TECHNISCHE
UNIVERSITÄT
WIEN

Dissertation

Context Algebra applied to Spatial Concepts

Ausgeführt zu Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Prof. Dr. Andrew U. Frank
Forschungsgruppe Geoinformation
E 120–2

Prof. Dr. Martin Raubal
Institut für Kartografie und Geoinformation
ETH Zürich

Eingereicht an der Technischen Universität Wien
Fakultät Mathematik und Geoinformation

von
Jürgen Hahn
Matrikelnummer: 0657023
Kaiserstrasse 28/1/14, 1070 Wien, Österreich

Wien, am 23. August 2016

Andrew U. Frank	Martin Raubal	Jürgen Hahn

Zusammenfassung

Wörter referenzieren Objekte in der Welt, wobei ein Wort viele Objekte in der Welt referenzieren kann, die als ähnlich wahrgenommen werden. Das Wort "Stadt" referenziert unter anderem die Objekte: Wien, Alexandria oder Las Vegas; das Wort "nahe" kann zum Beispiel die unterschiedlichsten Distanzen referenzieren wie "der Mond ist nahe der Erde" oder "nahe dem Stephansdom". Enthält eine Anfrage an ein Geografisches Informationssystem (GIS) das Wort "Stadt" oder "nahe" muss ein Algorithmus im GIS entscheiden, welche Stadt oder welche Distanz referenziert wird.

Um eine Entscheidung treffen zu können, baut diese Arbeit auf der Hypothese auf, dass *Kontext Referenzen zwischen Wörtern und Objekten in der Wirklichkeit selektiert*. Um diese Hypothese zu belegen, wird eine Kontext-Algebra vorgeschlagen, implementiert und damit das Wort "nahe" modelliert.

Die Kontext-Algebra basiert auf der Theorie eines um *Kontext erweiterten semiotischen Dreieckes* (Figure 1). Dieses Dreieck verbindet Objekte in der Wirklichkeit über Konzepte in einem Agenten mit Wörtern. Durch die Erweiterung um Kontext wird das Konzept in kontextualisierte Konzepte aufgeteilt, welche nur Objekte beinhalten, die in einem Kontext gültig sind. Wird ein Wort nun in solchem Kontext verwendet, entspricht es einem kontextualisierten Konzept, das somit die selektierten Objekten in der Wirklichkeit referenziert.

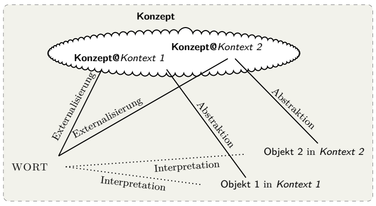


Figure 1: Semiotisches Dreieck um Kontext erweitert

Die Kontext-Algebra bietet eine Formalisierung für Kontext an. Dabei werden Kontexte durch eine *Ordnungsrelation* verglichen und neue Kontexte entstehen durch die Anwendung der *Konjunktions-* oder *Disjunktionsfunktion* von zwei Kontexten. Für die Ordnungsrelation und die beiden Funktionen sind algebraische Eigenschaften festgelegt. Mit diesen Eigenschaften, der Ordnungsrelation und den Funktionen wird Kontext mittels der mathematischen Struktur Verband formalisiert.

Alle Kontexte im Verband werden auf Objekte, beobachtet in der Wirklichkeit, zusammengefasst in kontextualisierte Konzepte abgebildet. Aus kontextualisierten Konzepten werden typische Objekte (Prototypen) ermittelt, die als Übersetzung von einem Wort auf ein Objekt in der Wirklichkeit dienen. Zum Beispiel kann mittels dem Kontext "Hauptstadt von Österreich" von dem Wort "Stadt" die prototypische Instanz Wien ermittelt werden.

Die Kontext-Algebra wird in Haskell implementiert, verifiziert, sowie dessen Komplexitätsklasse bestimmt. Diese Implementierung zeigt, dass die Kontext-Algebra mit akzeptabler Performanz realisierbar ist. Die Implementierung erfüllt die algebraischen Eigenschaften, verifiziert durch Tests, und mittels Benchmarks ist die Komplexitätsklasse *exponentiell* bestimmt worden.

Mittels der Kontext-Algebra wird das Wort “nahe” modelliert. Um das Modell realistisch zu gestalten sind Kontexte und Objekte (Distanzen) aus Immobilienbeschreibungen durch einen manuellen Prozess erhoben worden. Das initialisierte Modell für “nahe” ermöglicht es, verschiedene Distanzen abhängig vom Kontext (Größe des referenzierten Objektes, Aktivität) zu referenzieren. In unserem Beispiel referenziert “nahe dem See” eine Distanz von 1050 Metern, “nahe der U-Bahn” eine Distanz von 380 Metern und “Gehnähe” eine Distanz von 550 Metern.

Die Ergebnisse aus der Anwendung der Kontext-Algebra für das Wort “nahe” zeigen die Plausibilität des um Kontext erweiterten semiotischen Dreiecks. Dies lässt den Schluss zu, dass die Hypothese valide ist.

Schlüsselwörter: Kontext, Kontext-Algebra, Kontext erweitertes Semiotisches Dreieck, Geographische Konzepte, Modell für nahe

Abstract

Words are used to refer to objects in reality. One word can imply many references to different objects that are categorized as similar. For example, the word "city" can refer to Vienna, Alexandria, or Las Vegas; the word "near" can refer to a range of distances, *e.g.* "moon is near the earth" or "near St. Stephens cathedral". If a Geographic Information System (GIS) is queried with a sentence including "city" or "near", the challenge for an algorithm executed by the GIS is to decide which exemplar of the word "city" or which distance "near" refers to.

To overcome this challenge, the hypothesis is that *context selects references to objects in reality.* A context algebra is presented, implemented, and used to represent the word "near", in order to evaluate the hypothesis.

Context algebra makes use of the theory established by a *context-enriched semiotic triangle* (Figure 2). The semiotic triangle connects objects in reality to words via concepts in an agent. With context enrichment, the concept is separated into contextualized concepts that include objects in reality valid for a specific context. If words are used in this specific context, then they correspond to a specific contextualized concept, which then selects specific references to objects from reality.

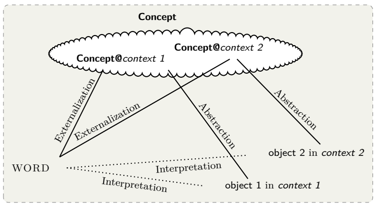


Figure 2: Context enriched semiotic triangle

Context algebra proposes a formalization for context. In this algebra, contexts are ordered with a *partial order relation* and can be combined with a *disjunction* or *conjunction* function to create other contexts. This relation and these functions satisfy algebraic properties that result in a lattice structure for context.

Each context included in the lattice is mapped to a contextualized concept. A contextualized concept is modeled with sets of objects observed from reality, where a typical object is determined. This typical object (prototype) is assumed to be the translation from a word to an object in reality. For example, the influencing context "capital of Austria" for the word "city" selects the prototypical instance Vienna.

Context algebra is implemented using Haskell, it is then proven, and the complexity class is determined. The implementation shows that the context algebra is realizable with reasonable performance. To prove that the implementation satisfies the proposed algebraic properties, tests are successfully executed. The benchmarks determine the complexity class *exponential* for the implementation.

The word "near" is modeled using the context algebra. To model "near" realistically, data is acquired via a manual identification process of real estate entries. The initialized model makes it possible to determine references to different distances according to contexts, *e. g.* the size of the referenced object or activity. For example, "near the lake" references 1050 meters, "near by subway" references 380 meters, and "near by walking" references 550 meters.

The results of the application for "near" provide evidence that the hypothesis is valid, and that it is able to select references which translate for example "near" into metric distances.

Keywords: context, context algebra, context-enriched semiotic triangle, concept, model for near

IF I HAVE SEEN FURTHER, IT IS BY STANDING ON THE SHOULDERS OF GIANTS.

ISAAC NEWTON

A SCIENTIFIC THEORY SHOULD BE AS SIMPLE AS POSSIBLE, BUT NOT SIMPLER.

ALBERT EINSTEIN

CONTEXT (POINT OF VIEW) IS WORTH 80 IQ POINTS.

ALAN KAY *AT A CREATIVE THINK SEMINAR*

JÜRGEN HAHN

CONTEXT ALGEBRA APPLIED TO SPATIAL CONCEPTS

TECHNISCHE UNIVERSITÄT WIEN,
DEPARTMENT OF GEODESY AND GEOINFORMATION,
RESEARCH GROUP GEOINFORMATION

Acknowledgments

This thesis was only possible, because of the support from many people.

Thank you Prof. Frank, for inviting me to join the research group, for giving me the freedom to develop ideas and for your comprehensive feedback to early versions of this thesis. I want to thank you Martin, for your positive feedback after the AGILE presentation, for being my second advisor, for your comments that improved this thesis and the discussions at ETH. Mille grazie Paolo for asking those kind of questions that helped clarifying ideas and for being a friend. Dankeschön Paul for the enlightening discussions about quantum cognition, for publishing a paper together, for inviting me to Zurich and more. Thanks colleagues at the research group geoinformation: Thomas Fuhrmann, Markus Mayr, Amin Abdallah, Heidi Hobel and Gerhard Navratil - the discussions with you created different perspectives on my ideas which strengthened the outcome. Eva Maria Holy, thank you for your support, this means a lot to me.

Thank you my darling Daniela, I couldn't have done it without you. You were patient when I needed it, and also pushed me to my limits for example with the experiment "Schlafen am Limit"! I'm so thankful for having such a family, you always supported my goals. I cannot imagine how difficult it is to have a family member that is working on a project that he can only explain at the very end - Dankeschön.

Thank you redaction of derStandard and willhaben for the data, especially Michela Biocchi and Gerhard Muggenhuber for your support to get the data. Thank you geocoders, for your time: Daniela, Markus, Mathias, Barbara, Melanie, Karin, Heidi, Christian, Mama and Angela. Thank you Barbara for reviewing parts of the thesis.

Thank you Andreas Pinzenöhler, for inspiring discussions. Your comments and critique helped me to stay focused and to find possible applications.

Thank you! Mille grazie! Dankeschön! Merci!

Publications

Particular results from this thesis have been published in the following articles:

- **Juergen, Hahn**, Paolo Fogliaroni, Andrew U. Frank, and Gerhard Navratil. A computational model for context and spatial concepts. In Tapani Sarjakoski, Yasmina Maribel Santos, and Tiina L. Sarjakoski, editors, *Geospatial Data in a Changing World: Selected papers of the 19th AGILE Conference on Geographic Information Science*, pages 3–19. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33783-8. DOI: 10.1007/978-3-319-33783-8_1. URL http://dx.doi.org/10.1007/978-3-319-33783-8_1.
- Andrew, U. Frank and **Juergen, Hahn**. Was sind Spiele. In *Realism – Relativism –Constructivism, 38 th International Wittgenstein Symposium August 9 – 15, 2015 Kirchberg am Wechsel, Austria.* The Austrian Ludwig Wittgenstein Society, 2015.
- **Juergen, Hahn** and Paul Weiser. A quantum formalization for communication coordination problems. In H. Atmanspacher, C. Bergomi, T. Filk, and K. Kitto, editors, *Quantum Interaction*, volume 8951 of *Lecture Notes in Computer Science*, pages 177–188. Springer International Publishing, 2015. ISBN 978-3-319-15931-7. DOI: 10.1007/978-3-319-15931-7_14. URL http://dx.doi.org/10.1007/978-3-319-15931-7_14
- **Juergen, Hahn** and Andrew, U. Frank. Select the appropriate map depending on context in a hilbert space model (scop). In H. Atmanspacher, E. Haven, K. Kitto, and D. Raine, editors, *Quantum Interaction*, Lecture Notes in Computer Science, pages 122–133. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-54943-4. DOI: 10.1007/978-3-642-54943-4_11 URL http://dx.doi.org/10.1007/978-3-642-54943-4_11
- **Juergen, Hahn**. A hilbert space model for concepts and contextual influences to concepts. In *Doctoral Colloquium, Conference on Spatial Information Theory (COSIT)*. Electronic Conference Proceedings, 2013.
- **Juergen, Hahn** and Andrew, U. Frank. Context determines the features which need to show on a map, produced on demand. In *1st International Workshop on Context for Business Process Management (CBPM'13), Eight International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2013), Annecy, Haute-Savoie, France.*, 2013.

Contents

List of Figures

List of Tables

List of Code Fragments and Algorithms

1 Introduction

A natural language interface for Geographic Information Systems (GIS) has to translate natural language into values usable for spatial algorithms. One example for a natural language input would be: "Show me grocery shops near my location". The spatial algorithm `within` (function illustrated in Figure 1.1) is capable of answering the query by selecting all grocery shops in a circle, with the personal location as the center. The `within` algorithm depends on a data set including grocery shops, the exact personal location, and the metric value of the radius for the circle. Data sets including grocery shops are publicly available (*e. g.* OpenStreetMap[1]) and the personal location can be measured through sensors using global navigation satellite systems. The metric distance (*e. g.* 100 meters) defining the radius, however, is missing because it cannot be determined from the natural language input alone. In order to use the algorithm, a translation mechanism between natural language and metric distances has to be found. The proposal of such a translation mechanism between natural language and metric distances is the goal of this thesis.

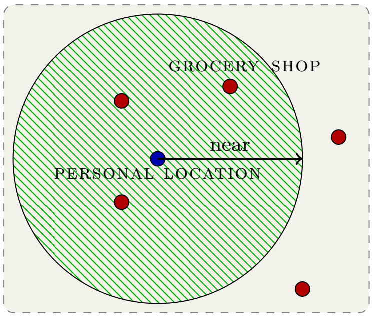


Figure 1.1: "Near" refers to which distance?

The approach to create a translation mechanism from words to metric distances is inspired by the human interpretation process. In this interpretation process, humans use natural language to refer to objects in reality. Humans carry out interpretation tasks in everyday situations, such as when translating "near" into metric values, or in wayfinding tasks. In the example of interpreting a wayfinding task, *e. g.* "turn right at the traffic light", the words must be interpretable with the available sensory inputs.

In formalizations from words to objects in reality (facts), ambiguities emerge. McCarthy [1987] and his coworkers established a knowledge base including facts about reality that are common to everyone. One such fact could be "Evening Star" referring to "planet Venus" – which is true, but it is also true that "Morning Star" refers to "planet Venus" as well. These two facts create a contradiction in the knowledge base for the word "Venus". From such contradiction everything else can be followed, ex falso quodlibet. To cope with ambiguities arising in formalizations from words to objects in reality is the major challenge of this thesis.

Context was proposed to resolve ambiguities arising in knowledge bases. The integration of context in knowledge bases used in Ar-

[1] `http://www.openstreetmap.org/`

tificial Intelligence (AI) [McCarthy, 1987] was key to distinguishing between facts. Additionally, software including context established, for example, the fields of ubiquitous computing [Weiser, 1993], and context-aware computing [Schilit et al., 1994]. In this thesis, context is proposed as a means to resolve ambiguities in the referring process, which is formalized to establish a translation mechanism from natural language to metric distances.

1.1 Introductory Example – Let's meet at a restaurant

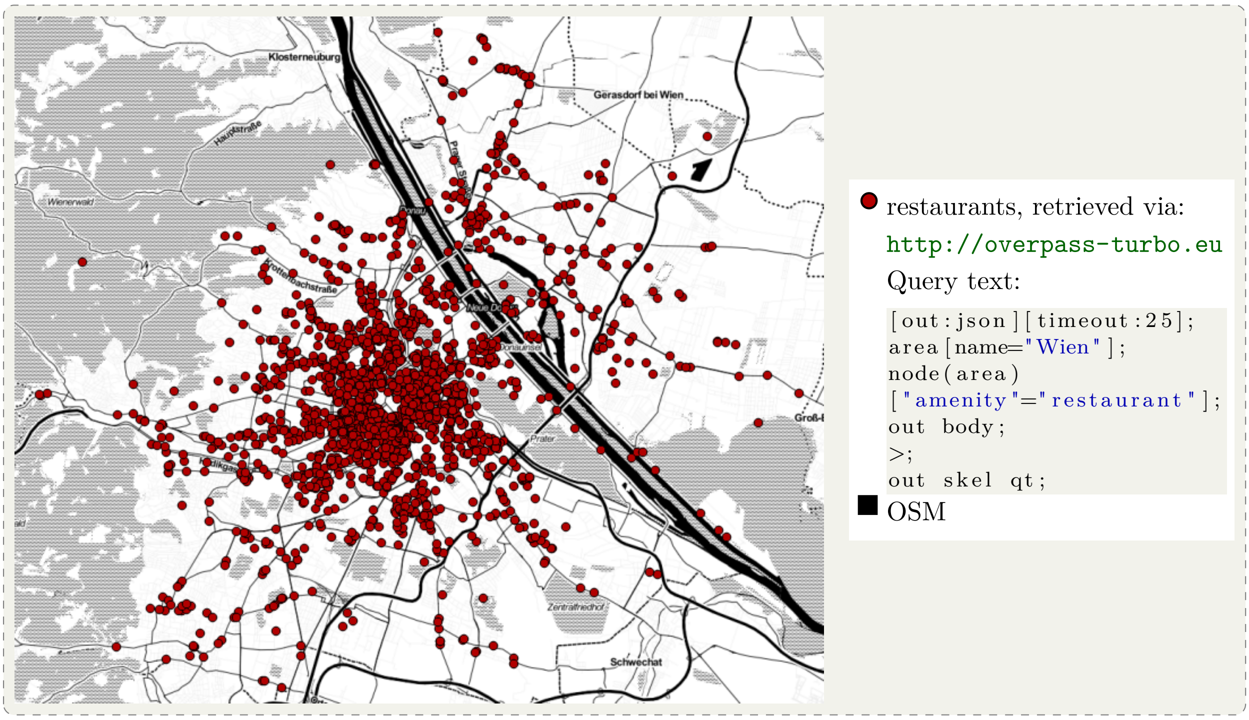
The conversation presented in this example shows how context influences the referring process. The example is as follows: two subjects want to meet for dinner, and one of them proposes: "Let's meet at a restaurant". The focus of this example is to show how context influences the number of references that have to be taken into account when inferring from the word "restaurant" corresponding objects in reality: *e. g.* vegan restaurants, fish restaurants, and so on. When two people want to meet at one restaurant, the references to multiple restaurants must be reduced to only one reference. But how do people use the word restaurant so that it refers only to one object in reality? The answer is stated by the guiding hypothesis of this work: *Context selects appropriate references to objects in reality.* This selection is illustrated by maps, showing the number of restaurants it actually refers to.

The references for the word "restaurant" can be reduced by specifying a regional context, *e. g.* *in Vienna*[2]. Including this context in the above statement results in "Let's meet at a restaurant in Vienna". This context influence reduces possible references to only these restaurants located in Vienna. Using the *OpenStreetMap* (OSM) data set[3], the number of restaurants is reduced from 10484 in Austria to 2048 in Vienna, shown as red points on the map in Figure 1.2. Compared to the number of references that are included only for Austria in the data set, the context "in Vienna" already reduces the number to 1/5.

Two contexts differ in the number of references they reduce, which is used to partially order the two respective contexts. Considering the contexts *in Vienna* and *Innere Stadt* (translation: *downtown*) the number of reduction differs. Influencing the statement with the second context results in the statement "Let's meet at a restaurant in Innere Stadt", where the number of restaurants included in the data set is reduced to 363 restaurants. Compared to the 2048 from *in Vienna*, this is a further reduction to approximately 1/5 of selected references. This fact is used to bring the two contexts in relation to each other, using the relation "is more selective". In this example context, *Innere Stadt* is "more selective" compared to *in Vienna*, as measured by the number of references. In summary, two contexts can be ordered with the order relation "is more selective" by counting the number of references. Pursuing the example above by illustrating the statement on a map, restaurants in other cities are presented in the map as well. In Figure 1.3 the map also includes restaurants in the cities Klagenfurt and Graz. This is counter intuitive to the human usage of context, as

[2] to indicate context, it will be styled with italic sans serif font throughout this work

[3] queried at July 15 2016



http://overpass-turbo.eu

Query text:

```
[out:json][timeout:25];
area[name="Wien"];
node(area)
["amenity"="restaurant"];
out body;
>;
out skel qt;
```

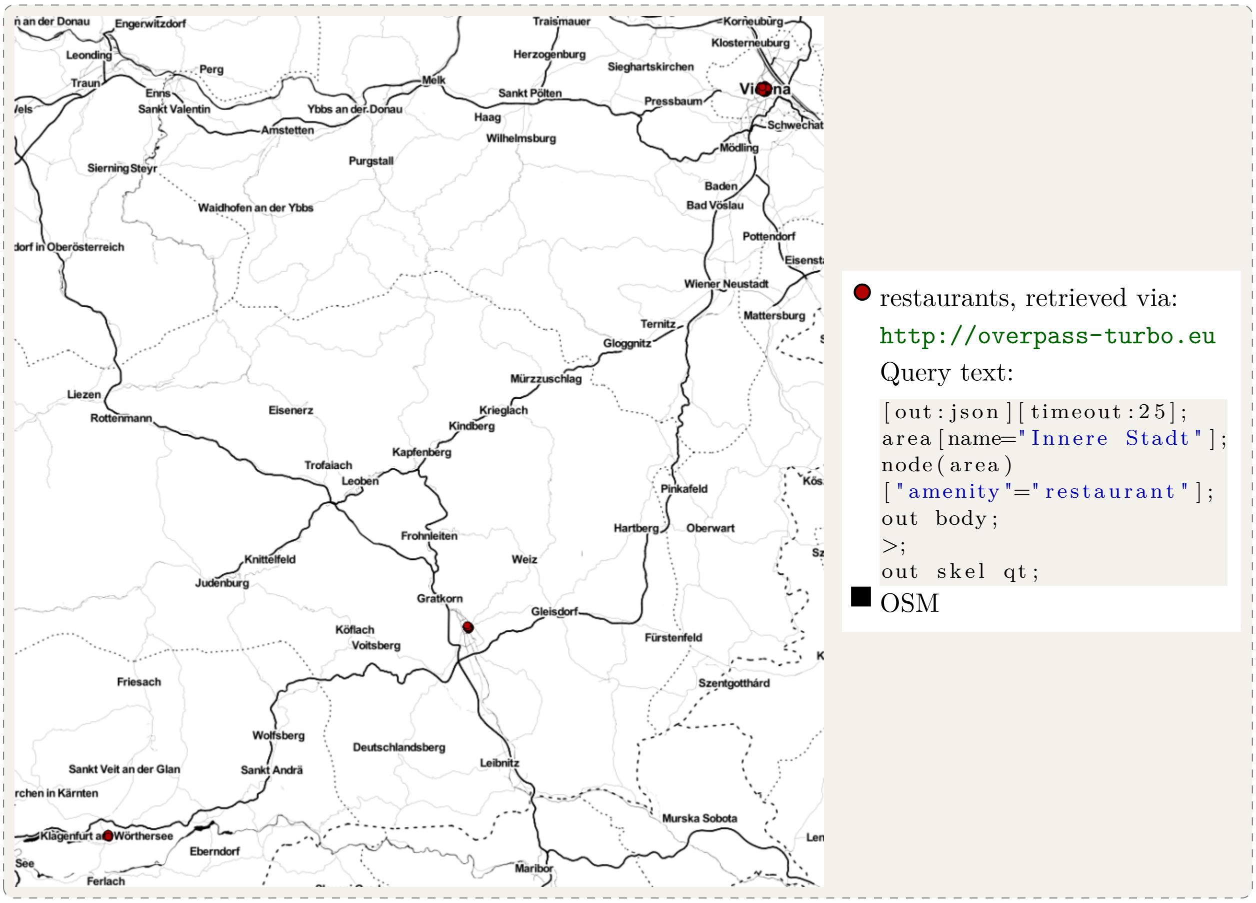
Figure 1.2: Map showing restaurants in Vienna

humans include already presented contexts in a conversation, and take them as basis for further conversation [Fauconnier, 1994]. To be able to consider both such past context and actual context, a combination mechanism for contexts has to be found.

