdf-concepts/.

Lang, Duncan Temple, Roger Peng, and Deborah Nolan. *CodeDepends: Analysis of R code for reproducible research and code comprehension*. R package version 0.4-2.

Le Lann, Gérard (1997). "An analysis of the Ariane 5 flight 501 failure-a system engineering perspective". In: *Proceedings International Conference and Workshop on Engineering of Computer-Based Systems*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), pp. 339–346. DOI: 10.1109/ecbs.1997.581900. URL: http://dx.doi.org/10.1109/ecbs.1997.581900.

Lerner, Barbara and Emery Boose (2014). "RDataTracker: collecting provenance in an interactive scripting environment". In: *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*.

McGuinness, Deborah L, Frank Van Harmelen, et al. (2004). "OWL web ontology language overview". In: *W3C recommendation* 10.10, p. 2004.

Miles, Simon et al. (2007). "Provenance-based validation of e-science experiments". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.1, pp. 28–38. DOI: 10.1016/j.websem.2006.11.003. URL: http://dx.doi.org/10.1016/j.websem.2006.11.003.

Missier, Paolo, Khalid Belhajjame, and James Cheney (2013). "The W3C PROV family of specifications for modelling provenance metadata". In: *Proceedings of the 16th International Conference on Extending Database Technology*. ACM. Association for Computing Machinery (ACM), pp. 773–776. DOI: 10.1145/2452376.2452478. URL: http://dx.doi.org/10.1145/2452376.2452478.

Missier, Paolo, Satya S Sahoo, et al. (2010). "Janus: From Workflows to Semantic Provenance and Linked Open Data". In: *International Provenance and Annotation Workshop*. Springer Science + Business Media, pp. 129–141. DOI: 10.1007/978-3-642-17819-1_16. URL: http://dx.doi.org/10.1007/978-3-642-17819-1_16.

Moreau, Luc et al. (2011). "The Open Provenance Model core specification (v1.1)". In: *Future Generation Computer Systems* 27.6, pp. 743–756. DOI: 10.1016/j.future.2010.07.005. URL: http://dx.doi.org/10.1016/j.future.2010.07.005.

Pebesma, E. J. and C. Stasch (2014). "Meaningfully Integrating Big Earth Science Data". In: *AGU Fall Meeting Abstracts*.

Pebesma, Edzer (2012). "spacetime: Spatio-temporal data in r". In: *Journal of Statistical Software* 51.7, pp. 1–30. DOI: 10.18637/jss.v051.i07. URL: http://dx.doi.org/10.18637/jss.v051.i07.

Pebesma, Edzer J. (2004). "Multivariable geostatistics in S: the gstat package". In: *Computers & Geosciences* 30.7, pp. 683–691. DOI: 10.1016/j.cageo.2004.03.012. URL: http://dx.doi.org/10.1016/j.cageo.2004.03.012.

Plale, Beth, Bin Cao, and Mehmet S Aktas (2011). "Provenance Capture of Unmanaged Workflows with Karma". In:

R Core Team (2000). "R language definition". Version 3.3.1 Patched (2016-09-13) DRAFT. In: *Vienna, Austria: R Foundation for Statistical Computing*. URL: https://www.R-project.org/ (visited on 09/17/2016).

— (2016). *R: A Language and Environment for Statistical Computing*. Version Version 3.3.1 Patched (2016-09-13). R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/ (visited on 09/15/2016).

Richard A. Becker, Original S code by, Allan R. Wilks. R version by Ray Brownrigg. Enhancements by Thomas P Minka, and Alex Deckmyn. (2016). *maps: Draw Geographical Maps*. R package version 3.1.1. URL: https://CRAN.R-project.org/package=maps.

Rikken, MGJ and RPG Van Rijn (1993). "Soil pollution with heavy metals-an inquiry into spatial variation, cost of mapping and the risk evaluation of copper, cadmium, lead and zinc in the floodplains of the Meuse west of Stein, the Netherlands". In: *Doctoraalveldwerkverslag, Dept. of Physical Geography, Utrecht University* 74.

Ryan, Jeffrey A. and Joshua M. Ulrich (2014). *xts: eXtensible Time Series*. R package version 0.9-7. URL: https://CRAN.R-project.org/package=xts.

Sauser, Brian J., Richard R. Reilly, and Aaron J. Shenhar (2009). "Why projects fail? How contingency theory can provide new insights–A comparative analysis of NASA's Mars Climate Orbiter loss". In: *International Journal of Project Management* 27.7, pp. 665–679. DOI: 10.1016/j.ijproman.2009.01.004. URL: http://dx.doi.org/10.1016/j.ijproman.2009.01.004.

Scheidegger, Carlos et al. (2008). "Tackling the Provenance Challenge one layer at a time". In: *Concurrency and Computation: Practice and Experience* 20.5, pp. 473–483. DOI: 10.1002/cpe.1237. URL: http://dx.doi.org/10.1002/cpe.1237.

Scheider, Simon (2012). *Grounding geographic information in perceptual operations*. Vol. 244. IOS Press.

Scheider, Simon et al. (2016). "Modeling spatiotemporal information generation". In: *International Journal of Geographical Information Science*, pp. 1–29. DOI: 10.1080/13658816.2016.1151520. URL: http://dx.doi.org/10.1080/13658816.2016.1151520.

Silles, Christopher Anthony (2014). "Provenance-aware CXXR". PhD thesis. University of Kent, URL: https://kar.kent.ac.uk/id/eprint/50499 (visited on 08/25/2016).

Stasch, Christoph, Edzer Pebesma, and Simon Scheider (2014). "Annotating spatio-temporal datasets for meaningful analysis in the Web". In: *Environmental Modelling & Software* 51, pp. 149–165.

Stasch, Christoph, Simon Scheider, et al. (2014). "Meaningful spatial prediction and aggregation". In: *Environmental Modelling & Software* 51, pp. 149–165. DOI: 10.1016/j.envsoft.2013.09.006. URL: http://dx.doi.org/10.1016/j.envsoft.2013.09.006.

Stephenson, Arthur G et al. (1999). "Mars climate orbiter mishap investigation board phase i report". In: *NASA, Washington, DC*. URL: ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf (visited on 08/25/2016).

Suriarachchi, Isuru, Quan Zhou, and Beth Plale (2015). "Komadu: A Capture and Visualization System for Scientific Data Provenance". In: *Journal of Open Research Software* 3.1. DOI: 10.5334/jors.bq. URL: http://dx.doi.org/10.5334/jors.bq.

Tierney, Luke (2015). *codetools: Code Analysis Tools for R*. R package version 0.2-14. URL: https://CRAN.R-project.org/package=codetools.

Wickham, Hadley (2014). *Advanced R*. Chapman and Hall/CRC. DOI: 10.1201/b17487. URL: http://dx.doi.org/10.1201/b17487.

— (2016). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.1.0. URL: https://CRAN.R-project.org/package=stringr.

Wickham, Hadley and Winston Chang (2016). *devtools: Tools to Make Developing R Packages Easier*. R package version 1.12.0. URL: https://CRAN.R-project.org/package=devtools.

Zandbergen, Paul A (2013). *Python scripting for ArcGIS*. Esri press.

Zhao, Jing Hua (2006). "Pedigree-drawing with R and graphviz". In: *Bioinformatics* 22.8, pp. 1013–1014. DOI: 10.1093/bioinformatics/btl058. URL: http://dx.doi.org/10.1093/bioinformatics/btl058.