The result of a combined context has to be another context, including the constituents which also act as selectors. In our example, the constituent contexts *in Vienna* and *Innere Stadt* are combined to create the new context *in Vienna in Innere Stadt*. The name *First District* is assigned to the combined contexts by Vienna residents. Combined context do not necessarily have to be assigned new names. For example, the contexts *in Vienna* and *serving pizza* are not assigned a new name in common usage, although these two do result in the context *in Vienna serving pizza* displayed on the map in Figure 1.4, even further reducing the references down to 140 restaurants. The combination mechanism for context is the basis for modeling the referring process, as multiple contexts may act simultaneously.

A context is also a possible complement of another context. For example, suppose our two subjects want to meet at a restaurant that is *not* serving pizza; then that context is a complement of the previously used context, *serving pizza*, resulting in the context *not serving pizza*. Applying the complemented context to the statement will result in: "Let's meet at a restaurant in Vienna that does not serve pizza".

Context influences the referring process by reducing the number of references from words to objects in reality. For our example statement, the goal is to select only one reference/object in reality where the meeting will take place. In order to reduce the references to only one restaurant, more context has to be introduced. Typical additional con-



● restaurants, retrieved via:

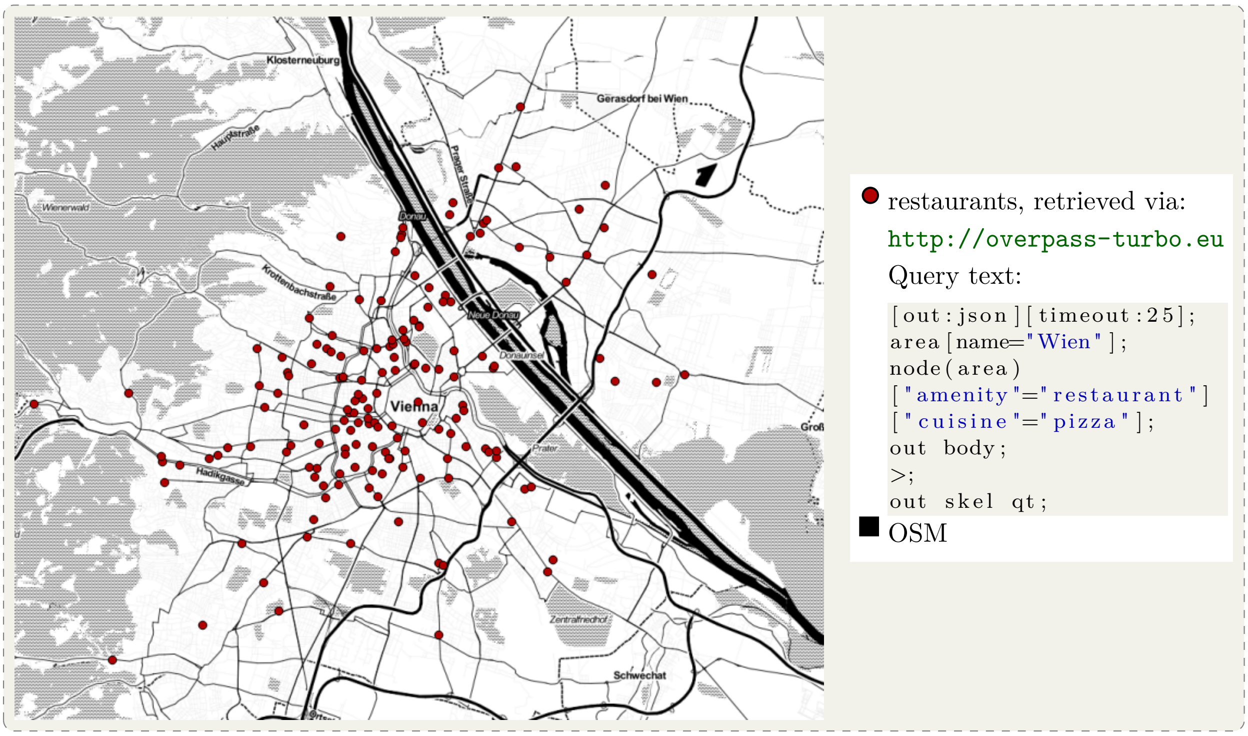
http://overpass-turbo.eu

Query text:

```
[out:json][timeout:25];
area[name="Innere Stadt"];
node(area)
["amenity"="restaurant"];
out body;
>;
out skel qt;
```

■ OSM

Figure 1.3: Map showing restaurants for *Innere Stadt*.



● restaurants, retrieved via:

http://overpass-turbo.eu

Query text:

```
[out:json][timeout:25];
area[name="Wien"];
node(area)
["amenity"="restaurant"]
["cuisine"="pizza"];
out body;
>;
out skel qt;
```

■ OSM

Figure 1.4: Map showing restaurants *in Vienna serving pizza*.

texts would be time, *e.g.* *at midnight for food*, or transportation mode, *e.g.* *by car* or *on foot*. As demonstrated above, different contexts can be ordered and combined – but what exactly should be considered a context?

1.2 What is "Context" in this example?

The word context originates from the Latin term *contextus* (past participle of contextere), meaning *to weave together or to join together* [Cobuild, 1987]. Colllins dictionary includes two synonyms for context: *circumstances* and *frame of reference*. According to Loyola [2007] context has two acceptations: (i) "the parts of a discourse that surround a word or passage and can throw light on its meaning" and (ii) "interrelated conditions in which something exists or occurs (Webster's English Dictionary) and that these acceptations relate to discourses, situations or events which are common to any organization". Having said that, Loyola [2007] concludes that this does not, however, shed light on what context is.

Every discipline and almost every application define context differently [Bazire and Brézillon, 2005]. The most-cited definition in the research field of ubiquitous computers relies on the definition by Dey [2001]: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". In an article from 1996 [McCarthy, 1996] McCarthy refuses to give a definition for context, but later on, in 1997, he defines context as "a generalization of a collection of assumptions, but there are important differences. For example, contexts contain linguistic assumptions as well as declarative and a context may correspond to an infinite and only partially known collection of assumptions" [McCarthy and Buvac, 1997]. In the field of linguistics, Sanders [1988, page 15] provide the following definition: context is a "... set of premises used in interpreting an utterance"; more specifically, "[a] context is a psychological construct, a subset of the hearer's assumptions about the world". In the field of GI-Science [Goodchild, 1990, Egenhofer et al., 2015], Kuhn [2005] states that "[c]ontext is an overloaded term and has many aspects". To generate common ground, Bazire and Brézillon [2005] extracted common factors from 150 context definitions within cognitive science. The most common factors included in the definitions are: attention, activity, situation, environment, and an observer. Ultimately, however, the study concludes that a common definition for context cannot be found: "... context stays a very ill-defined concept" [Bazire and Brézillon, 2005].

In this thesis, context is considered as it occurs in the communication process, where the receiver requires context to understand the meaning of the words supplied by the sender. The given operational definition is influenced by a statement from Leech [1981, page 66]: "[context] has the effect of narrowing down the communicative possibilities of the message as it exists in abstraction from context". In this thesis:

> CONTEXT IS ANY INFORMATION THAT SELECTS APPROPRIATE REFERENCES FROM A WORD TO OBJECTS IN REALITY.

1.3 A General Context operation

A proposal for a *general context operation* explaining the function of context is outlined. The general context operation is inspired by the constitutive rule for institutional facts, presented by the philosopher Searle [1995]. In order to distinguish between several meanings, Searle formulates the constitutive rule "X counts as Y in C" [Searle, 1995, page 43], illustrated in the lower part of Figure 1.5. He expresses this rule in an example considering a piece of paper in a specific format, with signs and a particular number on it. Applying Searle's statement to this piece of paper, the piece of paper (X) counts as currency (Y) in the context of the appropriate geographic region (C). The piece of paper in Figure 1.5 with the € symbol on it counts as currency only in Europe, while in the U.S. it would be a mere piece of paper, as illustrated in Figure 1.5. This example seems simple, because everyone can go to a bank and exchange one currency into another. It was designed from Searle to explain the constitutive rule and how context works.

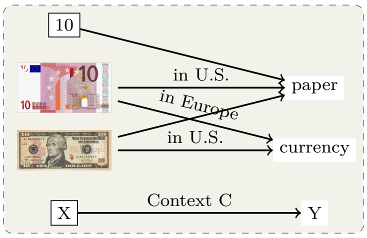


Figure 1.5: Idea of a General Context Operation

The general context operation is intended to formalize behavior, as per the constitutive rule stated by Searle. A vague shape for a general context operation was given by Shoham [1991]. He states that "... among those who talk about context to consider operations that accept a context and some other information, and produce another, usually more specific context". Bringing together the ideas of Searle [1995] and Shoham [1991] and the introductory example, a general context operation is formalized in pseudo-code in Algorithm 1.1.

```
function GENERALCONTEXTOPERATION(information, context)
    do
    moreSpecificContext = makeMoreSpecificContext(context)
    moreSpecificInformation = createInfo(information, moreSpecificContext)
return moreSpecificInformation
```

Algorithm 1.1: Pseudo code of an idea for the General Context Operation

Algorithm 1.1 uses two additional functions: (i) one that creates a more specific context; and (ii) another function that uses this more specific context to build more specific information. The idea of a general context operation is vague, but in the course of this thesis the parts will become clearer. The goal, ultimately, is to create a general context operation.

1.4 Hypothesis and Research questions

In this thesis the following hypothesis is evaluated:

A GENERAL CONTEXT OPERATION SELECTS REFERENCES FROM A WORD TO OBJECTS IN REALITY.

This hypothesis in conjunction with Algorithm 1.1 yields a series of research questions that are addressed in this thesis. By analyzing the general context operation, the following research questions appear:

1. *What properties does context have that a general context operation must respect?*
   The function `makeMoreSpecificContext` takes a context and creates a more specific context. What functionality is necessary to create a more specific context? Can algebraic laws be claimed that such a function must respect? Is there only a single more specific context, or are there several?

2. *How can a general context operation be connected to words and to reality?*
   The function `createInfo` takes information and context, and creates more specific information. This raises the question of how context is connected to information. For example, in the introduction example context is applied to the word "restaurant". It remains open whether this is information or a word, and how it relates to reality. What is meant by information? How does a formalization look like that respects context properties and is able to connect to a word and to objects in reality?

3. *What is necessary to implement the general context operation, assuring all context properties are respected, and what class of complexity can be determined?*
   Can the proposed formalization defined in the second research question be implemented? How can it be assured that the implementation respects the properties stated for context? In what complexity class is the implementation categorized?

4. *How can data be acquired that respect a connection between a word and reality?*
   Information used in `makeMoreSpecificContext` has to first be acquired. Which usable methods exist to acquire data observable from reality? Are these methods based on sensor data or questionnaires, or is it a manual process?

5. *What services can be generated by using a general context operation for natural language?*
   Is the formalization able to establish a translation mechanism from natural language to metric distances? Initializing the formalization with acquired data, what services for natural language interfaces for GIS can be created?

1.5 Research approach

In order to evaluate the hypothesis, the research questions are answered in sequential order. How these research questions are answered is outlined below.

Research Question 1: *What properties does context have that a context formalization must respect?* The goal of this research question is to identify algebraic properties of context. Existing formalizations are reviewed, with the latest approach reviewed in detail in Section 3.

Research Question 2: *How can a context formalization be connected to words and to reality?* To answer this question, a context formalization is built, and connected to a word and to reality. First, the context formalization is introduced using the conclusions of the preceding research question. In the next step, it is connected with a model for a word. Lastly, the word is formalized by a concept formalization to connect it to reality. The answer to this research question is presented in Section 4.

Research Question 3: *What is necessary to implement the general context operation, assuring all context properties are respected, and what class of complexity can be determined?* This question includes three related questions regarding the implementation. First, the formalization including the general context operation in Section 4 has to be implemented, which is achieved in Section 5. Second, the implementation is proven to satisfy all the properties assumed for context in Section 6. Third, the formalization is executed with test data to determine the complexity class with respect to memory and time consumption in Section 7.

Research Question 4: *How can data be acquired that respect a connection between a word and reality?* How data can be acquired is presented through the example case of the word “near”. In order to answer the question, several existing methods are discussed and evaluated by how well they satisfy the requirements needed to initialize the general context operation. The answer to this research question is outlined in Section 8.3.

Research Question 5: *What services can be generated by using a general context operation for natural language?* By initializing the implementation with data, example cases are presented that bridge the gap from natural language input to metric distances, a necessity for spatial algorithms as outlined in Section 8.5. Additionally, the model is able to reproduce results shown by cognitive research for vague spatial terms included in Section 8.5.

1.6 Contributions

- The identification of algebraic properties for context from former formalizations contributes to a general understanding of context. The detailed review that determines properties hidden in formalizations sheds light on all context properties assigned in former formalizations. This contributes to wider usage, and enables possible future formalizations that rely on these properties.
- The formalization of the identified properties in a context algebra abstracts from details. The context algebra is general, that is, it can be applied outside GIScience. This contributes to a better understanding of context, and may provide an impetus for interdisciplinary research.
- The implementation outlines interfaces for the integration into existing systems.
  - The implementation shows how a context algebra is realizable in a programming language.
  - The implementation can be used as a test case to evaluate further formalizations against it.
  - The implementation is publicly available in the appendix and on the internet, so that everyone can download an executable version of the code.
  - The code is enriched with documentation that gives hints for the decisions made in the implementation.
- The context algebra is applied for the example case of "near", where data are necessary to initialize the implementation. Contributions are:
  - A review of data acquisition methods.
  - A list of requirements for data in order to be useable for the context algebra.
  - A method of how to acquire data from existing data sets.

  These can be used to develop methods for automatic data acquisition, or they can be used as data modeling guidelines for systems influenced by context.

1.7 Outline

The thesis is structured in nine chapters; some of the chapters correspond with the stated research questions.

Chapter 2 reviews models for the referring process, in order to point out where context has an influence, and which additional elements have to be considered. The review of models for the referring process shows the concepts that have to be considered; both general former concept formalizations as well as those for usage specifically in the field of GIS are reviewed. The conclusions point out the elements that have to be considered in order to model the reference process, and a review of existing formalizations for these elements is given.

Chapter 3 reviews a novel model for concept combination using context, which context properties are then extracted from. The model is based on mathematics applied to quantum mechanics, and is reviewed to extract context properties. The conclusions show these context properties.

Chapter 4 establishes the mathematical theory of a context algebra. Relations, operations, and properties are introduced to model context. The model is extended by a connection to concepts that create a connection to objects in reality. This connection enables the calculation of different typical exemplars of a concept, dependent on the context. The conclusions are the mathematical structure for context, and the connection to concepts which relate to objects in reality.

Chapter 5 includes an implementation for the context algebra to present its realization. The context algebra is implemented using the functional programming paradigm, and differences between mathematics and implementation are pointed out. Within this implementation, the general context operation is realized, and applied to model the introductory "restaurant" example. Conclusions are drawn from the differences between the mathematical and implementation parts, and which elements are necessary to formalize the general context operation.

Chapter 6 evaluates the implementation by how well it satisfies the properties stated for the context algebra in Chapter 4. Automated algebraic tests are generated, which assure that all properties stated in the context algebra are satisfied by the implementation. This is further proven by executing the implementation with randomly generated input values. The conclusion that the implementation satisfies all algebraic properties is drawn from the generated reports of these test executions.

Chapter 7 evaluates the implementation to show the number of memory, and execution time needed for several numbers of randomly generated context. The file sizes and execution time required to establish a context formalism, as well as the time needed to calculate different typical exemplars are measured. The measured values are compared to the theoretical complexity of the context algebra. It is concluded that the theoretical complexity of the context algebra is recovered in the measurements.

Chapter 8 shows an application of the implementation for the word *near*. The goal of the application is to distinguish between several meanings of "near", *e.g.* near future, near friends, near tram station. For the geographical meanings, a translation mechanism to metric distances is created. Data sets that can address all predefined requirements are selected, and a web application is built to process the data that are to be used in the context algebra. Conclusions from the modeling of "near" and the demonstrated applications show that the context algebra establishes a translation mechanism from natural language to metric distances.

Chapter 9 contains the overall conclusion of this work, and lists future research directions. In the conclusion, the research questions are answered and the hypothesis is evaluated. From the lessons learned, possible future work is envisioned.

Appendix A includes the code for the implementation of the context algebra, the general context operation, and an example. Additionally, the modules needed to evaluate and benchmark the implementations are included as well. Measurements of the benchmarks are also included.

Appendix B contains the implementation needed to acquire the data for "near" and shows the results produced by the context algebra.

2
Related Work

This chapter reviews models for the referring process, trying to identify the included elements and their formalizations with respect to context. A naive idea of a referring process is given in Figure 2.1, where a word refers directly to an object in reality. Models from the field of semiotics also include concepts, integrated in the referring process through an agent. Existing models for concepts with respect to context influence, as well as their usage in GIS, are reviewed. Models for context are reviewed in order to identify context properties.

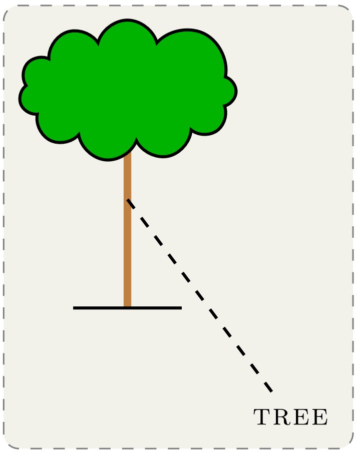


Figure 2.1: Idea of a referring process

2.1 Elements included in the referring process

The *referring process* relates a word to objects in reality. This process is in the focus of semiotics [Chandler, 2007, page 2]. In general, semiotics is the study of symbols or signs [Chandler, 2007, page 1], and two models for the referring process are of particular interest here. One of these models was introduced by de Saussure [1916], the other by Peirce [1931]; they differ in that the latter includes references to reality.

Similarly to the definition of context, a unique, agreed-upon definition for sign also cannot be found. Peirce [1931, CP 2.228] understands a sign as "something which stands to somebody for something in some respect or capacity"; for Eco [1976, page 7], a sign is everything that can be taken as a sign, *i.e.* that refers to something else. In a taxonomy for sign, Chandler [2007, page 2] distinguishes: texts, images, sounds, objects, and gestures.

Saussure's model For de Saussure a sign connects a **concept**[1] (mental representation) and a word (sound-image) [de Saussure, 1916, page 67], as expressed in Figure 2.2.

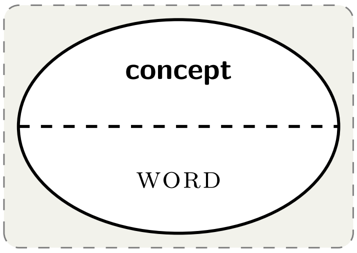


Figure 2.2: Adapted model of a sign by De Saussure [1916, page 67]

The relation between concept and word is arbitrary [de Saussure, 1916, page 67], and the cardinality of the relation is not a one-to-one relation. For example, the word BANK can stand for the concept **financial bank**, **river bank**, etc. In the perspective of humans, this cardinality is a reduction of memory usage because if every concept were expressed by exactly one word it would "... exceed our capability of learning, recalling and manipulating" [Chandler, 2007]. This cardinality is in contrast to the cardinality used in databases, where each database entry has its own identifier attached to it. The lack of

[1] In this work a concept is highlighted in bold font, *e.g.* **concept**, to distinguish it from the word it refers to, which uses small caps, *e.g.* WORD.

a one-to-one relation creates uncertainty as to what a given word is actually referring to. De Saussure also observes this uncertainty and states that the uncertainty is removed by considering surrounding signs [Chandler, 2007, page 18], as illustrated in Figure 2.3, also known as the *interpretation process*.

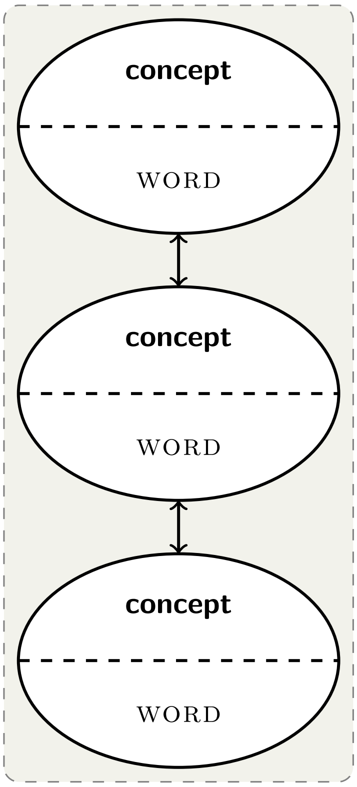


Figure 2.3: Interpretation process of a sign according to De Saussure [1916, page 67]

As was pointed out above, the word BANK can refer to both the concepts **financial bank** or **river bank**. Surrounding the word with a word like FINANCIAL, the relation to the concept **river bank** is released, while the relation to **financial bank** remains. In the perspective of the present work, surrounding signs are considered as context and it is assumed, that the effect of context is a selection of relations between concepts and words. Leech [1981] describes the effect of context as "narrowing down the communicative possibilities of the message as it exists in abstraction from context."

Using de Saussure's model to formalize concepts, we face the problem that it only uses signs, and that references to reality are missing. De Saussure's model was used by computer scientists in Artificial Intelligence (AI) research to represent concepts as a system, built from algorithms and symbols. It became prominent as the symbolic model of mind. If this symbolic system is "capable of generating behavior indistinguishable from that of a person", Harnad concludes that the symbolic model "... must have a mind". Searle [1980] counters this view with the *Chinese Room Argument*. Searle explains this argument as follows: "Imagine a native English speaker, let's say a man, who knows no Chinese locked in a room full of boxes of Chinese symbols (a data base) together with a book of instructions for manipulating the symbols (the program). Imagine that people outside the room send in other Chinese symbols which, unknown to the person in the room, are questions in Chinese (the input). And imagine that by following the instructions in the program the man in the room is able to pass out Chinese symbols that are correct answers to the questions (the output). The program enables the person in the room to pass the Turing Test for understanding Chinese but he does not understand a word of Chinese." That lack of understanding is referred to as the *symbol grounding problem* [Harnad, 1990].

The Chinese Room Argument states that algorithms (programs) only have syntax, and symbols are arbitrarily chosen. AI methods were changed to include sensors that provided "appropriate and often enough" [Brooks, 1999] observation of the world, resulting in *embodied AI*. In the field of GIScience, Kuhn [2005] suggests using the second model from the field of semiotics, as established by Peirce [1931], to overcome the symbol grounding problem.

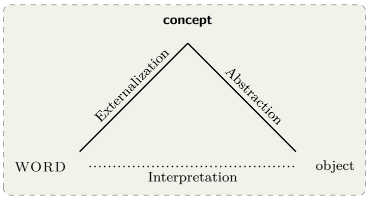


Figure 2.4: Model of a symbol by Peirce arranged as triangle by Ogden and Richards [1946]

Peirce's model For Peirce [1931, CP 2.228], a sign is comprised of an object, a WORD, and a **concept**, arranged in a triangle by Ogden and Richards [1946]. The *object* stands for what is represented, the WORD for how it is represented, and the **concept** for how it is interpreted [Chandler, 2007, p 29][2]. The WORD is similar to the WORD in Saussure's model, but the **concept** is divided further into a mental part

[2] Note: many variants of this terminology exist.

and an object part.

Ogden and Richards [1946] presented the three elements of a sign, arranged as triangle called a *semiotic triangle*, as shown in Figure 2.4. The most important difference of the semiotic triangle to the model of de Saussure [1916, page 67] is that the object can be a place outside the sign system. The object can relate to physical reality, abstract concepts, and fictional entities [Chandler, 2007, page 33].

The semiotic triangle defines three relations between a word, a concept, and an object. The relation between an object and a concept is called *abstraction*. The relation between a concept and a word is called *externalization*. And the dashed line, called *interpretation*, reflects the fact that there is not necessarily a connection between a WORD and an object. For example, the word BANK can be interpreted at least as a river bank or a financial bank as presented in Figure 2.5.

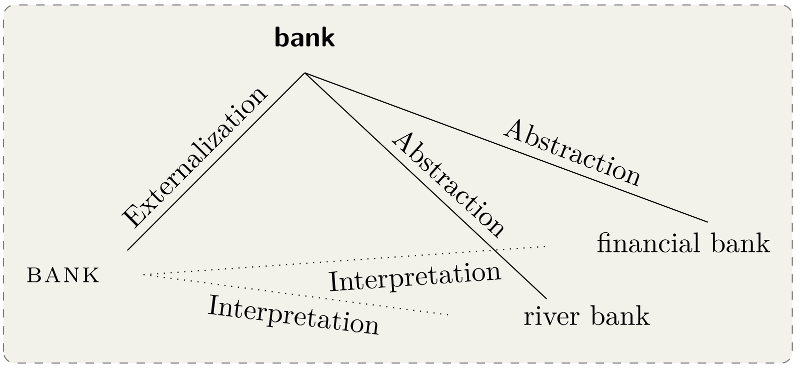


Figure 2.5: Semiotic triangle for the word BANK, referring to river bank and financial bank

The cardinality of the interpretation relation is many-to-many, which means words can relate to multiple objects. This cardinality is a result of the cardinalities given by the abstraction relation and the externalization relation. The abstraction relation is a many-to-one relation, as many objects in reality are abstracted to one concept. The externalization relation is a one-to-many relation, as one concept can be expressed by multiple words. By using the abstraction relation and the externalization relation to establish the interpretation relation, the interpretation relation must also have a cardinality of many-to-many.

To reduce the many-to-many relation for the interpretation relation, context is used to select intended relations. The relations are from object in reality to a word via a concept.

Concepts

The nature of concepts, *i.e.* what they really are, is subject of much debate [Margolis and Laurence, 2014]. What can be agreed on is that "[c]oncepts are the constituents of thoughts" [Margolis and Laurence, 2014] and "... provide a taxonomy of things in the world", as Smith and Medin [1981, page 8] reviewed Woods work. Barsalou et al. [1993, page 1] assume "... that concepts are people's psychological representations of categories". "Consequently, they are crucial to such psychological processes as categorization, inference, memory, learning, and decision-making" Margolis and Laurence [2014]. Smith and Medin distinguished between two functions for concepts. First, a *categorization* function, which determines if an object is a member of a concept. Second, a *concept combination* function, to enlarge the taxonomy.

The categorization function is included in the semiotic triangle as an abstraction relation. Models for the categorization function are defined as similarity formalizations and explanation-based formalizations, and are reviewed in Section 2.2. This review highlights the incorporation of context and their application in GIS.

Experiments investigating concept combination showed results which existing formalizations were not able to represent. A particular experiment shows the *guppy effect*, explained in Section 2.5. An approach reproducing these experimental results is proposed by Aerts and Gabora [2005a] using context extensively. This usage of context is reviewed in Chapter 3 to extract context properties usable for the context algebra.

Spatial Concepts

Representations of spatial concepts have to respect semantic and structural aspects [Freksa and Barkowsky, 1996] of spatial concepts. Spatial concepts are "notions that describe spatial aspects of a subset of the world" [Freksa and Barkowsky, 1996]. Examples include *e.g.* **mountain**, **near**, and **lake**. Spatial concepts have vague boundaries, such as *e.g.* **lake**, **ocean**, **regions** [Montello et al., 2014], **downtown** [Montello et al., 2003]; are size- or scale-dependent, *e.g.* **pond**, **lake**, **ocean** [Smith and Mark, 1998, Mark et al., 1999]; and differ across languages *e.g.* **pond - étang** [Mark, 1993].

Spatial concepts inherit multiple interpretations, determined by three types of relation [Freksa and Barkowsky, 1996]. These relations are: " (1) between concepts and physical objects, (2) between concepts and related concepts (context of discourse), [and] (3) between objects and situation context". This indicates that context has an effect on the interpretation of concepts, as evidenced by the following experiment. Subjects were asked to list typical objects for the concept **geographic space**. The listed objects differed according to the influencing context. If influenced by the context *city*[3], subjects interpreted the concept **geographic space** in objects such as: *streets, buildings*, and *parks*, etc. In contrast to that interpretation, when influenced by the context *country*, the listed objects includes *mountains, lakes*, and *rivers*, etc. [Egenhofer and Mark, 1995]. In conclusion, this suggests that a relation between a concept and a context exists, and that context influences the interpretation of a concept.

2.2 Concept formalizations

Concept formalizations enriched with context are built out of experiments conducted by cognitive scientists to understand the relation between the world and the mind [Varela et al., 1997, page 4], and how

[3] To indicate a context, it is formatted in sans-serif and italics in this thesis

mental processes affect a subject [Fodor, 1994].

Two general views on concept formalizations can be distinguished: similarity-based views, and explanation-based views. Both views have advantages and disadvantages, and are able to model different experiments. In the following section, both views and their formalizations are discussed and compared. If experiments indicated the context dependence of the formalizations, the context-enriched versions are also reviewed.

Similarity-based formalizations

Similarity-based formalizations use features of previously experienced exemplars to determine whether a new exemplar can become part of an existing concept or not. Several theories have been proposed, which introduced various formalizations. Each formalization fits one or many experiments, but none of them are able to model all of the experiments. The observed influence of context resulted in the integration of context into the formalizations. For most of these formalizations, applications in the geospatial domain exist and are also included here.

The *classical view* dates back to Aristotle and propagates that *necessary* and *sufficient* features have to be met by an exemplar in order for it to be included in a concept. As a consequence, concept membership is a Boolean rating function [Komatsu, 1992]. For example, the concept **dog** can be defined by the following features: has four legs, barks, etc. An animal with four legs that barks is a dog, but is my neighbor's barking animal with only three legs a dog? In logic, necessary and sufficient conditions are commonly used, but the definitions of "necessary" and "sufficient" themselves show "systematic ambiguity" when applied to natural language conditionals [Brennan, 2012], which makes their use difficult. In this classical view, the defining features included in a rating function are of central importance.

A formalization of the classical view is Formal Concept Analysis (FCA). FCA creates concept hierarchies using *necessary features*. The aim of formal concept analysis is to develop mathematical models that are appropriate for a conceptual structure [Wille, 2005]. Wille follows the philosophical tradition that a concept is constituted by its *extension* (the concept members) and by its *intension* (the features), which apply to all objects of the extension. For FCA, the features ("formal attributes") distinguish concepts that are arranged in a hierarchy with other concepts, meaning that subconcepts inherit all attributes of the super-concept. The hierarchy is generated by using a table called *formal context*, thereby obtaining the extension and intension of the concept [Wille, 2005]. From a formal context, a concept lattice is obtained. One such concept lattice for the concept **body of water** is shown in Figure 2.6. FCA is used in practical work by software engineers, linguists, information retrieval engineers etc. [Wille, 2005, Preface].

For the geospatial domain, FCA is adapted by Frank [2006] to generate a so-called Taxonomic Lattice. Frank changed the "formal at-

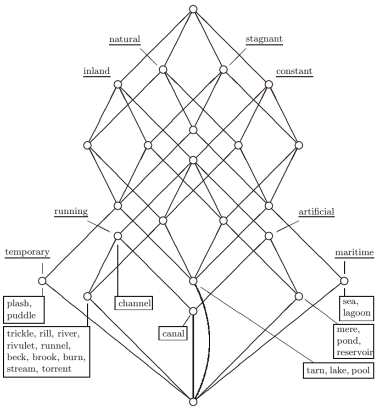


Fig. 2. Concept lattice of the formal context in Fig. 1

Figure 2.6: Concept lattice for the concept **body of water** by Wille [2005]

tribute" used by FCA to a *is_a* relation. This relation is used to categorize same entities, called taxa, which include same features. These taxa build hierarchies where *distinctions* can be added, or two taxa can be merged.

Prototype theory is mentioned by Wittgenstein [1953] and justified by experiments conducted by Rosch and Mervis [1975]. Wittgenstein [1953] stated that concepts do not have features which are used to decide membership (classical view). Rather, membership of an exemplar is determined by a function of family resemblance. This statement was confirmed by experiments conducted by Rosch and Mervis [1975]. Rosch and Mervis observed that each exemplar of a concept shares at least one feature with other exemplars included in this concept, called graded membership. At least this information is captured and abstracted from concept members to identify a typical exemplar of a concept [Komatsu, 1992]. This typical exemplar is called a prototype, hence the name prototype theory. For example, the concept **pet** inherits many exemplars (*e.g.* cat, dog, rabbit, etc.) where the prototype is cat; the typicality is illustrated as distance from the mid point in Figure 2.7, where typicality data are taken from data collected with a questionnaire [Aerts and Gabora, 2005a]. Further studies indicate that prototypes are classified similarly compared to other members [Hintzman and Ludlam, 1980] after a retention interval, meaning that prototypes are more resistant to forgetfulness. Prototype theory is similar to the classical view because both views are based on a membership function built of features [Komatsu, 1992].

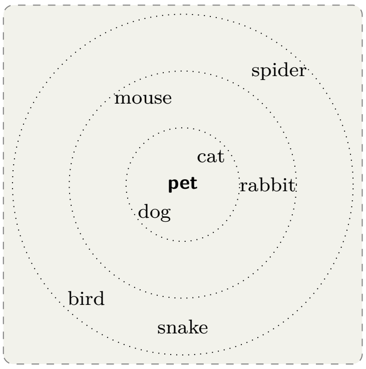


Figure 2.7: Graded membership for exemplars of the concept **pet**

Prototypes are included in the membership function that distinguishes concept members from non-members [Minda and Smith, 2011]. The membership function is a similarity measure that calculates the typicality of an exemplar in comparison to the features of the prototype. The exemplar can become a member of a concept when it is highly typical.

In the geospatial domain, similarity measurement functions are applied to rate the semantic similarity of two exemplars, for example in *geographic information retrieval*. The approach by Rodríguez et al. [1999] uses a matching distance model to calculate the semantic distance of two exemplars e_1 and e_2, shown in Equation 2.1. The function includes the features: parts S_p (*e.g.*, building features: windows, roof), functions S_f (*e.g.*, the function of a cinema is: entertainment), and attributes S_a (*e.g.*, additional features: building height). These features are equipped with corresponding weights w_p, w_f and w_a. These weights cannot be fixed because they are influenced by context [Tversky and Gati, 1982]. Rodríguez and Egenhofer [1999, 2004] therefore presented formulas that can adjust the weighting factors according to context. The context-enriched semantic similarity measure detected three out of four cases when compared to human judgments [Rodríguez and Egenhofer, 1999]. One open question that remains is how the weights and contexts for the similarity measure function are to be determined. Janowicz et al. [2010] introduced an approach in which pre-calculated similarity rankings are adjusted by users. From the adjustment, the implied contexts are learned and added as extra weights. With the application of similarity measures in geographic information retrieval Janowicz et al. [2011] introduced a "generic framework for semantic similarity measurement" pointing to the role of context.

$$S(e_1, e_2) = w_p \cdot S_P(e_1, e_2) + w_f \cdot S_f(e_1, e_2) + w_a \cdot S_a(e_1, e_2) \qquad (2.1)$$

Figure 2.8: Matching distance equation for two exemplars, according to Rodríguez et al.

Fuzzy set theory is an approach to model prototype theory. A concept is represented as a fuzzy value in the range $[0, 1]$. This value is also a typicality measure of an exemplar [Zadeh, 1976]. This model is able to represent variations of a concept; for example, the concept **young** has variations such as **very young**, **not very young**, or **old**, as demonstrated in Figure 2.9.

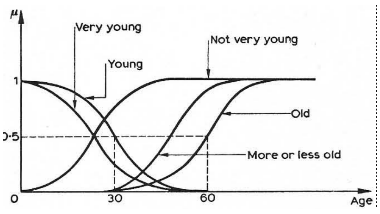


Figure 2.9: Fuzzy value for the concept **young** and its variations by Zadeh [1976, Figure A1]

Fuzzy sets are used to model spatial concepts in the geospatial domain. Fuzzy sets represent terms that are included in a query language, *e.g.* the concept **close** for a query "Find all the cities which are close to at least several camping sites" [Wang, 1994]. Using this representation, Wang [1994] concluded that an influencing context may change the fuzzy set representation, and proposed different fuzzy sets according to context, *e.g.* **close**$_{walking}$, **close**$_{driving}$.

Conceptual spaces Gärdenfors uses a multidimensional metric feature space to calculate prototypes. The approach treats every feature as *separable* and models the features as dimensions in a vector space. The dimensions are linear combinations representing one object. Gärdenfors [2004] proposed this metric approach to calculate prototypes using a Voronoi tessellation. Gärdenfors claims that similarities can be calculated by a distance function in the metric space between two exemplars $d(e_1, e_2)$. The distance function has to respect symmetry properties and the triangle equation as given in Equations 2.2.

$$\begin{aligned} & d(e_1, e_2) >= 0 \text{ and } d(e_1, e_2) = 0 \text{ only if } e_1 = e_2 \\ & d(e_1, e_2) = d(e_2, e_1) \\ & d(e_1, e_2) + d(e_2, e_3) >= d(e_1, e_3) \end{aligned} \qquad (2.2)$$

Conceptual spaces were applied to represent *landmarks* for data descriptions and similarity measures in the geospatial domain. Raubal [2004] used this technique to represent spatial landmarks. He argues that difficulties regarding similarity and the triangle equation raised by Tversky and Gati [1982] can be avoided if different weights are given according to context, and contexts are not mixed. In an extended approach to overcome the difficulties, Schwering and Raubal [2005] included the shape, size, and distance of the metric space a concept inherits in their similarity calculation. Keßler [2006] used conceptual spaces to show how to generate data descriptions adequate for human cognition. Conceptual spaces were applied by Schwering and Kuhn [2009] to build a hybrid similarity measure to account for the complex semantics included in spatial data.

Exemplar view Nosofsky [1986] uses already-present exemplars in a concept to decide if a new exemplar can become part of the concept. In contrast to the prototype view, a concept is represented by multiple exemplars [Nosofsky, 2011] and not only by a prototype. Experiments showed good results when exemplars are "... retrieved from long term memory depend[ent] on context, goals, prior processing, frequency of occurrences or retrieval, time of last retrieval" [Komatsu, 1992, page 508]. Regarding confidence in the classification process, subjects are more confident if they classify the prototype compared to other exemplars [Komatsu, 1992]. In general, the exemplar view uses the same approach as the prototype view, with the only difference being that the similarity measure takes into account multiple exemplars.

Explanation-based formalization

Explanation-based formalization emerged from the results of experiments with infants, where differences between concepts of infants and adults were recognized [Carey, 1985]. These findings led to the idea that concepts change over time, similar to scientific theories which are accepted or revised depending on experimental results and observations [Gopnik and Meltzoff, 1997]. Siegler [2002, page 34] states that "knowledge moves consistently from less to more advanced ...", and illustrates it as overlapping waves of theories (Figure 2.10).

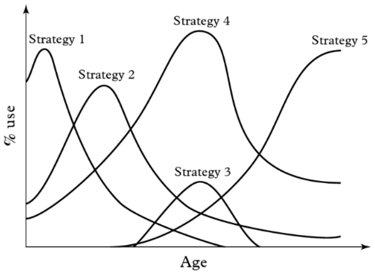


Figure 2.10: Wave model of overlapping theories by Siegler [2002, page 34]

Theory theory Gopnik and Meltzoff [1997] introduced theory theory, distinguishing three features of theories [Gopnik and Meltzoff, 1997, page 32-41]. First, structural features lead to an ontological commitment about the world. Second, functional features predict events. Third, dynamic features compare predictions with reality. Summarizing the view of theory theory on concepts, they are structured representations relating to other concepts [Laurence and Margolis, 1999] that are accepted or revised.

Twaroch and Frank [2005] used theory theory to describe the change of spatial theories for infants. Twaroch and Frank used three mechanisms to change theories: specialization, generalization, and dynamic weighting. Within these mechanisms, theories are formalized as a lattice structure. The specialization is described by a step down in the lattice, and the generalization by a step up. Dynamic weighting is used to choose between comparable theories.

2.3 Context formalizations

Previous approaches to formalize contexts were inspired by technological developments, *e.g.* ubiquitous computing, reasoning, ontology, and generality in a knowledge base inheriting common knowledge for Artificial Intelligence.

Data-structure-driven formalizations

Mobile computers [Weiser, 1991] are used in different contexts, which leads to several context formalizations integrated in software. One of the first context formalizations was based on a *key-value model* proposed by Schilit et al. [1994]. Held et al. [2002] defines requirements for context models as: structured, interchangeable, composable, uniform, extensible, and standardized. Several of these requirements are satisfied through the use of *markup scheme models* [Loyola, 2007]. Cyganiak et al. [2014] introduced a schema for context in the Resource Description Framework (RDF), and concluded that it satisfies all the requirements. In summary, the proposed models structure context in a human-readable format.

Brézillon et al. [2002] introduced the *contextual graph model* that highlights context in decision trees. A contextual graph represents actions and events needed to make a decision. In Figure 2.11 an example of a contextual graph is shown for the situation "Sick traveler [o]n a train". In this figure, boxes indicate actions and circles indicate events to make a decision. Reaching an event node, the previous path is considered as context and has to be included in the decision-making process.

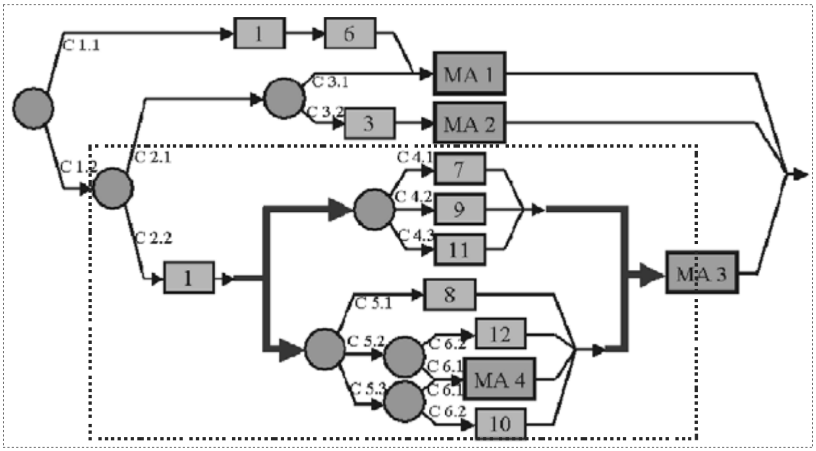


Figure 2.11: Contextual Graph of the situation "sick traveler on a train" according to Brézillon et al. [2002]

Contextual reasoning

Giunchiglia [1993] introduced *contextual reasoning* to perform reasoning within context. This formalization assumes that humans use a subset of a global knowledge base to do reasoning, where context defines the subset of knowledge [Bouquet et al., 2003, Giunchiglia, 1993]. Bouquet et al. [2003] categorized this context usage as compose-and-conquer. They described it with a box metaphor, where the knowledge used to reason is put in the box, the reasoning is applied, and the result is presented as illustrated in Figure 2.12. Such a reasoning process is described in Algorithm 2.1. The context is specified as *Morning* (SWITCH CONTEXT) and from that point the knowledge taken into account for reasoning is fixed to morning. At the end of the reasoning

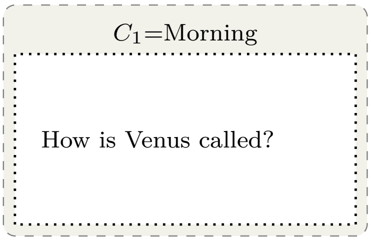


Figure 2.12: Context as box metaphor by Giunchiglia

process, the results valid in this context are given; in the example case this would be “morningstar”. With respect to algebraic properties, Giunchiglia [1993] observed a partial order relation for context.

Algorithm 2.1: Pseudo code of a contextual reasoning system, related to

```
SWITCH CONTEXT "Morning"
ActualVenusName = inferActualNameForPlanet Venus
RETURN ActualVenusName
```

Ontology Strang et al. [2003] introduced ontology-based context models that are equipped with a reasoner using Context Ontology Language (CoOL) [Loyola, 2007].

Ontology-based models are used for mediation and *geospatial business intelligence* in the geospatial domain. Cai [2007] presented a “framework for semantic interoperability of geospatial information” that interprets vague spatial concepts in the interaction between GIS and users. Diallo et al. [2015] present a context ontology designed for mobile *geospatial business intelligence* applications, enriched with contextual metrics built from location and time.

2.4 Context as a logic model

McCarthy [1987] introduced context as a logic model to implement generality in Artificial Intelligence (AI). McCarthy envisioned that a collection of facts of reality is able to model human common-sense knowledge. The facts should be valid in general, *e.g.* “planet Venus is the Evening Star”, but as the number of facts increased, other facts generated ambiguities with existing ones, e.g. “Venus is the Morning Star”. In order to distinguish whether Venus is the Evening Star or the Morning Star, McCarthy introduced context.

In McCarthy’s perspective, context was the information necessary to distinguish between facts about the world. Such use of context was categorized by Bouquet et al. [2003] as a divide-and-conquer approach to context. Guha [1991] applied this approach to knowledge bases, and established *microtheories* as shown in Figure 2.13. The microtheories contain other microtheories given in proposition `Ist(context, inner microtheory)` and different reasoning results are achieved by using different influencing contexts, as illustrated in the left and right parts in Figure 2.13. The research contributions from McCarthy [1987, 1993] and his coworkers [Guha, 1991] are summarized in unpublished articles called “A logical AI approach to context” McCarthy [1996] and “context formalization (extended notes)” by McCarthy and Buvac [1997]. The most concise axiomatic framework from these findings was proposed before by Shoham [1991], and is reviewed below. He proposed a logic framework for context using propositional language, and extending their work by adding “. . . a few more . . . all still somewhat speculative and imprecise” propositions to that framework.

Partial order relation Context is *partially ordered* with the relation *“specialized”* by McCarthy and Buvac [1997], or by the converse *“more*

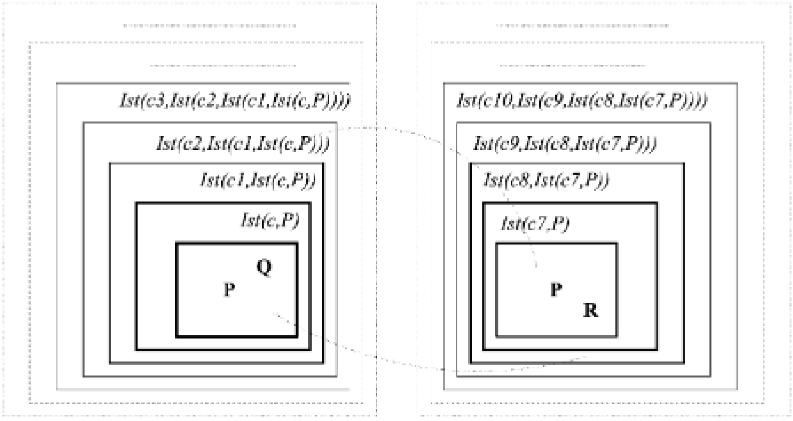

Ist(c3,Ist(c2,Ist(c1,Ist(c,P))))
Ist(c2,Ist(c1,Ist(c,P)))
Ist(c1,Ist(c,P))
Ist(c,P)
Q
P
Ist(c10,Ist(c9,Ist(c8,Ist(c7,P))))
Ist(c9,Ist(c8,Ist(c7,P)))
Ist(c8,Ist(c7,P))
Ist(c7,P)
P
R

Figure 2.13: Divide and Conquer use of context by [Bouquet et al., 2003, Fig. 1]

general than" by Shoham [1991] which will be used here. The relation states that for two contexts c_1 and c_2 the order relation shown in Equation 2.3 holds and is read as c_2 is "more general as or as general as" c_1. The order relation is partial because "not every pair of contexts are comparable under it" [Shoham, 1991].

$$c_i \in C, \quad c_1 \supset c_2 \tag{2.3}$$

The partial order relation is transitive as given in Equation 2.4.

$$c_i, c_j, c_k \in C, \quad c_i \supset c_j \quad \text{and} \quad c_j \supset c_k \implies c_i \supset c_k \tag{2.4}$$

Within the order relation, lower and upper bounds for context are included. McCarthy [1996] argues that there cannot be something like a universal context: "This is a fact of epistemology (both of the physical world and the mathematical world). It is always possible to generalize the concepts one has used up to the present. Attempts at ultimate definitions always fail—and usually in uninteresting ways. Humans and machines must start at middle levels of the conceptual world and both specialize and generalize." For McCarthy a bound is the outermost context that "is needed to give computer programs the ability to reason about the totality of all they have thought about so far [McC96]." Akman and Surav [1996] describe the acceptance of a bound context as a mathematical simplification. Shoham [1991] includes both bounds, a most general context called *tautological context*, and a most specific context called *contradictory context*.

Context operations Contexts are combined by two operations, conjunction denoted $\wedge$ and disjunction denoted $\vee$, and have a negation operation denoted $\neg$ [Shoham, 1991]. The conjunction of two contexts results in the greatest lower bound c_3 of c_1 and c_2 as given in Equation 2.5. The disjunction of two contexts results in the least upper bound c_4 of c_1 and c_2 as given in Equation 2.6. The negation operation is given in Equation 2.7.

$$c_1 \wedge c_2 = c_3 \tag{2.5}$$

$$c_1 \vee c_2 = c_4 \tag{2.6}$$

$$\neg c_i \tag{2.7}$$

The partial order relation and the context operation build a lattice structure for context. So far context was seen as stand-alone entity, but the benefit of a context formalization is to connect context with another entity, as the following quote highlights:

> "The structure of contexts is of little interest without a way of associating assertions with contexts; that is stating what is the case in each context" – Shoham [1991]

Within the connection between context and an assertion, more algebraic laws for context are included in the work of Shoham. The connection from contexts to assertions is denoted by the property p^c. This property reads as assertion p is valid in context c and is referred to here as the *context application function*. Guha [1991] implemented a context application function in his dissertation. He lists many examples how a context application function can be reinterpreted by a function like $assuming(p, c)$. To assign a value to the property, Guha introduced proposition $value(c, p)$. This proposition enables the evaluation of a value; for example, the assertion: *value*(observed at 8am, planet Venus) evaluated in the value "Morning Star", compared to *value*(observed at 8pm, planet Venus) with a changed context, evaluates to the value "Evening Star".

Shoham [1991] observes properties for this function depending on one of two different aspects: (i) properties concerning only contexts and (ii) properties concerning the context application function (context and assertion properties).

Context properties Shoham stated six properties for proposition p^c [Shoham, 1991][Proposition 9,10,11,12,13,14] reviewed here:

The first context property is called *context completeness* and is given in Equation 2.8. It states that either the proposition for context c is valid or the proposition for the negated context $\neg c$ is valid, under the assumption that both (the negated context and the context) are included in the context lattice C.

$$p^c \vee p^{\neg c} \quad \forall c, \neg c \in C \tag{2.8}$$

The second context property is called *positive reflection* by Shoham and refers to the algebraic property *idempotent*. It is given in Equation 2.9. It states that the application of the same context c multiple times is equal to a single application. A similar property called *negative reflection* is stated for the case that a proposition is negated and the same context is applied multiple times in Equation 2.10. From where this proposition follows is not included in Shoham's framework.

$$(p^c)^c \equiv p^c \tag{2.9}$$

$$(\neg(p^c))^c \equiv \neg(p^c) \tag{2.10}$$

The third context property is called *symmetry* and is given in Equation 2.11. It assumes that the sequence of influence of two different contexts c_1 and c_2 does not affect the result.

$$(p^{c_1})^{c_2} \equiv (p^{c_2})^{c_1} \tag{2.11}$$

The fourth context property is called *exponentiation* and is given in Equation 2.12. It transforms the sequential application of context into a conjunction of contexts.

$$(p^{c_1})^{c_2} \equiv p^{c_1 \wedge c_2} \tag{2.12}$$

The fifth context property is called *associativity* and is given in Equation 2.13. It states that brackets do not matter in the evaluation.

$$p^{(c_1^{c_2})} \supset (p^{c_1})^{c_2} \tag{2.13}$$

Context and assertion properties Six properties considering context in connection with assertions are stated by Shoham [1991][Propostitions 1,2,3,4,7,8]. In the following propositions, the operations $\wedge, \vee, \neg$ can refer to operations used either for assertions or for context.

The first combined property is called *assertion weakening* and is given in Equation 2.14. It assumes that assertion p is a superset of assertion r. If p is valid in context c_1, r is also valid in context c_1,

$$p^{c_1} \wedge (p \supset r) \supset r^{c_1} \tag{2.14}$$

The second combined property is called *context strengthening* and is given in Equation 2.15. It states that if assertion p is valid in context c_1 it is also valid for context c_2, if c_2 is a superset of c_1.

$$p^{c_1} \wedge (c_2 \supset c_1) \supset p^{c_2} \tag{2.15}$$

The third combined property is called *disjunctive context* and is given in Equation 2.16. It states that assertion p_1 is true in context c_1 and p_2 is true in context c_2 which is a superset of the disjunction of the assertions $p_1 \vee p_2$ in the disjunctive context $c_1 \vee c_2$. This property assumes that the context set is or-closed.

$$p_1^{c_1} \wedge p_2^{c_2} \supset (p_1 \vee p_2)^{c_1 \vee c_2} \tag{2.16}$$

The fourth combined property is called *conjunctive context* and is given in Equation 2.17. It proposes that either proposition p_1 is valid in context c_1 or proposition p_2 is valid in context c_2 and this is a superset of the disjunction of both assertions $p_1 \vee p_2$ valid in context $c_1 \wedge c_2$. This property assumes that the context set is and-closed.

$$p_1^{c_1} \vee p_2^{c_2} \supset (p_1 \vee p_2)^{c_1 \wedge c_2} \tag{2.17}$$

The fifth combined property is called *general consistency* and is given in Equation 2.18. It states that the assertion p in context c_1

and the negation of this assertion $\neg p$ in context c_1 together negated is valid.

$$\neg(p^{c_1} \wedge (\neg p)^{c_1}) \tag{2.18}$$

The sixth combined property is called *assertion completeness* and is given in Equation 2.19. It states that either p in context c_1 is valid or the negation of $\neg p$ in context c_1 is valid.

$$p^{c_1} \vee (\neg p)^{c_1} \tag{2.19}$$

The logical model approach for context includes several drawbacks that makes it difficult to apply in organizations. Loyola [2007] observes as drawbacks that the approach needs a clear schema of how to formalize context in the perspective of an organization, and that predictive calculus used for the logic model is not common knowledge. Akman and Surav [1996] state that logic formalizations are based on the premise "that when several contexts occur in a discussion, there is a common context above all of them". For a comparison of logic models, consider the survey from Akman and Surav [1996].

2.5 Concept combination – Guppy Effect

So far concept formalizations and context formalization were reviewed without mentioning concept combination. Concept combination created a problem for existing concept formalizations, *e.g.* fuzzy sets. A successful attempt to formalize concept combination based on quantum mechanics is introduced because it combines context with concepts.

Concept combination is a common process in language, such as combining two concepts like a **pet** and **bird** into the concept **pet bird**. The difficulty in formalizing such a combination is that "prototypes of constituent concepts can differ from the prototypes of their combinations in unpredictable way" [Fodor, 1994]. Osherson and Smith [1981] confirmed that observation in experiments. For example, the prototype of the conjunctive concept **stripped apple** differs from the constituent concepts **apple** and **stripped**, given in Figure 2.14. These "unpredictable" effects occurring in concept combination are categorized as the so-called *guppy effect*. The prototypical example for the guppy effect is as follows: subjects are asked to state typical exemplars for the concepts **fish** and **pet**. Typical exemplars for the concept **fish** are trout, mackerel etc., while for the concept **pet** typical exemplars are dog or cat. If subjects are asked to list exemplars for the conjunction of both concepts, the concept **pet-fish**, none of the typical exemplars for the constituents are included. A prototype for the combined concept **pet-fish** is guppy, which is not a prototype of the constituent concepts. Hampton [1997] concludes that subjects overextend prototypes for the combined concept. In summary, combined concepts can inherit the guppy effect, which makes the calculation of the prototype "unpredictable" due to overextending.

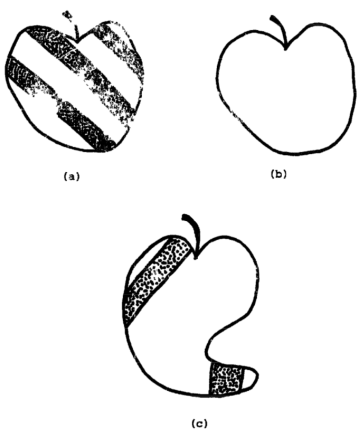


Figure 2.14: Prototypes of the concepts **apple** (a), the more typical apple **apple** (b) and the combined concept **stripped apple**, by Osherson and Smith [1981]

Concept formalizations such as fuzzy sets fall short when computing prototypes for a combined concept, because of their combination rule.

Osherson and Smith [1981] argue that the minimum combination rule for conjunction in fuzzy sets does not reproduce overextending results that are shown in cases where the guppy effect occurs. Explanation-based formalizations are similarly unable to explain concept combination [Komatsu, 1992]. Adams and Raubal [2009] introduced a metric conceptual space algebra designed to model concept combination. The model uses similarity measures including context as weights and is able to model the combination of spatial concepts such as countries. What remains open is the determination of context weights and how it can be used to model the guppy effect.

A successful attempt in formalizing the guppy effect was made with a quantum-mechanics-based approach including context. The approach is called *State COntext Property* (SCOP) and was proposed by Aerts and Gabora [2005a,b]. The main idea is to influence the concepts before combining them with context, and calculate their combination afterwards. With this approach they successfully formalized the guppy effect. First, the concept **pet** was influenced by context *the pet is a fish*, which results in a different state of the concept. Second, the concept **fish** was influenced by the context *the fish is a pet*, also resulting in a particular state of the concept **fish**. Third, the combination of both states' concepts, **pet-fish**, was calculated, and the obtained prototype was indeed the guppy.

Because of the power to explain and model concept combination, the understanding of context from SCOP is reviewed in great detail in Chapter 3. The differences from SCOP to other models such as conceptual spaces, fuzzy sets, etc. is reviewed by Gabora et al. [2008].

2.6 Conclusion

Context C is what influences the mapping between two sets X and Y (constitutive rule from Searle). This general statement applied to the referring process modeled by the semiotic triangle uses for the set X signs, terms (bank), speech, etc., and for the set Y concepts describing classes of real things (restaurants, financial bank, river bank, etc.).

What is taken as context is explained in the following chapter, where the formalization of SCOP is reviewed.

3
State Context Property Formalism – SCOP

State COntext Property (SCOP) formalism was introduced by Aerts and Gabora and is used to, for example, model concept combination [Aerts and Gabora, 2005a,b, Gabora et al., 2008]. The formalism applies quantum theory to cognitive science, in the research field of *quantum interaction*. The goal in introducing SCOP is to understand the context application in SCOP and to extract context properties.

The field of quantum interaction uses models originating in quantum theory to model experiments conducted in cognitive science [Wang et al., 2013a]. Busemeyer and Bruza [2012] and others ([Gabora, 2001, Gabora et al., 2008, Kitto, 2008]) state that if cognitive experiments present similar effects as experiments in quantum physics, then quantum interaction models are beneficial for example for effects of contextuality, similarity, compatibility, and order.

Hogarth and Einhorn [1992] observed an order effect in human belief updating, and Moore [2002] in answers to questionnaires. The same order effect occurs in electronic spin experiments where the order of the measurements influences the outcome. This similarity was used by Trueblood and Busemeyer [2010], Busemeyer et al. [2011], Busemeyer and Bruza [2012], Atmanspacher and Römer [2012], Wang et al. [2013b] to model the order effect in human belief updating, and for answers in questionnaires with quantum models.

In the field of GIScience, Hahn and Weiser [2015] observed an order effect during perspective taking when a knowledgeable source gives route directions to a stranger. In giving a route direction to a stranger, the source evaluates first the knowledge of the stranger and changes the expressions accordingly. For example, the knowledgeable source changes “Stephansdom” into “cathedral”, because the source is aware of the missing knowledge from the stranger. In the process of changing the expression, the order of perspective taking is from stranger → source and leads to a different outcome compared to the order of source → stranger.

The quantum effect of superposition of an electron is perceived similarly to effects occurring in context combination as described by Aerts and Gabora [2005b] and further discussed in this section. Schrödinger [1935] explains superposition with the thought experiment about a cat put in a box. A device in the box is filled with a radioactive substance and connected to a Geiger counter. If the radioactive substance decays, the counter discharges and the device releases poison. The decay of the radioactive substance is not predictable from outside the box. Therefore, no one knows if the cat is still alive or not. The cat is in a superposition state, only a measurement can bring evidence in which state the cat is in.

In the field of cognitive science, the use of quantum models for cognitive effects is under debate. For example, Pothos and Busemeyer [2013] received 35 highly controversial comments on their use of quantum models for the order effect. From these comments one can conclude that more experiments directly targeting quantum effects, such as order effect and superposition, have to be conducted to supply broad evidence for the legitimate use of quantum models for cognition. Nevertheless, the SCOP formalism proposes many properties for context that may be considered in the context algebra.

Quantum theory is "a set of rules allowing the computation of probabilities for the outcomes of tests which follow specified preparations" [Peres, 1934]. In contrast to classic probability theory which relies on Kolmogoroff [1933] axioms, quantum theory relies on von Neumann [1932] axioms. The difference is that Kolmogoroff axioms are based on set theory, where probabilities are assigned to events modeled as sets, whereas von Neumann axioms are based on vector spaces, where probabilities are assigned to events modeled as a subspace of a vector space. Another important aspect in both these approaches is that while classical probability theory is described as *Boolean algebra*, quantum theory is described as *partial Boolean algebra* [Hughes, 1992, page 192]. An outstanding difference is that, in contrast to classical probability theory, quantum theory excludes the *distributive axiom*. In classical probability theory, three events A, B, C follow the distributive axiom obtained from Boolean logic and given in Equation 3.1.

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{3.1}$$

Modeling these events within quantum theory, the distributive axiom does not hold [Hughes, 1992, page 206], as shown in Equation 3.2, because quantum theory relies on "quantum logic" [Hughes, 1992, chapter 7]. The absence of the distributive axiom is used to model effects such as quantum superposition [Busemeyer and Bruza, 2012].

$$A \cap (B \cup C) \neq (A \cap B) \cup (A \cap C) \tag{3.2}$$

In order to apply quantum theory to cognitive experiments, the mathematical theory has to be generalized. This was achieved by various researchers [Atmanspacher et al., 2002, Aerts and Aerts, 1997, Khrennivov, 1999], and the SCOP model [Aerts and Gabora, 2005a,b] is reviewed here in detail. This review includes an introduction to Hilbert space, because context is modeled as a projection operator which requires knowledge of Hilbert space.

3.1 Hilbert space

A *Hilbert space* is a special vector space defined over a field $\mathbb{F}$. Typical instances for the field $\mathbb{F}$ are the Real numbers $\mathbb{R}$ or the Complex numbers $\mathbb{C}$, which in a Hilbert space are called *scalar values*. Unless otherwise stated, this introduction considers only Hilbert spaces *over the field of the Real numbers* $\mathbb{R}$. The reason is twofold. First, complex numbers $\mathbb{C}$ are mostly used in physics (quantum experiments) and not often applied in quantum interaction; and second, it reduces the complexity of the explanation but still includes all the properties required for context.

SCOP uses the notation introduced by Dirac [1939] (*Dirac notation*) for vectors in a Hilbert space. A *column vector* with real scalar values r_1, r_2 is called *ket vector* and written in Dirac notation as in Equation 3.3.

$$v = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} = |v\rangle \tag{3.3}$$

A *row vector* with real scalar vectors r_3, r_4 is called a *bra vector*, as given in Equation 3.4.

$$\begin{pmatrix} r_3 & r_4 \end{pmatrix} = \langle u| \tag{3.4}$$

Operations of *vector addition* and *inner product* are defined in a Hilbert space. Vector addition is equal to vector addition in any other vector space [Lawden, 2005, page 12]. The inner product of a bra and a ket vector is called *braket* and results in a positive real number just as in ordinary vector spaces [Mac Lane and Birkhoff, 1991, page 304]. The scalar elements of the vectors are multiplied, and the results are added as shown for the braket built by $\langle u|$ and $|v\rangle$ in Equation 3.5. The Dirac notation is beneficial here, because it indicates that $\langle u|$ has to be a row vector, and $|v\rangle$ a column vector.

$$\langle u|v\rangle = r_3 r_1 + r_4 r_2 \tag{3.5}$$

As any other vector space, a Hilbert space is also established by *base vectors*. Base vectors span the space and are comparable to elementary events in classical probability theory [Busemeyer et al., 2011]. The base vectors can be chosen arbitrarily with the restriction that each base vector is *orthonormal* to all other base vectors. Orthonormality is a conjunction of two properties: *orthogonality* and *normality*. An example for a vector space with orthogonal base vectors is Euclidean space. Orthonormality can be interpreted in a statistical sense, meaning that base vectors refer to independent events. Note that base vectors are not equipped with scalar values.

3.2 Concept represented as state vector

The special vector called *state vector* is used to model the actual state of the system and acts as "working memory" [Baddeley, 1992]. A state vector is comparable to the probability function from classical probability theory [Busemeyer et al., 2011]. In other words "… [a] state [vector] is characterized by the probabilities of the various outcomes of every conceivable test" [Peres, 1934, page 24] where every conceivable test is modeled with basic vectors.

The state vector is normalized and can include scalar values for multiple base vectors. Normality means that the vector has a length of 1. This property is similar to classical probability theory where the sum of all conceivable tests has to sum up to 1.

The state vector is used by Aerts and Gabora [2005a] to model a concept and its evaluation in the process of refinement. This implies for the concept that it can have multiple *states of a concept*. Gabora [2001] illustrates the states of a concept as shown in Figure 3.1. Starting at time t_0, concept **p** can refine into the possible states **p1** (t_1), **p2** (t_1), **p3** (t_1), and **p4** (t_1).

The refinement from one state to another state is achieved by applying context, modeled as an operator. For example, in Figure 3.1 the context e_3 is applied to concept **p** (t_0), which refines it into state **p3**(t_1). The further application of context e_7 refines the concept into state **p7**(t_2), as shown in Figure 3.1.

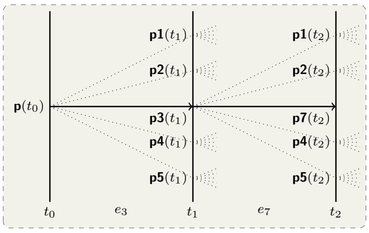


Figure 3.1: States of a concept by Gabora [2001]

3.3 *Context represented as a projection operator from state to state*

In quantum theory, an operator $\mathbf{A}$ is applied to a vector $|v\rangle$ which yields another vector $|v'\rangle$ given in Equation 3.6. Operators respect the *associative* property given in Equation 3.7. Examples of operators are rotation operators and Pauli [1927] matrices for electron spin.

$$\mathbf{A}|v\rangle = |v'\rangle \tag{3.6}$$

$$\langle u|(\mathbf{A}|v\rangle) = (\langle u|\mathbf{A})|v\rangle = \langle u|\mathbf{A}|v\rangle \tag{3.7}$$

Projection operators, or projectors for short, are a special type of operator that map from the space represented by a vector to a subspace of the vector. An example for a projection operator is the identity projector $\mathbf{I}$ which maps to the whole space. An example for a three-dimensional identity projector is given in Equation 3.8. For each base vector a projection operator exists, which is built by the outer product of the considered base vectors, having only one entry in the diagonal. Projection operators for more than one base can also be generated by the outer product. Projection operators can be interpreted as yes/no questions [Peres, 1934, page 66] for those base vectors that are considered by the projector. For example, the identity projector (Equation 3.8) includes yes questions for all base vectors.

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

Projection operators are *idempotent* [Hughes, 1992, 1.26], as given in Equation 3.9. Idempotency states that a multiplication of a projector with itself results in the same projector.

$$\mathbf{AA} = \mathbf{A} = \mathbf{A}^2 \tag{3.9}$$

Aerts and Gabora [2005a] used projection operators to model context. For example, the context *the pet is a fish* represented as $\mathbf{P}_{\textbf{fish}}$

applied to the concept **pet** (modeled by a state vector) $p(t_0)$ refines this state into the state “pet is a fish” $p(t_1)$, which is a subset of the former state. The application is illustrated in Figure 3.2, showing base vectors. In terms of identifying algebraic properties for context, the properties of projection operators also apply to context. In fact, context also satisfies the property of idempotency.

$$\begin{array}{c}\begin{array}{cccccccc}\rotatebox{90}{the pet is a dog} & \rotatebox{90}{the pet is a trout} & \rotatebox{90}{the pet is a trout} & \rotatebox{90}{\dots} & \rotatebox{90}{the pet is a guppy} & \rotatebox{90}{the pet is a cat} & \rotatebox{90}{\dots} & \rotatebox{90}{the pet is a hamster}\end{array}\\ \begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&\ddots&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&\ddots&0\\0&0&0&0&0&0&0&1\end{pmatrix}\\ \text{state, } \textbf{pet}, \ p(t_0)\end{array} \cdot \begin{array}{c}\overbrace{\hphantom{0\quad0\quad0\quad0}}^{\textit{is a fish}}\\ \left(\begin{array}{c:cccc:ccc}0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&\ddots&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&\ddots&0\\0&0&0&0&0&0&0&0\end{array}\right)\\ \text{projector, } \textit{is a fish}, \ \mathbf{P}_{\mathbf{fish}}\end{array} = \begin{array}{c}\begin{pmatrix}0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&\ddots&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&\ddots&0\\0&0&0&0&0&0&0&0\end{pmatrix}\\ \text{state, } \textbf{pet is a fish}, \ p(t_0)\end{array}$$

Figure 3.2: state **pet** influenced by context *is a fish* resulting in the state “pet is a fish”

The set of all projection operators of a Hilbert space forms a *lattice* structure [Basieva and Khrennikov, 2015]. The set of projection operators corresponds one-to-one to sub-spaces [Hughes, 1992][page 47]. Aerts and Gabora [2005a] expressed this correspondence between projection operators and sub-spaces through mapping λ. The sub-spaces are partially ordered [Beltrametti and Cassinelli, 1981][page 105] to form the lattice. The lattice is bound with the zero operator $\mathbf{0}$ given in Equation 3.10, and the identity operator $\mathbf{I}$ in Equation 3.8.

The lattice structure of projectors with its partial order relation applies to context as well. Aerts and Gabora [2005a] label the partial order relation “is stronger than or equal to”, denoted by $\leq$. They assign the zero operator the name *zero context*, and the identity operator the name *unit context*. The *partial order relation* respects the properties of *reflexivity*, *transitivity*, and *symmetry.*

$$\mathbf{0} = \begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix} \tag{3.10}$$

Two contexts are combined by conjunction or disjunction resulting in another context. The combination of *two* contexts with “and” (conjunction) results in an infimum context, denoted $\bigwedge c$. This infimum context is more concrete than the constituents. The calculation of the infimum context for contexts c_i included in a context set C is the intersection of two contexts given in Equation 3.11, [Aerts and Gabora, 2005a].

$$\bigcap_{c_i \in C} \lambda(c_i) = \lambda(\bigwedge_{c_i \in C} c_i) \tag{3.11}$$

The combination of two contexts with "or" (disjunction) results in a supremum context, denoted $\bigvee c$. This context is more abstract than the constituents.

Aerts and Gabora [2005a] calculated the supremum context for contexts c_i in a context set C with the *closure* operation given in Equation 3.12. The closure operation in Equation 3.12 does not produce an equal for the disjunction of two contexts $\lambda(c_1) \cup \lambda(c_2) \neq \lambda(c_1 \vee c_2)$. The reason for selecting the closure operation is based on a thought experiment which lacks empirical evidence (surprisingly, they conducted a questionnaire but didn't include questions tackling this major argument for use of the closure operation). The thought experiment creates a hypothetical situation where a person cannot decide which of two alternatives is valid. In their perspective, the situation resembles the behavior of an electron in superposition modeled with the closure operation in quantum mechanics. Busemeyer and Bruza [2012][page 2] explain superposition for cognition of being in an indefinite state, *e. g.* "conflicted, ambiguous, confused, uncertain" where either of two or more possibilities are achievable. Note: this does not mean that more possibilities can be realized at the same time. However, this thought experiment led Aerts and Gabora [2005a] to conclude that the context structure is quantum-like, also calling the supremum context superposition context.

$$\bigcup_{C_i \in C} \lambda(c_i) \subset \lambda(\bigvee_{c_i \in C} c_i) \tag{3.12}$$

Each context c has an orthocomplement context $c^{\perp}$. The orthocomplement function has the properties stated in Equation 3.13. Aerts and Gabora [2005a] argue that it is not a complement because of the "existence of superposition states". Here superposition is referred to as a principle that tells which set of states is admissible from the system [Hughes, 1992]. Aerts and Gabora [2005a] argue for the existence of the orthocomplement via a thought experiment constructed by the disjunction of a context c with its orthocomplemented $c^{\perp}$ built as: $\lambda(c) \cup \lambda(c^{\perp}) \subset \lambda(c \vee c^{\perp})$. This equation relies on superposition, and within the mapping λ it differs from a complement version: $\lambda(c) \cup \lambda(c^{\perp}) \neq \lambda(c \vee c^{\perp})$ because of possible *admissible states* created by the superposition. Besides the thought experiments included in the argument of Aerts and Gabora [2005a], there is no empirical evidence (same as for superposition context) for the preference of the orthocomplement over the complement.

$$\begin{aligned} &(c^{\perp})^{\perp} = c \\ &c_i \leq c_j \implies c_j^{\perp} \leq c_i^{\perp} \\ &c \wedge c^{\perp} = 0, c \vee c^{\perp} = 1 \end{aligned} \tag{3.13}$$

Aerts and Gabora [2005a] conclude that context forms a complete orthocomplemented lattice *e. g.* in Figure 3.3.

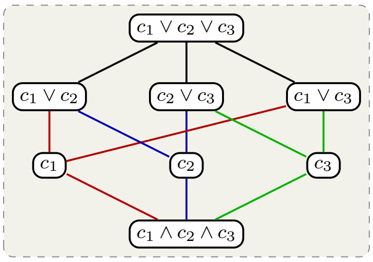


Figure 3.3: Lattice structure for context

3.4 Basic context

The context lattice is built by elements chosen to cover all possible contexts. Aerts and Gabora [2005a] presented in an example how these elements could be selected to model the concept **pet**.

The elements were collected from a *questionnaire*. In the questions, subjects rated the likelihood of an exemplar of the **pet** concept influenced by context *e. g.* *The pet is chewing a bone*. The most likely exemplars, *e. g.* dog, cat or hamster responded by the participants are used as basic contexts, *e. g.* *The pet is a hamster*, to build the Hilbert space indicated in Figure 3.2. For each mention of an exemplar (multiple mentions possible), another basic context is included which results in a Hilbert space of dimension 1400 for the concept **pet** [Aerts and Gabora, 2005b]. Basic contexts are located at the bottom of a context lattice, *e. g.* contexts c_1, c_2, c_3 in Figure 3.3.

Basic contexts are used to build all other contexts in the context lattice by combination. Other contexts, *e. g.* *The dog is chewing a bone*, are built by basic contexts c_1 *The pet is a dog* and c_2 *The pet is chewing a bone*, illustrated as $c_1 \vee c_2$ in Figure 3.3.

The number of basic contexts included in a context is used to calculate a prototype. The calculation of the prototype is done by counting the number of basic contexts for an exemplar divided by the whole number of basic contexts. For example, the basic context *The pet is a dog* is counted seven times and the basic contexts *The pet is a hamster* is counted twice. This results for dog in a probability of $^7/_9$, and for hamster of $^2/_9$. The maximum number is picked as the prototype, in this example dog as $^7/_9 > ^2/_9$.

3.5 Conclusion

SCOP takes context into account for the relation between concepts and exemplars. Context acts as a glue to connect exemplars and concepts. The concept influenced by a particular context relates to those exemplars that are valid in the context. Turning this model into the constitutive rule from Searle [1995], the concept **X** (represented as state vector) counts as the prototype Y (basic context) in context *C* (represented as projection operator). For example, the concept **pet** acts as dog in the context *the pet is chewing a bone*.

The remarkable idea of SCOP is that concepts probably take a state influenced by context, and change over time as context is refined. In each state of the concept, other exemplars are taken into account to calculate the prototype. By using this approach for each state, different prototypes may arise depending on the context. The state of a concept serves as input for the next calculation where it is influenced by another context, and results in another state.

The SCOP formalism includes algebraic properties for context and the partial order relation. Contexts are associative and idempotent, the partial order relation is reflexive, transitive, and symmetric. Similar properties were already stated by Shoham [1991] reviewed in Chapter 2.4, and are confirmed by SCOP [Aerts and Gabora, 2005a,b].

The influencing context is modeled by a lattice structure with two combination functions where the "or" combination is under discussion. Contexts are "and" and "or" combined, as in normal language where Aerts and Gabora argue that the "or" combination of contexts shows effects similar to the superposition of electrons. Their argument is not based on empirical evidence, however, and for that reason it is not considered in the remaining work.

Basic contexts are gathered from answers of questionnaires and are combined to populate the Hilbert space. The established Hilbert space has finite dimensions equipped with real scalar values, which in turn makes use of a Hilbert space superfluous as demonstrated by the context algebra in the following Section 4.

In the geospatial domain, several features included in SCOP and reviewed in this chapter are topics of high interest. For example, di Donato [2010] argues that existing techniques, *e.g.* ontologies, are not able to model *geospatial semantics*, where SCOP could be a promising alternative. Di Donato [2010] highlights the idea of states for a concept and concludes that an in-depth analysis of SCOP is necessary to identify applicability in the geospatial domain. Such an analysis is shown in this chapter, extracting properties usable for the geospatial domain, *i.e.* context properties for the representation of near. The idea that concepts inherit states was also proposed by Raubal [2008]. He presented an algebraic model inspired by time geography and conceptual spaces and justified it using the spatial concept "landmark". These two examples document the interest by the geospatial domain to integrate concepts and context into one formalization, as is demonstrated in this work.

4
Context Algebra mapped to concepts

In this chapter, context algebra is introduced, and a mapping from a context to a concept formalization is established to calculate context-dependent prototypes. To achieve this goal the chapter is structured into two sections. First, an algebra for contexts is defined in Section 4.1. Second, from that context algebra a mapping is defined to a concept formalization in Section 4.2. Within this mapping, different prototypes can be calculated dependent on context influence. All properties from the algebra and the mapping are discussed to create a self-contained description of the context algebra, including some of the properties already introduced in Chapters 2 and 3.

4.1 Context Algebra

Context algebra provides methods to calculate with context in a general manner, allowing it to be applied in many applications. A context algebra is an instance of an algebraic system. According to Gill [1976, page 94] an algebraic system includes:

- “a set V known as domain of the algebra”
- “a set of relations on V”
- “a set of operations on V”
- “a set of postulates . . . for the operations”
- “a set of theorems that are statements concerning V and can be proved from the postulates by means of accepted rules of logic.”
- “a set of definitions about symbolism and terminology”
- “a set of algorithms”.

The context algebra is introduced in this sequence, with all elements included.

The domain of the context algebra is a context set C. Possible elements included in this set are listed in Table 4.1. These contexts are used within this chapter, where a particular context is referred to as c_i or by its label. The contexts in Table 4.1 establish a relation between the concept **restaurant** and specific exemplars, *e. g.* Plachutta, as given in the introduction in Section 1.1.

context	label
c_1	*in Vienna*
c_2	*in Innere Stadt*
c_3	*serving pizza*
c_4	*serving regional cuisine*
c_5	*in Vienna serving pizza*
c_6	*not in Vienna*
c_7	*in Vienna serving pizza at 8 o'clock*
c_8	*at 8 o'clock*
c_9	*serving pizza or serving regional cuisine*
c_{10}	*first district*

Table 4.1: Example contexts included in context set C also used in the introduction example; context maps from the concept **restaurant** to exemplars e.g Plachutta

Partial order relation

Two contexts $c_i, c_j \in C$ are partially ordered by the binary order relation *is more selective than or equally selective as*. This relation is denoted $\leq$ [1], and its mathematical definition is given in Equation 4.1.

$$c_i \leq c_j \quad \forall c_i, c_j \in C \tag{4.1}$$

The order relation is read as context c_i is more selective than or equally selective as context c_j. An example for such an ordering is given by the contexts c_1 and c_2. They can be ordered as $c_2 \leq c_1$, reading context c_2=*in Innere Stadt* is more selective than or equally selective as context c_1=*in Vienna* for the statement "Let's meet at a restaurant".

The partial order relation $\leq$ is asymmetric. This property states that the converse of $\leq$ is the relation $\geq$. The converse partial order relation is given in Equation 4.2 and reads *is less selective than or equally selective as* the other context.

$$c_j \geq c_i \quad \forall c_i, c_j \in C \tag{4.2}$$

By defining both partial order relations, the equality of two contexts can also be calculated, which is done as stated in Equation 4.3. If two contexts satisfy both relations $\leq, \geq$ then the two contexts are *equally selective.*

$$c_i \leq c_j \text{ and } c_i \geq c_j \implies c_i = c_j \tag{4.3}$$

The partial order relations $\leq, \geq$ are reflexive. Reflexivity states that the partial order relation can be applied to every context $c_i \in C$ and returns meaningful results. The relation is also meaningful if a context c_i is related to itself, as given in Equation 4.4.

$$\forall c_i \in C \qquad c_i \leq c_i \tag{4.4}$$

The partial order relations $\leq, \geq$ are transitive, as stated in Equation 4.5.

$$c_i \leq c_j \text{ and } c_j \leq c_k \implies c_i \leq c_k \tag{4.5}$$

For example, consider the contexts c_1, c_5, and c_7. Contextc_5=*in Vienna serving pizza* is more selective than or equally selective as context c_1=*in*

[1] Mathematics read the symbol $\leq$ as "less than or equal than" and it is used here because the number of references included in the left context are "less than or equal to" compared to the number of references in the right context. For the reading, this implies that the context on the left side is "more selective or equally selective" as the context on the right side.

Vienna. Furthermore, context c_7=*in Vienna serving pizza at 8 o'clock* is more selective than or equally selective as context c_5=*in Vienna serving pizza*. Now, the transitivity property states that c_7 is also more selective than or equally selective as context c_1. Within transitivity, the three contexts are ordered as: $c_7 \leq c_5 \leq c_1$.

Greater selective contexts and *least selective contexts* for three contexts can be determined within the partial order relation.

Greater selective contexts are more selective or equally selective with respect to two other contexts. A greater selective context is denoted as g and is also included in the context set C. In order to be a greater selective context it has to satisfy the law stated in Equation 4.6.

$$g \leq c_i \tag{4.6}$$

An example greater selective context can be found for the two contexts c_1=*in Vienna* and context c_3=*serving pizza*: the greater selective context is context $g = c_5 =$ *in Vienna serving pizza*.

Least selective contexts are less selective or equally selective with respect to two other contexts. A least selective context is denoted as l and is also part of the context set C. A least selective context is found if it satisfies the law stated in Equation 4.7.

$$l \geq c_i \tag{4.7}$$

An example for a least selective context from the contexts c_3 and c_4 is already included in the context set C. The least selective context for these is context $c_9 = l$=*serving pizza or serving regional cuisine*, which is less selective than or equally selective as each of the other contexts c_3=*serving pizza* and c_4=*serving regional cuisine*.

One universal *least context* and one universal *greatest context* in a partially ordered context set C exist; and if they exist, they are a least selective context or greatest selective context for all contexts included in the context set. The universal *least context* denoted $\perp$ satisfies Equation 4.8. In the perspective of the constituting rule of Searle, the least context represents a context where no relation between X and Y can be determined, *i. e.* no relation to reality can be determined.

$$\forall c_i \in C \qquad \perp \leq c_i \tag{4.8}$$

The universal *greatest context*, denoted $\top$ satisfies Equation 4.9. In Searle's perspective the greatest context includes all relations between X and Y.

$$\forall c_i \in C \qquad c_i \leq \top \tag{4.9}$$

The least and greatest contexts are unique for a context set C. For every context $\perp, \top, c_i \in C$ it follows that $\perp \leq c_i \leq \top$. The $\perp$ context is more selective than or equally selective as all other contexts, and the $\top$ context is less selective than or equally selective as all other contexts in the context set. $\top$ and $\perp$ context can only be determined with respect to a specific model, as noted by McCarthy [1996] reviewed in Section 2.4.

In summary, a partially ordered set of contexts is represented by the mathematical structure *poset*. The poset can be drawn as an ordering diagram. For example, considering the contexts $\bot, c_1, c_3, c_4, c_8, \top$, the ordering diagram included in Figure 4.2 is created.

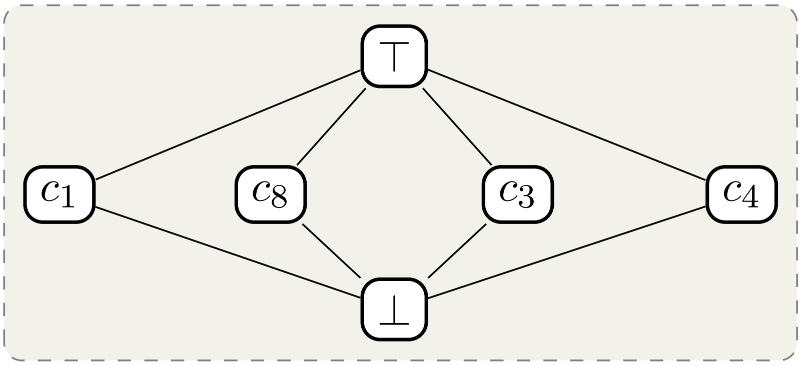


Figure 4.1: Ordering diagram for contexts $c_1, c_8, c_3, c_4, \bot, \top$ depicted from in Table 4.1

Context combination operations

Contexts included in a context set C can be combined with two binary operations, conjunction and disjunction, resulting in another context.

The *conjunction* operation of two contexts, denoted $\wedge$, is given in Equation 4.10. The conjunction of two contexts results in a greater selective context as both constituents $c_i, c_j \in C$.

$$c_i \wedge c_j = g \quad g \leq c_i, c_j \in C \tag{4.10}$$

A combination for two contexts results in another context. For example, the context c_1 = *in Vienna* conjunct with the context c_2= *in Innere Stadt* results in the context c_{10}= *first district*, which is determined by a new name. Another example for a combination can be created from context c_1= *in Vienna* and context c_3= *serving pizza*, which results in context *in Vienna and serving pizza*. In most cases, the term "and" can be omitted, which is given in context c_5 = *in Vienna serving pizza*.

The *disjunction* of two contexts, denoted $\vee$, is stated in Equation 4.11. The disjunction of two contexts results in a least selective context.

$$c_i \vee c_j = l \quad l \leq c_i, c_j \in C \tag{4.11}$$

For example, the disjunction of the context c_3= *serving pizza* and c_4= *serving regional cuisine* results in context c_9= *serving pizza or serving regional cuisine*.

Conjunction and disjunction of two contexts satisfy the *idempotent* property. This property is given in Equation 4.12 and shows that the combination of one context with itself results in itself.

$$c \wedge c = c, \qquad c \vee c = c \tag{4.12}$$

For example, context c_1= *in Vienna* in conjunction with itself results in *in Vienna and in Vienna* which equals *in Vienna*. Also, the disjunction of context c_1 with itself resulting in *in Vienna or in Vienna* is the context itself.

Both operations are *commutative*, as given in Equation 4.13. This property states that the order of the context does not have any influence on the outcome.

$$c_i \wedge c_j = c_j \wedge c_i, \qquad c_i \vee c_j = c_j \vee c_i \tag{4.13}$$

For example, if context c_1= *in Vienna* is conjunct with context c_3= *serving pizza*, there is no difference between $c_1 \wedge c_3$= *in Vienna serving pizza* or $c_3 \wedge c_1$= *serving pizza in Vienna*.

Both operations are *associative*, as given in Equation 4.14 which states that parentheses do not affect the operation.

$$c_i \wedge (c_j \wedge c_k) = (c_i \wedge c_j) \wedge c_k, \qquad c_i \vee (c_j \vee c_k) = (c_i \vee c_j) \vee c_k \tag{4.14}$$

Both operations satisfy the *consistency* law stated in Equation 4.15.

$$c_i \leq c_j \qquad c_i \wedge c_j = c_i, \qquad c_i \vee c_j = c_j \tag{4.15}$$

For example, context c_5= *in Vienna serving pizza* is more selective than or equally selective as c_1= *in Vienna*, resulting in $c_5 \leq c_1$. If both are combined by the conjunction operation, the result is the more selective context c_5. If both contexts are combined by the disjunction operation, the result is the less selective context c_1.

Both operations are *isotone*, as stated in Equation 4.16.

$$c_j \leq c_k \qquad c_i \wedge c_j \leq c_i \wedge c_k \qquad c_i \vee c_j \leq c_i \vee c_k \tag{4.16}$$

If the context poset includes a universal least context $\perp$ and a universal greatest context, $\top$ laws given in Equation 4.17 and 4.18 for the conjunction and disjunction operation have to be satisfied. The laws for the least context are stated in Equation 4.17, and those for the greatest context are stated in Equation 4.18

$$c_i \in C \qquad c_i \wedge \perp = \perp \qquad \perp \vee c_i = c_i \tag{4.17}$$

$$c_i \in C \qquad c_i \wedge \top = c_i \qquad c_i \vee \top = \top \tag{4.18}$$

Considering the contexts from Table 4.1, the $\perp$ and $\top$ context can be determined. The $\perp$ context has to be more selective compared to all other contexts. As no context is more selective than all others, the $\perp$ context has to be added to the poset C. There exists one context which is more selective than all others, called the empty context= $\{ \}$. It is created by the conjunction of all contexts in the poset C, which represents $\perp$ in this example. The $\top$ context is the least selective context and is created by the disjunction of all contexts included in the poset C.

The contexts resulting from the disjunction and conjunction operations are included in the ordering diagram, which also shows greater and least context. An example ordering diagram for the generic contexts c_i, c_j, c_k, c_l, $\perp$, $\top$ is shown in Figure 4.2. The ordering diagram includes greater contexts (conjunction) and least (disjunction) contexts for each context. They can be determined by following the path of both constituting contexts ascending or descending, and when the paths meet, the least or greater context is found. For example, the greater context of c_i, c_k and c_j, c_k, c_l is determined by following the descending path indicated by dotted lines ending up at context c_k. The least context is represented by the vertex that is reachable from both constituent contexts via an ascending path. For the contexts c_i, c_k and c_j, c_k, c_l, the least context is c_i, c_j, c_k, c_l where the path is indicated by the dashed line.

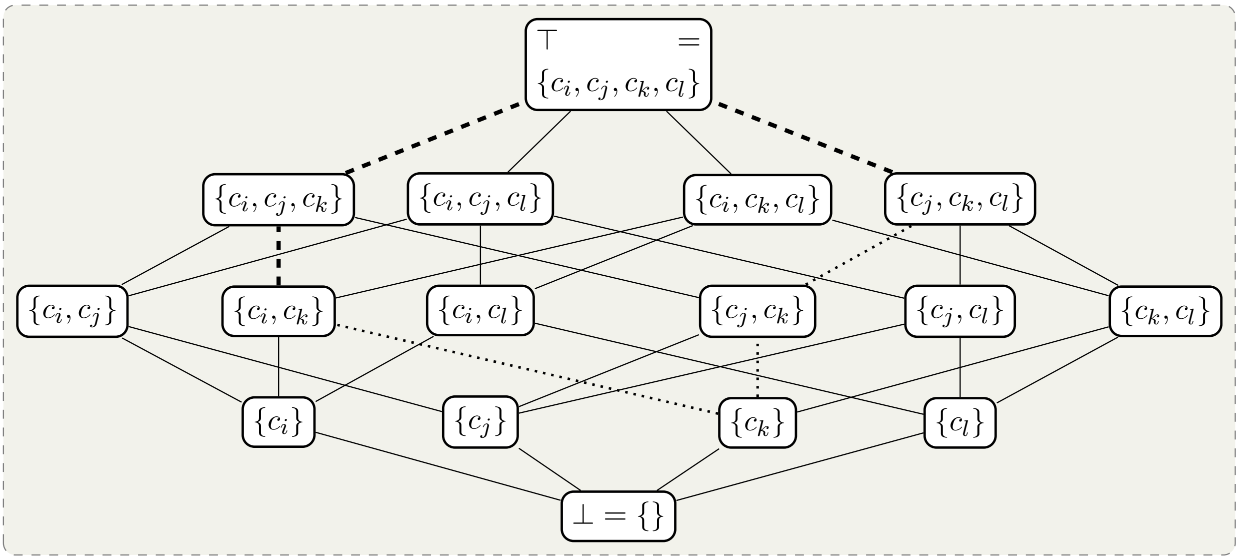


Figure 4.2: Greater selective context and least selective context in an ordering diagram for context set C including the contexts $c_i, c_j, c_k, c_l, \bot, \top$

Context Lattice

Within partial order relation and combination operations, a *lattice structure* for contexts can be derived. To derive a context lattice, the partial order relation and the operations are realized with set functions. The partial order relation is realized as it is. The $\wedge$ function is realized as set intersection, given in Equation 4.19, the $\vee$ function is realized by the set union given in Equation 4.20, and summarized in Table 4.2.

combination operation	set function
conjunction $\wedge$	intersection $\cap$
disjunction $\vee$	union $\cup$

Table 4.2: Context combination functions realized as set functions

$$\text{conjunction:} \quad c_1 \wedge c_2 = c_1 \cap c_2 \tag{4.19}$$

$$\text{disjunction:} \quad c_1 \vee c_2 = c_1 \cup c_2 \tag{4.20}$$

A context c_i has a *complement* context denoted, as given in Equation 4.21.

$$c_i \in C \qquad \overline{c_i} \tag{4.21}$$

A *context lattice* is complemented if for every $c \in C$ there is a complement element $\bar{c}$. For example, the complement of context c_1= *in Vienna* is $\overline{c_1} = c_6$= *not in Vienna*.

The complement of the complement of a context results in the context itself, as shown in Equation 4.22

$$\bar{\bar{c}} = c \tag{4.22}$$

For two contexts $c_i, c_j \in C$ and the referring complement contexts $\overline{c_i}, \overline{c_j}$ the partial order relation given in Equation 4.23 relates them.

$$c_i \leq c_j \implies \overline{c_j} \leq \overline{c_i} \tag{4.23}$$

Furthermore, the complement laws stated in Equation 4.24 are satisfied by the combination operations.

$$c_i \wedge \overline{c_i} = \bot \qquad c_i \vee \overline{c_i} = \top \tag{4.24}$$

The complement function is realized as a negation operation, assuming a bound context lattice C.

4.2 Mapping from Context to Concepts

In this section the mapping from the context algebra to a concept representation is introduced. This mapping enables the representation for the introduction example "Let's meet at a restaurant". Within this mapping, context (*e.g.* *serving pizza*) is used to select references from all the possible references of a word (*e.g.* RESTAURANT) to exemplars in reality (*e.g.* Plachutta). To introduce the mapping, first the idea is sketched and then the formalization is presented. At the end of this section an example will demonstrate how the mapping is used to calculate prototypes depending on context.

Contextualized Concepts

The idea of mapping context to a concept representation is based on the context-enriched semiotic triangle illustrated in Figure 4.3. In general, the semiotic triangle highlights the issue of many-to-many relations between a word, a concept, and the exemplar in reality. Here, context is introduced to select only those references that are intended to be referred to. The enriched semiotic triangle was initially introduced by Hahn et al. [2016] for spatial concepts, but it is used here for concepts in general.

The semiotic triangle describes the abstraction and externalization process for a concept labeled **Concept** in Figure 4.3. In Figure 4.3 two observations are made in two different contexts for the **Concept**. The first observation includes exemplar exemplar 1 in *context 1*, while exemplar 2 is observed in *context 2*. By making the context explicit in the abstraction process, the **Concept** can include multiple sub-concepts depending on context; they are called *contextualized concepts*. In Figure 4.3 two contextualized concepts, denoted by **Concept@***context 1*[2] and **Concept@***context 2*, are established from the two observations.

By introducing contextualized concepts, references used in the interpretation process can be distinguished. For example, by using context

[2] this specific notation for a contextualized concept will be used from now on

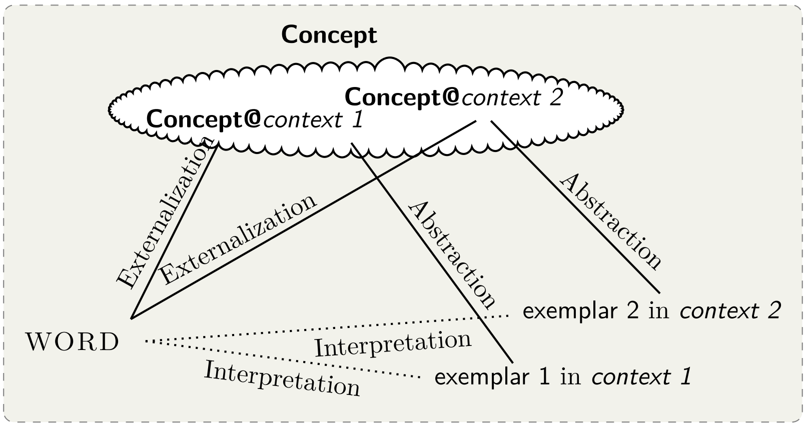


Figure 4.3: The semiotic triangle from Ogden and Richards [1946] used for geographic information science by Kuhn [2005], enriched with context Hahn et al. [2016].

context 1 only the reference to exemplar 1 is selected. The establishment of contextualized concepts represents the idea that concepts inherit multiple states, as was concluded in Section 3.5.

Formalizing the mapping of context to concepts

The idea of contextualized concepts established by a context-enriched semiotic triangle is formalized here. In this idea, context acts on the relation between concepts and exemplars. To respect this idea, the context lattice is equipped with a mapping to a mathematical structure representing the concept. In order to establish the mapping, the mathematical representation for the concept is introduced using sets.

In order to be as general as possible, concepts are represented as a set of exemplars. Sets can include all types of elements and are equipped with an equality relation. The set representation of a concept is the co-domain of the mapping function.

The mapping function has to select sets (co-domain) for a particular context (domain). The co-domain is given as the set E including exemplars $e \in E$. The domain is given by the context lattice including contexts $c \in C$. The mapping function m operates on the structures C, E and is defined as $m : C \mapsto E$. It selects sets of exemplars E_a as defined in Equation 4.25 and reads *influenced by context.*

$$m(c) = \begin{cases} \emptyset & \text{if } c = \bot \\ E_a \quad |a \in C, a \leq c & \text{if } c \neq \bot \end{cases} \tag{4.25}$$

The mapping function distinguishes the cases for $\bot$ context and all others contexts. If the context equals to the $\bot$ most selective context, no particular set E is selected, which results in the empty set denoted $\emptyset$. For all other contexts, the result is built by a set of exemplars E_a built from atomic context representation.

Atomic context representation (or short atomic contexts) represented as variable a in Equation 4.25 are the building blocks for the context representation. Atomic contexts are contexts which are less selective than the most selective context $\bot$. One can think of as a minimal non-zero element. Atomic contexts are chosen from an agent and depend on the contexts influencing a concept. Atomic contexts build the mapping from the lattice structure to the concept structure including exemplars. The concept is partitioned according to the atomic context representation, resulting in contextualized concepts denoted E_a.

The benefit of using atomic context representations is that every non-atomic context can be built by a disjunction of atomic contexts, as given in Equation 4.26.

$$c_i, a_1, a_2, \ldots, a_h \in C \qquad c_i = a_1 \vee a_2 \vee \ldots \vee a_h \tag{4.26}$$

Each atomic context representation a of the context lattice has to satisfy the laws in Equations 4.28 and 4.27.

$$c_i \wedge a = a \qquad \mathit{or} \qquad c_i \wedge a = \bot \tag{4.27}$$

Furthermore, a is an atom if $a \neq \bot$ and if, for every $c_i \in C$

$$a \leq c_i \qquad \mathit{or} \qquad c_i \wedge a = 0 \tag{4.28}$$

Atomic context representations of a context lattice are at the bottom of the ordering diagram, as indicated by Figure 4.4.

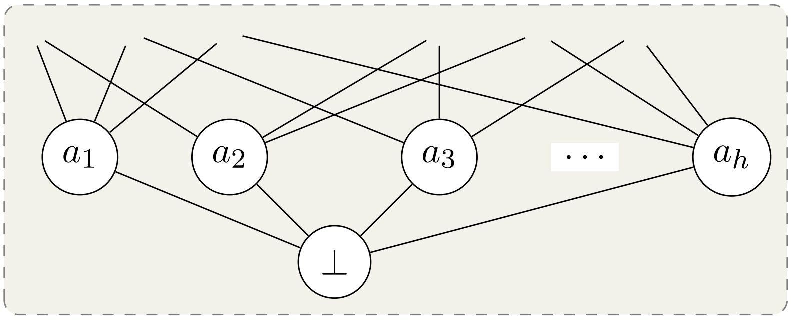


Figure 4.4: Atomic representation included in a context lattice (general version by Gill [1976, page 166])

The atomic context representations a_i are mapped to contextualized concepts E_a. The entities included in sets E_a are the exemplars that have a relation with this particular context. Examples are given in the following explanation.

Explanation of the mapping by formalizing the introductory example

In the introduction, the example sentence "Let's meet at a restaurant" was illustrated by maps (Figure 1.2, 1.4), which are now used to present the mapping from the context lattice to the concept representation. As context, the two contexts c_1= *in Vienna* and c_3= *serving pizza* are considered here. As concept, the concept **restaurant** is used to represent the sentence.

To establish the context lattice, atomic context representations are determined. The contexts c_1 and c_3 are not atomic contexts, because they can be further combined to build more selective contexts. The combination of both contexts results into the three *atomic context* *representations*: a_1= *in Vienna and serving pizza*, a_2= *in Vienna and not serving pizza*, and a_3= *not in Vienna and serving pizza*. They are used to build c_1 and c_3 by disjunction of the atomic context representations. All other contexts resulting from the disjunction of these three atomic contexts $a_2 \vee a_3$ and $\top = a_1 \vee a_2 \vee a_3$ and the $\perp$ context result in the context lattice for this example, illustrated in Figure 4.5.

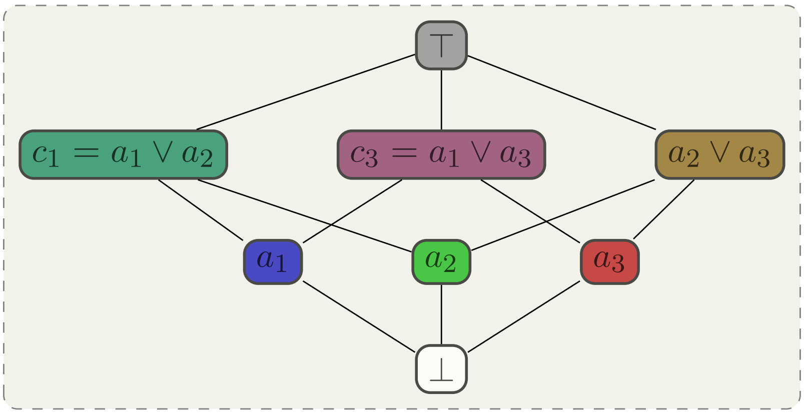


Figure 4.5: Context lattice for c_1= *in Vienna* and c_3= *serving pizza* built by atomic context representations

Contextualized concepts are modeled as sets E_i built by exemplars. The exemplars are observed in reality and for example stored in a database for a hypothetical web service. It shows exemplars/ locations of concept **restaurant** on a map similar to Figures 1.2 and 1.4. The web service includes four restaurant brands: Plachutta labeled P, Pizza Hut labeled H, Vapiano labeled V, and Cantinetta labeled C. The contextualized concepts (**restaurant@***in Vienna serving pizza* and **restaurant@***serving pizza*) including the observations for the exemplars (different locations for a brand) are illustrated as Venn diagrams in Figure 4.6. The contextualized concepts E_{c_1} and E_{c_3} are mapped to contexts that are a combination of atomic context representations. In the web service, exemplar Plachutta is observed once (one location included in the database), Cantinetta is observed three times (three locations included in the database), and Vapiano is observed five times (five locations included in the database) in the combination of context c_1 and c_3, illustrated by the overlapping area indicated by E_{a_1} in blue color. In this case, the combination of the contexts results in the atomic context representation a_1; but the fact that a combination of two contexts results in atomic context representations is not usually true, as becomes clear if one imagines, for example, a bigger context lattice. The Venn diagram also shows how the contextualized concepts respect the combination of contexts starting with the atomic context representation. For example, the left circle indicates the contextualized concept that maps to the atomic context representation a_2. Disjunct, the contextualized concepts established from the atomic context representation a_2 and a_1 generate the contextualized concept E_{c_1} covering the green and blue areas.

All possible mappings from the context lattice shown in Figure 4.5 to all possible contextualized concepts E_i are included in Table 4.3.

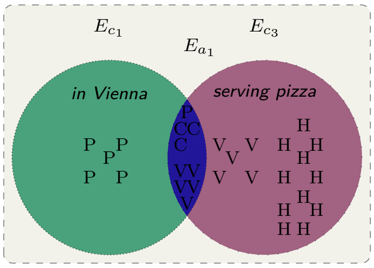
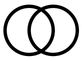
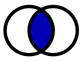
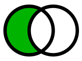
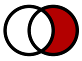
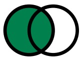
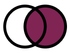
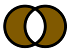
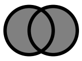


Figure 4.6: Contextualized concepts represented with Venn diagrams including exemplars

Together, Table 4.3 and Figure 4.5 illustrate the function of mapping $m(c)$. Every element in the context lattice is connected by the mapping $m(c)$ with a contextualized concept E_i sharing the same color. For example, the $\perp$ context does not consider any of the atomic context representation and returns the empty set, as given in Equation 4.25. The $\top$ context considers each atomic context representation which is less selective than all other contexts. The context c_1= *in Vienna* is created by the union of the atomic context representation of a_1= *in Vienna* $\wedge$ *serving pizza* and the atomic context representation a_2= *in Vienna* $\wedge\,\overline{\textit{serving pizza}}$ represented by the dark green area.

Table 4.3: All possible contexts generated from atomic context representations for the restaurant example

context	disjunction of	E_i
$\perp$		
in Vienna $\wedge$ *serving pizza*	a_1	
in Vienna $\wedge\,\overline{\textit{serving pizza}}$	a_2	
$\overline{\textit{in Vienna}}\wedge$ *serving pizza*	a_3	
in Vienna	$a_1 \vee a_2 = c_1$	
serving pizza	$a_1 \vee a_3 = c_3$	
in Vienna $\vee$ *serving pizza*	$a_2 \vee a_3$	
$\top$	$a_1 \vee a_2 \vee a_3$	

Hahn et al. [2016] applied context algebra to the concept **near**, and observed that some combination of contexts can yield *non-sensical contexts*. They established a context lattice including the contexts *walking* and *driving* for the concept **near**. By combining these two contexts, the combination context *walking* $\wedge$ *driving* is built, which results in a non-sensical context because it cannot be realized. Hahn et al. [2016] concluded that non-sensical contexts are not an issue for the model because the mapping from the non-sensical contexts maps to an empty contextualized concept or the context is considered as $\perp$.

Prototype calculation of contextualized concepts

Every concept selects a prototype, for which it is assumed that it can be calculated from observed exemplars. Methods to calculate a prototype are reviewed in Sections 2.2 and 3. The calculation of the prototype is not the focus of this research, which is why the approach by Aerts and Gabora [2005a] is selected:

The calculation of the prototype classifies exemplars by the number of observations, designating the most observed exemplar as prototype. Which observations and exemplars e are considered is given by the contextualized concept E in Equation 4.29. Function p is defined over a contextualized concept and an exemplar, and calculates a number $p(E,e) : e \rightarrow \mathbb{R}$ in the range $[0,1]$ called *contextual typicality*. The contextual typicality is calculated by dividing the number of observations for one exemplar $\#e$ by the overall number of observations for all included exemplars in the contextualized concept $\#E$ as given in Equation 4.29.

$$p(E,e) = \frac{\#e}{\#E} \tag{4.29}$$

The prototype equals the exemplar with the highest contextual typicality value for one contextualized concept calculated with Equation 4.30.

$$q(E,e_i) = \max_{e_i \in E} \left(p(E,e_i)\right) \tag{4.30}$$

The number of observations for each exemplar for contextualized concepts E_i is presented as *observation table*. The tabular form shows contextualized concepts in columns and the number of references to exemplars in columns. This table is called observation table because it classifies /lists exemplars as they are observed in reality. For the database of the hypothetical web service example from above, Table 4.4 shows the observation table for the exemplars considered for **restaurant** given in Figure 4.7.

exemplars	contexts						
	$\bot$	*in Vienna*	$p($*in Vienna*$)$	*serving pizza*	$p($*serving pizza*$)$	$\top$	$p(\top)$
Plachutta	0	6	$\frac{6}{14} = 0.43$	1	$\frac{1}{25} = 0.04$	6	$\frac{6}{30} = 0.20$
Pizza Hut	0	0	$\frac{0}{14} = 0.00$	11	$\frac{11}{25} = 0.44$	11	$\frac{11}{30} = 0.37$
Cantinetta	0	3	$\frac{3}{14} = 0.21$	3	$\frac{3}{25} = 0.12$	3	$\frac{3}{30} = 0.10$
Vapiano	0	5	$\frac{5}{14} = 0.36$	10	$\frac{10}{25} = 0.40$	10	$\frac{10}{30} = 0.33$
$\#E_i$	0	14	1	25	1	30	1

Table 4.4: Observation table for the restaurant example

The observation table does not have to include atomic context representations. If it included only atomic context representations, the numbers in the table would sum up to the $\top$ context. In this example, the numbers for the $\top$ context for Plachutta and Vapiano are not the sum: $0+6+1 \neq 6$ and $0+5+10 \neq 10$. The reason is that both contexts *in Vienna* and *serving pizza* are not atomic representation contexts. The exemplars are counted for each of the two contexts, but only *once for the top* context, summing up to 6 and 10 observations as shown in Figure 4.7.

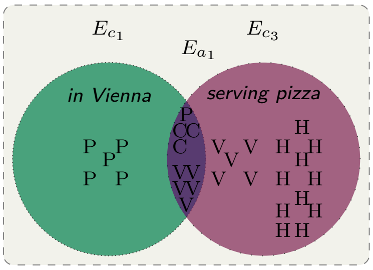


Figure 4.7: Observation Table 4.4 represented with Venn diagrams

Prototypes are picked from contextual typicality values included in observation tables. To calculate the prototype, the observation table includes contextual typicality, with the highest values designating the respective prototypes. For example, in Table 4.4 the prototype for context *in Vienna* is Plachutta, having a contextual typicality of 0.43; and for context *serving pizza* it is Pizza Hut, having a contextual typicality of 0.44. The observation table lists all contextual typicality values for each exemplar supporting the designation of a prototype.

The prototype calculation function equals the interpretation relation in the context-enriched semiotic triangle. Here it is *assumed* that the interpretation of the word equals the prototype of the concept.

4.3 Conclusion

In this section the mathematical structure for (i) context, (ii) concepts, and (iii) the mapping between both is presented.

Context inherits the partial order relation $\leq$ read as "is more selective than or equally selective as". For the order relation the properties of asymmetry, reflexivity, and transitivity are stated. For three contexts, greater selective and least selective contexts can be found. One unique greatest context $\top$ and one least context $\bot$ are included in the context structure. These properties of context enable the mathematical structure of a partially ordered set for context.

Contexts can be combined by the binary functions conjunction and disjunction. The conjunction of two contexts results in a context that is "more selective than or equally selective as" the two constituents. The disjunction of two contexts results in a context that is "less selective than or equally selective as" the two constituents. Both operations satisfy idempotency, commutativity, associativity, consistency, and the isotone law.

Contexts form a lattice structure, including a complement operation. For every context in the lattice such a complement context can be found. Specific rules for the complement of $\bot$ and $\top$ are included. By these operations, a lattice structure for context is applied. Atomic context representations are introduced to build all other contexts by disjunction.

A mapping from the context lattice to concepts is introduced that allows us to distinguish between multiple interpretations of a concept.

1. Concepts are modeled with a set labeled E.

2. The mapping of contexts in lattice C to a concept is given by function $m : C \mapsto E$ and used to partition the concept into *contextualized concepts*. Contextualized concepts are subsets of the concept set representing observations only valid for one context.

3. Multiple interpretations of a concept are calculated using contextualized concepts. It is assumed that the interpretation corresponds with the prototype of the contextualized concept.

4. The prototype of a contextualized concept is calculated with two functions. The first function calculates a contextual typicality value $p(E, e) = \frac{\#e}{\#E}$ for one exemplar e. The second function applies the former function to each exemplar in a contextualized concept, and designates the exemplar with the highest contextual typicality value as the prototype: $q(E, e_i) = \max\limits_{e_i \in E} \big(p(E, e_i)\big)$.

Summarizing the content of this chapter in the perspective of the constitutive rule of Searle:

- X represents a concept externalized by a sign (text, image, sound, etc.)

- Y represents the concept and its interpretation, formalized as contextualized concepts with prototype calculation functions where the prototype equals the interpretation; and

- C determines the contextualized concept.

5

Implementation of the Context Algebra

In this chapter the context algebra and the mapping introduced in Chapter 4 are implemented. The goal of this implementation is to present a realization of the context algebra. In contrast to the context algebra that "avoids mentioning unnecessary details" the formal implementation "avoids imprecision" [Loeckx et al., 1997]. Haskell [Marlow, 2010] is selected as the programming language, because it allows to check the consistency of the context algebra using the static type system. Additionally, the syntax of Haskell is almost identical to the mathematical notations introduced in context algebra in Section 4.

The implementation is structured into several modules according to the mathematical structure. An overview of all modules is given in Figure 5.1. In the middle of Figure 5.1 there is the module `Mapping`, integrating context and concept implementation. The reason for it being in the middle is that it relies on context and on concept implementation. Context implementation is divided in two modules, module `ContextAlgebra`, stating abstract classes, and module `ContextAlgebraSetInstance`, instantiating the abstract classes. Concept is implemented by the module `Concept`. Contextualized concepts are specified by module `ContextualizedConcept` and instantiated in module `NominalExemplars`.

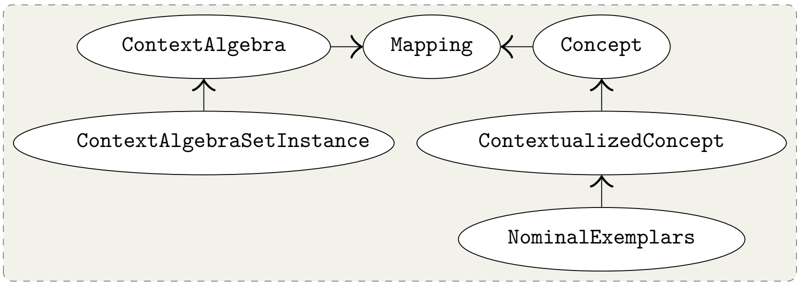


Figure 5.1: Haskell modules implementing the Context Algebra

The General Context Operation introduced in Section 1.3 is implemented on behalf of the context algebra implementation. Use of the General Context Operation is demonstrated using the introductory example "Let's meet at a restaurant" presented in Section 1.1.

5.1 *Context Algebra*

The ContextAlgebra module is built upon several abstract classes, each representing a mathematical structure. The abstract classes given in Section 5.1 provide a polymorphic interface that can have multiple implementations. These are presented and discussed before an implementation using sets is introduced.

Abstract Classes

All necessary abstract classes needed to build the context algebra are given in Figure 5.2. Each class includes functionality that is closely related, *e.g.* partial order relations are part of module PartialOrder. Attention is drawn to specifying the data types as polymorphic data types in order to enable the highest flexibility for the instances.

The ContextLattice class integrates all other classes and builds the context lattice. To establish the ContextLattice the classes PartialOrder, Universe, Bound, and Complement are prerequisites. Each of the classes includes functions (given below the class name in Figure 5.2) that other classes rely on, indicated by arrows in Figure 5.2. To highlight the abstract classes in the algorithms they are written in red font. All classes are included in module Context Algebra given in Code Fragment A.1.

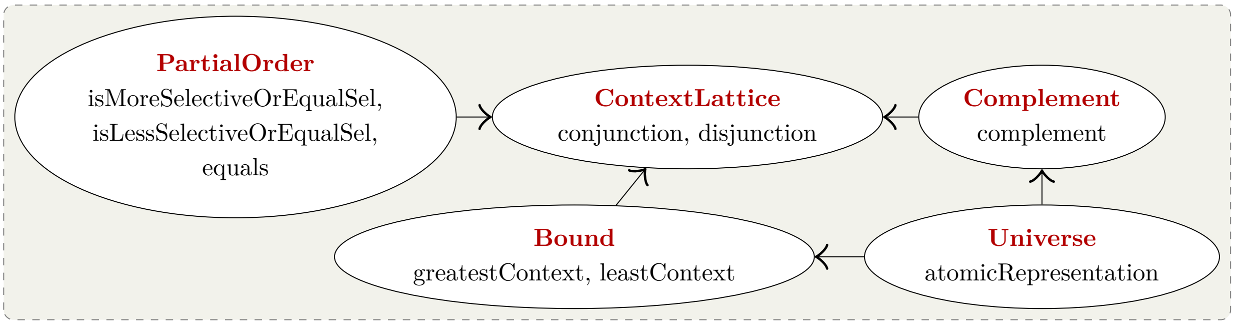


Figure 5.2: Abstract classes constituting the Context Lattice

The PartialOrder class includes three functions to relate two contexts and compare two contexts for equality. These functions are isMoreSelectiveOrEqualSel, isLessSelectiveOrEqualSel and equals. Function isMoreSelectiveOrEqualSel represents the partial order relation $\leq$ introduced in Equation 4.1 and the converse function introduced in Equation 4.2 is represented by isLessSelectiveOrEqualSel. The definitions for these functions are given in Code Fragment 5.1. Each function takes two contexts as input parameters and evaluates to a Boolean value. The equals function makes use of these two functions; if both evaluate True, then the two contexts are considered equal.

Code Fragment 5.1: Abstract class Partial Order

```
class PartialOrder c where           -- ^ c refers to a polymorphic context type
  isMoreSelectiveOrEqualSel :: c     -- ^ first input context
                            -> c     -- ^ second input context
                            -> Bool  -- ^ true if first context is more or equal selective
                                     --   compared to the second context

  isLessSelectiveOrEqualSel :: c     -- ^ first input context
                            -> c     -- ^ second input context
                            -> Bool  -- ^ true if first context is less or equal selective
                                     --   compared to the second context

  -- | compares two contexts and returns true if they are equal,
  --   includes a default implementation
  equals :: c     -- ^ first input context
         -> c     -- ^ second input context
         -> Bool  -- ^ true if the first context equals to the second context
  equals c1 c2 = c1 `isMoreSelectiveOrEqualSel` c2
                 &&
                 c1 `isLessSelectiveOrEqualSel` c2
```

The `Bound` class includes functions defining the ⊤ and the ⊥ context for the partial order relation. The `greatestContext` function implements the ⊤ element of the partial order relation as given in Equation 4.9. Analogously, the `leastContext` function implements the ⊥ element as given in Equation 4.8. The definition of the class including the two functions is given in Code Fragment 5.2.

Code Fragment 5.2: Abstract class Bound

```
class  Bound c where
  -- | returns the universal least context, bottom
  leastContext :: c -- ^ universal least context
  -- | returns the universal greatest context
  greatestContext :: c -- ^ universal greatest context, top
```

The `Universe` class includes a function that returns all atomic representation contexts. It is named `atomicRepresentation` and is necessary to establish the `greatestContext` function from the `Bounded` class. The definition for this class is given in Code Fragment 5.3.

Code Fragment 5.3: Abstract class Universe

```
class Universe c where
  -- | returns all atomic representation contexts
  atomicRepresentation :: [c] -- ^ list including all atomic representation contexts
```

The `Complement` class includes the function `complement`. This unary function gets a context and evaluates to the complement of this context as given in Equation 4.21. The class definition is given in Code Fragment 5.4.

Code Fragment 5.4: Abstract class Complement

```
class Complement c where
  -- | complements the input context
  complement :: c -- ^ input context to be complemented
             -> c -- ^ complemented context
```

The `ContextLattice` class integrates all the previous classes to specify the combination functions `conjunction` and `disjunction`. The definition is given in Code Fragment 5.5.

Code Fragment 5.5: Abstract class ContextLattice

```haskell
class (PartialOrder c, Bound c, Complement c) => ContextLattice c where
  -- | returns the conjunction of first and second context
  conjunction :: (Bound c, Complement c, PartialOrder c) =>
                    c -- ^ first input context
                 -> c -- ^ second input context
                 -> c -- ^ conjunct result context
  -- | returns the disjunction of first and second context
  disjunction :: (Bound c, Complement c, PartialOrder c) =>
                    c -- ^ first input context
                 -> c -- ^ second input context
                 -> c -- ^ disjunct result context
```

Several attempts exist for implementing a lattice structure for an abstract data type in the functional paradigm. For example, the implementation from Jones [1992] shows a lattice established from abstract classes similar to this approach. Another implementation for a lattice is available on the Haskell library repository hackage: `https://hackage.haskell.org/package/lattices`. The article by Jones and the online library are used for insights into the implementation of the context algebra.

Implementation

All the abstract classes mentioned above are included in module `Context-Algebra`, which is instanced in module `ContextAlgebraSetInstance`. The requirement for the implementation is that functions are executable without specifying the data type for context. To achieve this requirement, a container that holds context is necessary.

The abstract data type `Data.Set` is selected as container for context. With this selection all the functions included in the abstract classes have to be implemented based on the functions available in `Data.Set`. For convenience, the `Data.Set` container is referred to as set.

A context is implemented with polymorphic data types included in a set. Code Fragment 5.6 shows the implementation for context having a polymorphic type parameter `c`. The function `createContext` creates a context representation by taking the context and creating a singleton set. It is used to represent atomic context representations, where combined contexts are created by the disjunction of the atomic context representations. The full implementation is given in Code Fragment A.2.

Code Fragment 5.6: Data type representing context

```haskell
import qualified Data.Set as Set

type Context c = Set.Set c

-- ^ creates atomic context representations
createContext :: c             -- ^ polymorphic type for context
              -> Context c     -- ^ context representation
createContext  = Set.singleton
```

The instance of the PartialOrder class implements functions isMoreSelectiveOrEqualSel and isLessSelectiveOrEqualSel using set functions. What can be a feature that distinguishes two sets and can establish a partial order? For the relation isMoreSelectiveOrEqualSel all contexts included in the first set have to build a *subset* of the contexts included in the second set. This is implemented with the Set.isSubsetOf function given in Code Fragment 5.7. For the contrary relation isLessSelectiveOrEqualSel, two cases are distinguished by guards (indicated by the | symbol) shown in Code Fragment 5.7. The first guard is for the case of two contexts being equal, in which case this relation is true. The second guard is for all other cases, in which case it is the negation of the isMoreSelectiveOrEqualSel function. The equals function does not have to be implemented because it already has a default implementation in the abstract class.

Code Fragment 5.7: Instance of the abstract class PartialOrder

```haskell
instance (Ord c) => PartialOrder (Context c) where
  isMoreSelectiveOrEqualSel = Set.isSubsetOf

  isLessSelectiveOrEqualSel c1 c2
   | c1 == c2 = True
   | otherwise = not $ c1 `isMoreSelectiveOrEqualSel` c2
```

An instance for the Universe module cannot be given here, because it depends on the use case. An instance of the class has to implement the function atomicRepresentation, returning all atomic representations used in the specific use case. An example use case is given for the restaurant example in Table 4.3, showing the Universe instance in Code Fragment A.7.

The instance of the Bound class implements the functions greatestContext and leastContext. The implementation of the greatestContext function relies on the function atomicRepresentation given in the Universe class. The result of the greatestContext function is built by the union of all atomic contexts given in Code Fragment 5.8. The leastContext function is represented by an empty set.

Code Fragment 5.8: Instance of the abstract class Bound

```haskell
instance (Universe c, Ord c) => Bound (Context c) where
  greatestContext = Set.unions . map createContext $ atomicRepresentation

  leastContext = Set.empty
```

Within the Complement instance the function complement is implemented. The complement selects all the contexts, but not the input context. To implement this behavior, all contexts have to be found and the input context has to be removed. This is implemented with the set function difference between the greatestContext and the input context, shown in Code Fragment 5.9.

Code Fragment 5.9: Instance of the abstract class Complement

```haskell
instance (Universe c, Ord c) => Complement (Context c) where
  complement = Set.difference greatestContext
```

The class `ContextLattice` uses functions provided by the previous classes to implement the functions `conjunction` and `disjunction`. Both functions include several special cases that have to be dealt with. These special cases are given by the Equations 4.12, 4.15, 4.17, 4.18, 4.19, 4.20 and 4.24, and are checked by the guards at the beginning of the functions given in Code Fragment 5.10. If no special case applies, the conjunction and disjunction function have to be implemented as functions available on sets. Possible candidates for the implementation are the union and the intersection function, which indeed are able to represent conjunction and disjunction as given in Code Fragment 5.10.

Code Fragment 5.10: Instance of the abstract class ContextLattice

```haskell
instance (Universe c, Complement c, Ord c)=> ContextLattice (Context c) where
  disjunction c1 c2
   | c1 `equals` leastContext = c2                    -- Equation 4.17
   | c2 `equals` leastContext = c1                    -- Equation 4.17
   | c1 `equals` greatestContext = greatestContext    -- Equation 4.18
   | c2 `equals` greatestContext = greatestContext    -- Equation 4.18
   | c1 `equals` c2 = c1                              -- Equation 4.12
   | complement c1 `equals` c2 = greatestContext      -- Equation 4.24
   | complement c2 `equals` c1 = greatestContext      -- Equation 4.24
   | c1 `isMoreSelectiveOrEqualSel` c2 = c2           -- Equation 4.15
   | otherwise =  c1 `Set.union` c2                   -- Equation 4.20

  conjunction c1 c2
   | c1 `equals` leastContext = leastContext   -- Equation 4.17
   | c2 `equals` leastContext = leastContext   -- Equation 4.17
   | c1 `equals` greatestContext = c2          -- Equation 4.18
   | c2 `equals` greatestContext = c1          -- Equation 4.18
   | c1 `equals` c2 = c1                       -- Equation 4.12
   | complement c1 `equals` c2 = leastContext  -- Equation 4.24
   | complement c2 `equals` c1 = leastContext  -- Equation 4.24
   | c1 `isMoreSelectiveOrEqualSel` c2 = c1    -- Equation 4.15
   | otherwise = c1 `Set.intersection` c2      -- Equation 4.19
```

5.2 *Mapping from Context to Concepts*

The mapping function introduced in Equation 4.25 is specified as an abstract class and implemented in module `Mapping`. A prerequisite for the mapping implementation is an implementation for a concept.

Concept implementation

For the concept implementation, two containers are *invented*. The first container is used as a connection to context, while the second container is used to build the concept.

The first container is built by an abstract data type labeled `Observation`. `Observation` stores a tupel of a context data type `c` and an exemplar of data type `e` given in Code Fragment 5.11. The exemplar represented by data type `e` can be a logical axiom [McCarthy, 1987], an instance of a concept [Aerts and Gabora, 2005a], an abstract class of exemplars [Hahn and Frank, 2014], or something else influenced by context. The only requirement is that exemplar `e` is observed in context `c`. As this connection has to be observed, this combination of context and exemplar is called `Observation`.

Code Fragment 5.11: Data type Observation combining an exemplar and a context

```haskell
data Observation c e = Obs (c,e) deriving (Show,Eq,Ord)
```

The data type of `Observation` is given in Code Fragment 5.11. It derives the type classes `Show` (to have a String representation for printing it on the command line), `Eq` (==) and `Ord` for comparisons (greater, lower) necessary for the next container.

The second container is an abstract data type labeled `Concept` storing multiple `Observation`. A `Concept` includes all observations considered in a particular use case. To implement a `Concept` the abstract data type `MultiSet` is selected, because a `MultiSet` can include same entries multiple times in contrast to a set that can hold equal entries once. `MultiSet` is an external library available on Hackage: `https://hackage.haskell.org/package/multiset` and is an extension of `Data.Map`. The definition of the `Concept` data type is given in Code Fragment 5.12, it derives the type class `Show` in order to have a String representation.

Code Fragment 5.12: Data type representing a concept

```haskell
data Concept c e = C (MultiSet (Observation c e)) deriving (Show)

createConcept :: Observation c e
              -> Concept c e
createConcept = C . MultiSet . singleton
```

Functions to manipulate a `Concept` are implemented. To establish a `Concept` an `Observation` is created and added to the `Concept`. The `Observation` is created with the constructor of the data type `Obs` and is changed to a `Concept` by function `createConcept`. This new concept can be added to an existing `Concept` using the method `addConcept`. This and other functions are provided in the `Concept` module given in Code Fragment A.3 in Appendix A.

Abstract class

The abstract class `Mapping` specifies the mapping function `m` given in Equation 4.25. The function takes a context `c` and a `Concept c e` as input, and returns a `Concept c e`, as given in Code Fragment 5.13. To map the context to the concept it is necessary that the polymorphic data type for the context equals the type for the concept given in Code Fragment 5.13.

Code Fragment 5.13: Abstract class Mapping

```haskell
class  Mapping c e where -- data type c equals for context and concept
  m :: (Ord c  , Ord e, Bound c, PartialOrder c) =>
       c             -- ^ more selective context or equal selective context
    -> Concept c e   -- ^ concept including Observations now influenced with context c
    -> Concept c e   -- ^ concept with Observations for contexts c and more selective ones
```

Implementation of the Mapping: $m : C \mapsto E$

The implementation of the abstract class `Mapping` distinguishes two cases introduced by Equation 4.25. The two cases are distinguished in the implementation by guards `|` that enable pattern matching. The first case matches if the actual context `ctx` equals the `leastContext` and results in a `Concept` without any `Observation`, given in Code Fragment 5.14. In the second case, only those `Observations` are returned that are observed in a “more selective as or equally selective as” context compared to the given context.

Code Fragment 5.14: Instance of the abstract class Mapping

```
instance Mapping c (Concept c e) where
  m ctx concept
    | leastContext `equals` ctx = emptyConcept
    | otherwise =  fromObservationList
                   [obs | obs <- toObservationList concept
                        , getContext obs `isMoreSelectiveOrEqualSel` ctx]
```

5.3 Contextualized concepts

A contextualized concept is created by the mapping of a context to a concept. For a contextualized concept a function to calculate a prototype is implemented. The implementation for this class depends on the type of the exemplars that are taken into account. An implementation for exemplars on a nominal measurement scale [Stevens, 1946] is provided in Code Fragment A.6.

Abstract class

The abstract class `ContextualizedConcept` defines the function to calculate a prototype given in Code Fragment 5.15. The function to calculate the prototype is defined as `calculatePrototype`, which operates on a concept and results in a tupel consisting of the prototype exemplar of type `e` and a contextual typicality of type `Double`. The abstract class definition is included in module `ContextualizedConcept` Code Fragment A.5.

Code Fragment 5.15: Abstract class ContextualizedConcept

```
class ContextualizedConcept c e where
  -- | returns a tupel including the prototype with its contextual typicality value
  calculatePrototype :: (Show e, Ord e) =>
                        Concept c e -- ^ contextualized concept including exemplars e
                     -> (e,Double)   -- ^ (prototypical exemplar,contextual typicality)
```

Implementation

The implementation of contextual typicality depends on the type of exemplars that is considered. An implementation of the abstract class `ContextualizedConcept` for exemplars categorized by a nominal measurement scale [Stevens, 1946] is presented in Code Fragment 5.16 included in module `NominalExemplars` presented in Code Fragment A.6.

The implementation of the `calculatePrototype` function relies on the function `rateObservations` that counts all observations for each exemplar, as mathematically introduced in Equations 4.29 and 4.30. The number of observations for each exemplar is divided by the total number of all observations made for this context. The result of the `rateObservations` function is a list including each exemplar equipped with a contextual typicality value. These contextual typicality values are compared, and the maximum contextual typicality is designated as the prototype in function `calculatePrototype`.

Code Fragment 5.16: Instance of the ContextualizedConcept class for exemplars on a nominal measurement scale

```
instance ContextualizedConcept c e where
  calculatePrototype  = List.maximumBy (compare `on` snd ) . rateObservations

rateObservations :: (Show e, Ord e) =>
                    Concept c e  -- ^ contextualized concept
                 -> [(e,Double)] -- ^ contextual typicality for all exemplars
rateObservations con = map (\(exemplar, occurrences) ->
                              (exemplar,fromIntegral occurrences /
                                        fromIntegral totalOccurrences) )
                       . Mset.toAscOccurList . Mset.fromList
                       . concatMap (\(O (_,e) ,o)-> replicate o e)
                       . toOccurrenceList $ con
  where totalOccurrences = numberExemplars con
```

5.4 *General Context Operation*

An idea of the General Context Operation was presented as pseudocode in Section 1.3, and is now implemented. For a better understanding the pseudocode is given here in Code Fragment 5.17.

Code Fragment 5.17: Pseudo code of an idea for a General Context Operation

```
function GeneralContextOperation(information, context)
    moreSpecificContext = makeMoreSpecificeContext(context)
    moreSpecificInformation = createInfoForMoreSpecificContext(information, moreSpecificContext)
return moreSpecificInformation
```

Implementation

The implementation of the General Context Operation is shown in Code Fragment 5.18. The input parameters of the `generalContextOperation` are extended by the actual context, which indicates in which context the concept is before executing the `generalContextOperation` operation. This actual context is combined with the new context to establish a `moreSelectiveContext`. The `moreSelectiveContext` is the equivalent to the `moreSpecificContext` variable in the pseudocode. This `moreSelectiveContext` context influences the concept `con` implemented as mapping `m`. The result of the mapping creates a contextualized concept according to the `moreSelectiveContext`. This contextualized concept includes observations that are valid in the `moreSelectiveContext` which is according to the pseudo code algorithm moreSpecificInformation.

Code Fragment 5.18: Implementation of the General Context Operation

```
generalContextOperation :: (Ord c, Ord e, Bound c, PartialOrder c, ContextLattice c
                            , Mapping c e) =>
                            c
                         -> c
                         -> Concept c e
                         -> Concept c e
generalContextOperation actualCtx newCtx con = m moreSelectiveContext con
 where moreSelectiveContext =  actualCtx `conjunction` newCtx
```

The general context operation presented in Code Fragment 5.18 combines two contexts with the conjunction operation. In the same fashion, a general context operation with a disjunction operation can be implemented. Here the conjunction is selected in order to formalize the introduction example in the next section. The example in the next section refines the context to even more selective contexts, where the conjunction combination operator is applicable.

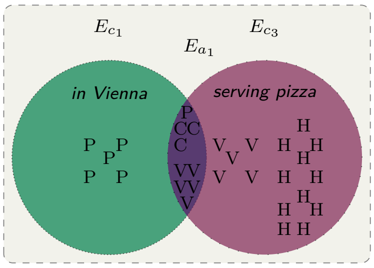


Figure 5.3: Contextualized concepts represented with Venn diagrams including exemplars

Example using the General Context Operation

The introduction example "Let's meet at a restaurant" is implemented using the `generalContextOperation`. In this example, two subjects negotiate about which restaurant to meet at. Restaurant is considered here as concept **restaurant** because it refers to many objects in reality. Context influence on the concept is used to reduce the number of references to objects in reality. One subject considers an online map that is equipped with a `generalContextOperation` and displays possible references to restaurants according to the context stated in the negotiation. The context of the negotiation changes as messages are exchanged, which is represented here as states. In addition to the introductory example a prototype for each state is obtained.

The Context lattice has to be initialized with atomic context representations. The influencing contexts are *in Vienna* and *in Vienna serving pizza*. To include the two contexts in the lattice it is initialized with the atomic context representations a_1= *in Vienna and serving pizza*, a_2= *in Vienna and not serving pizza* and a_3= *not in Vienna and serving pizza* given in Figure 4.5. The atomic context representations are implemented with the function `createContext` shown in Code Fragment 5.19. The context *in Vienna* is created by the disjunction of the atomic contexts a_1 and a_2 given in Code Fragment 5.19. The necessary instance for the `Universe` class is implemented using the three atomic context representations given in Code Fragment A.7 in Appendix A.

Code Fragment 5.19: Contexts included in the restaurant example

```
type CtxType = Context String

a1 = createContext "(in Vienna) and (serving pizza)"
a2 = createContext "(in Vienna) and not (serving pizza)"
a3 = createContext "not(in Vienna) and (serving pizza)"

inVienna = a1 `disjunction` a2
servingPizza =  a1 `disjunction` a3
inViennaOrServingPizza = a2 `disjunction` a3
```

Concept **restaurant** is initialized with observations of exemplars taken from Venn Diagram 5.3. Observations for the exemplars Plachutta, Pizza Hut, Vapiano, and Cantinetta are established following the number given in Venn Diagram 5.3. **Restaurant** is represented in the implementation as given in Code Fragment 5.20.

Code Fragment 5.20: Observations initializing concept "restaurant"

```
plachutta1 = O (a1, "Plachutta")
plachuttaV = replicate 5 $ O (a2, "Plachutta")
pizzahut   = replicate 11 $ O (a3, "Pizza Hut")
candinetta = replicate 3 $ O (a1, "Candinetta")
vapianoV   = replicate 5 $ O (a1, "Vapiano")
vapianoP   = replicate 5 $ O (a3, "Vapiano")

restaurant = unionConcept . map createConcept $ obs
 where obs =[plachutta1]++plachuttaV++pizzahut++candinetta++vapianoV++vapianoP
```

The first state of the discussion considers the sentence: "Let's meet at a restaurant". For this sentence, no context influences **restaurant**. Without context influence, a context from the context lattice has to be selected that has no effect on the concept, which is the `greatestContext`. For the `greatestContext` all restaurants available in the concept have to be considered. Applying the `generalContextOperation` with the `greatestContext` does not have any effect on the **restaurant**. In this state, the online map displays all restaurants available around the globe.

The negotiation proceeds with the sentence: "Are you in Vienna today?". The listener acknowledges the sentence with "Yes I'm in Vienna till Sunday.". Both sentences can be fused, which results in the next state the context algebra can process:

The second state is "Let's meet at a restaurant in Vienna". In this sentence, **restaurant** is influenced by the context *in Vienna*, resulting in the contextualized concept **restaurant@***in Vienna*. How this is calculated using `generalContextOperation` is shown in Code Fragment 5.21.

Code Fragment 5.21: Applying the General Context Operation for "restaurant" in context "in Vienna"

```
restsInVienna = generalContextOperation greatestContext inVienna restaurant
prototypeInVienna = calculatePrototype restsInVienna
```

The output includes only restaurants that are located in Vienna. The online map is refreshed according to this outcome, and now shows only restaurants in Vienna, as shown in Figure 1.2. Additionally, the prototype for the contextualized concept is determined as given in Code Fragment 5.21, resulting in Plachutta with a contextual typicality of 0.43. The negotiation proceeds in the following exchange:

The third state is "Let's meet at a restaurant in Vienna serving pizza". The discussion is left out because it is not important how the third state is reached. In this state, **restaurant@***in Vienna* is influenced by *in Vienna serving pizza*, which is a more selective context compared to *in Vienna* resulting in contextualized concept **restaurant@***in Vienna* *serving pizza*. The calculation is given in Code Fragment 5.22 where the context *in Vienna serving pizza* is represented as variable `a1` (consider Code Fragment 5.19). The result is the contextualized concept **restaurant@***in Vienna serving pizza* which is shown in Figure 1.4. For the contextualized concept, the obtained prototype is Vapiano with a contextual typicality of 0.55.

Code Fragment 5.22: Applying the General Context Operation for “restaurant” in context “in Vienna serving pizza”

```
restsInViennaServingPizza = generalContextOperation inVienna  a1 restsInVienna
protoInViennaServingPizza = calculatePrototype restsInViennaServingPizza
```

Note that the third input parameter `restsInVienna` is the output of the previous state and includes only restaurants in Vienna. This implies that in further states only “more selective” contexts can be processed, *e.g.* **restaurants@***in Vienna serving pizza on a Sunday*. If a “less selective” context is applied, not every observation included in the initial concept is taken into account.

The negotiation has to proceed in order to agree on one restaurant, or both agree on a prototype. The goal here was to present the principle of applying the `generalContextOperation`, but if the whole negotiation is to be processed all further contexts have to be included in the context lattice. A self-contained algorithm of this example is included in Code Fragment A.7. Additionally, the prototype calculation mechanism is demonstrated, which obtains a prototype for every state of the negotiation. The prototype can be beneficial in this negotiation because if both subjects share a prototype, an additional reduction of references is reached.

5.5 Conclusion

The context algebra is implemented in this chapter using the functional language Haskell. A requirement for the implementation is the polymorphic property for data types. To guarantee polymorphic properties, abstract data types are introduced for context and concepts. This guarantees that a later application can rely on self-defined data types.

The implementation is structured in modules distinguishing abstract classes according to mathematical structures. The implementation relies on one module for the context algebra, one for the mapping, and one for the concept it is mapped to. The abstract classes are instanced using existing abstract data structures based on sets. For contextualized concepts, a function to calculate a prototype is included that relies on exemplars classified on a nominal measurement scale. All implementations of the modules are included in Appendix A, so that later researchers can reuse them.

The implementation of the general context operation shows a use case for context algebra. It is applied to model the example “Let’s meet at a restaurant” introduced in Section 1. Additionally, the benefit of the prototype calculation function is shown, which can help further reduce references to objects. The full implementation of the example including the general context operation is included in Code Fragment A.7 in Appendix A.

The general context operation is similar to the filter operation used by Crease [2013] to represent geographic relevance. Crease [2013] develops a mechanism to represent geographic relevance in mobile applications in multiple steps, beginning with a filter operation. His filtering approach was a manually applied contextual analysis based on space, time and activity. This manual process can be exchanged with the general context operation to establish automatic filtering. The general context operation is also more general because it is not limited to space, time and activity. The usage of the general context operation to generate geographic relevance is beneficial for two reasons (i) it is more general than the contextual analyses and (ii) it is implemented and can be executed by a computer.

6
Evaluation of the Implementation

In this chapter, the implementation is tested to verify that it satisfies all laws of the context algebra outlined in Section 4.1. The evaluation concentrates on the functions included in the abstract classes specified in module `ContextAlgebra`, which are implemented in module `ContextAlgebraSetInstance`.

The evaluation of the implementation is achieved with property test functions designed to verify algebraic laws of functions. The test functions are created on behalf of the functions in module `ContextAlgebra`, including abstract classes only. Test functions created for functions defined in abstract classes without an implementation are beneficial because no implementation details are considered. The test functions make use of the functionality offered by the *QuickCheck*[1] library, which is specially designed for algebraic property testing [Claessen and Hughes, 2011].

Input contexts used by the QuickCheck library are randomly generated to ensure that many input combinations are tested. The randomly generated contexts are taken from the existing formalization of the introduction example included in module `GeneralContextOperation` Code Fragment A.7.

Automatic execution of the property test functions is done by the module `ContextAlgebraEvaluation`, which integrates the functionality of several modules as shown in Figure 6.1. Within the QuickCheck library, the tests are sequentially executed and results are presented in a detailed report for each property test.

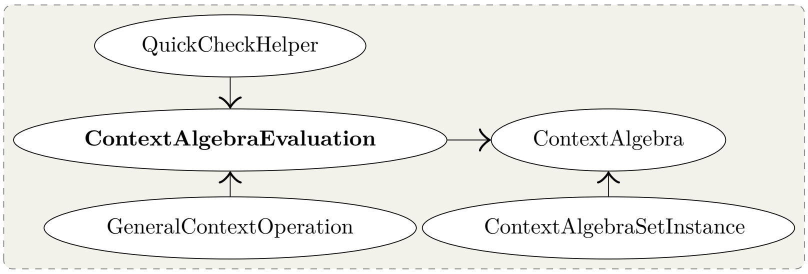


Figure 6.1: Haskell modules establishing the algebraic tests

[1] `https://hackage.haskell.org/package/QuickCheck`

6.1 Property tests for algebraic laws

Property tests are implemented on behalf of the abstract classes specified in module `ContextAlgebra` because each instance of a context algebra has to satisfy the algebraic tests. The functions defined in the abstract classes include all necessary information to create the property tests.

Each algebraic law given in the context algebra in Section 4.1 is proven by a unique property test. For example, the algebraic law of idempotency (Equation 4.12) of the function `conjunction` is tested by `prop_isIdempotent` given in Code Fragment 6.1. In this algorithm, the advantage of using Haskell can be observed because the implementation of the test is identical to the mathematical formulation in Equation 4.12, aside from different names for the functions.

A property test has to evaluate either to a `Bool` or to a `Property` data type to be used by the QuickCheck library. A property test evaluates either to `True` if the test is passed successfully, or to `False` otherwise.

Code Fragment 6.1: Implementation of a property test for the idempotent law

```
prop_isIdempotent :: (PartialOrder c, ContextLattice c) =>
                     c
                     -> Bool
prop_isIdempotent c =  (c `conjunction` c) `equals` c
                       &&
                       (c `disjunction` c) `equals` c
```

The property tests are structured according to the abstract classes they belong to, *e.g.* for class `PartialOrder` the property tests `prop_isAsymmetric` and `prop_isTransitive` are included, as shown in Code Fragment 6.2. In the same fashion, tests for all other laws are implemented. The respective algorithms are included in the appendix; Algorithm A.8 implements tests for the `Bound` class, Algorithm A.9 for the `Context Lattice` class, Algorithm A.10 tests including `greatest` and `least` context, and Algorithm A.11 implements tests for the `Complement` class.

Some laws have prerequisites for input contexts that have to be checked before executing the test. For example, the property test `prop_isTransitive` (included in Code Fragment 6.2) has the prerequisite that context c_1 is more selective than or equally selective as context c_2, and the same is also true for $c_2 \leq c_3$. In order to ensure the prerequisites, QuickCheck offers the operator `==>` which checks first the input contexts and executes the test only if the condition is satisfied.

Each executed property test results in a report that can be customized to ensure that every possible input combination is tested. A default report of an execution is stated in Code Fragment 6.3. A detailed report includes information about the number of contexts used as input for a predefined classification. Such a classification is established for the property test `prop_isAssociative` relying on Equation 4.14. The classification for this test is the ordering of input con-

Code Fragment 6.2: Implementation of property tests for the laws included in the PartialOrder abstract class

```
-- * PartialOrder relation tests

-- | test if two contexts are asymmetric
-- input restriction, only equal contexts are tested
prop_isAsymmetric :: (PartialOrder c, Show c) =>
                     c        -- ^ first context
                  -> c        -- ^ second context
                  -> Property -- ^ True if the two contexts are asymmetric
prop_isAsymmetric c1 c2 = (c1 `equals` c2) ==>
                     c1 `isMoreSelectiveOrEqualSel` c2
                     &&
                     c1 `isLessSelectiveOrEqualSel` c2

-- | test if three contexts are transitive
prop_isTransitive :: (PartialOrder c) =>
                      c        -- ^ context one
                   -> c        -- ^ context less selective than context one
                   -> c        -- ^ context less selective than context two
                  -> Property -- ^ True if three contexts are transitive
prop_isTransitive c1 c2 c3 = (c1 `isMoreSelectiveOrEqualSel` c2
                              &&
                              c2 `isMoreSelectiveOrEqualSel` c3) ==>
                              c1 `isMoreSelectiveOrEqualSel` c3
```

texts which is achieved by function `classify` provided by QuickCheck. The `classify` function takes a classification criterion, *e.g.*`(c1 `isMoreSelectiveOrEqualSel` c2)`, and a label for this classification `"c1 <= c2"` as given in Code Fragment 6.4. The used labels show the mathematical symbol for the partial order relation, `<=` for the `isMoreSelectiveOrEqualSel` relation and `>=` for the `isLessSelectiveOrEqualSel` relation, the reason for being that they consume less console output space. For the associative law, six classifications are established as given in Code Fragment 6.4. An example report produced by executing the test is included in Code Fragment 6.8. The classification of input contexts included in the detailed report ensures that all possible context combinations are used to test the claimed laws.

Code Fragment 6.3: Default report of an executed property test

```
is asymmetric property:
+++ OK, passed 1000 tests.
```

6.2 *Random context generation for tests*

Randomly generated input contexts are used as inputs to execute property tests. The contexts obtained from the introductory example “Let’s meet at a restaurant” are: $\top$, $\bot$, a_1, a_2, a_3, $a_1 \vee a_2$, $a_1 \vee a_3$, $a_2 \vee a_3$; the ordering is given in Figure 4.5. These contexts are formalized to represent the introduction example in the `GeneralContextOperation` module as given in Code Fragment 5.19. The random selection of input contexts is achieved by an instance of the abstract class `Arbitrary` offered by QuickCheck. The instance has to implement the function `arbitrary` which can be achieved in multiple ways. The way chosen here uses the function `elements` offered by QuickCheck, which randomly selects an element from a list of contexts defined in Code

Code Fragment 6.4: Customized property test to classify input contexts

```
prop_isAssociative :: (PartialOrder c, Bound c, Complement c, ContextLattice c) =>
                   c          -- ^ context c1
                   -> c       -- ^ context c2
                   -> c       -- ^ context c3
                   -> Property -- ^ True if the three input contexts are associative
prop_isAssociative c1 c2 c3 = classify (c1 `isMoreSelectiveOrEqualSel` c2) "c1 <= c2" $
                              classify (c1 `isLessSelectiveOrEqualSel` c2) "c1 >= c2" $
                              classify (c1 `isMoreSelectiveOrEqualSel` c3) "c1 <= c3" $
                              classify (c1 `isLessSelectiveOrEqualSel` c3) "c1 >= c3" $
                              classify (c2 `isMoreSelectiveOrEqualSel` c3) "c2 <= c3" $
                              classify (c2 `isLessSelectiveOrEqualSel` c3) "c2 >= c3" $
                              m `equals` n
                              &&
                              o `equals` p
  where m = c1 `conjunction` (c2 `conjunction` c3)
        n = (c1 `conjunction` c2) `conjunction` c3
        o = c1 `disjunction` (c2 `disjunction` c3)
        p = (c1 `disjunction` c2) `disjunction` c3
```

Fragment 6.5.

Code Fragment 6.5: Instance of the Arbitrary class for the Context Algebra

```
import GeneralContextOperation

instance Arbitrary CtxType where
  -- | input contexts for the QuickCheck tests
  arbitrary = elements [leastContext
                       ,greatestContext
                       ,a1
                       ,a2
                       ,a3
                       ,inVienna
                       ,servingPizza
                       ,inViennaOrServingPizza]
```

6.3 *Automatic test execution and resulting reports*

QuickCheck has to determine how many random input contexts are needed to execute the test, which is automated by wrapper functions. For example, the property function `prop_isTransitive` (included in Code Fragment 6.2) takes three input contexts wrapped by function `qCheckTernary` as shown in Code Fragment 6.6. Several versions of wrapper functions are implemented, distinguishing the input of needed input contexts; the ones that result in `Bool` and others that result in `Property` are included in module `QuickCheckHelper` given in Code Fragment A.14.

The wrapper functions establish the connection to QuickCheck. Execution is achieved through function `quickCheckWith`, as shown in Code Fragment 6.6. The number of executions is set via the value `numberTests` to 1000, to ensure that each possible input combination is tested.

Every wrapped property test is executed sequentially in the main function implemented in module `ContextAlgebraEvaluation` in Algorithms A.12 and A.13. The wrapping of the property test `prop_isAssociative` is represented as function `testisAssociate` given at the

Code Fragment 6.6: Wrapper functions for QuickCheck execution

```
numberTests=1000

qCheckTernaryP  ::  (Arbitrary c, Show c) =>  (c -> c -> c -> Property) -> IO()
qCheckTernaryP   = quickCheckWith stdArgs { maxSuccess = numberTests }
```

top of Code Fragment 6.7. Every other test (included in Code Fragments 6.2, A.8, A.9, A.10, A.11) is also wrapped in a manner given in Algorithm A.12. All wrapped tests are executed sequentially in the `main` function, a part of which is included in Code Fragment 6.7 and its entirety in Algorithms A.12 and A.13 in Appendix A.

Code Fragment 6.7: Part of the main executable connecting QuickCheck with the property tests

```
testisAssociative  = qCheckTernaryP
                        (prop_isAssociative  :: CtxType-> CtxType-> CtxType-> Property)

main = sequence_  [putStrLn "is associative property:"
                  ,testisAssociative
                  ...
                  ,
                  ,putStrLn "test combination with complement:"
                  ,testwithCombBoundComp]
```

The report generated from the execution of the `main` function shows that the implementation satisfies all laws. For the execution of the property tests the instance `ContextAlgebraSetInstance` of the context algebra is selected. The created report in Code Fragment 6.8 provides evidence that each test is executed, and that the set implementation respects all laws given in the context algebra Section 4.

6.4 Conclusion

The set implementation of the context algebra was successfully proven to respect all algebraic laws stated by the context algebra. The test for algebraic laws is executed via separated property tests for each algebraic law. To ensure that the test is executed with many different input combinations, randomly generated context combinations obtained from the introduction example “Let’s meet at a restaurant” are used. Classifications of the input contexts ensure that every line in the implementation is covered. The automatic execution of all tests resulted in a successful report, which concludes that the set implementation satisfies all laws stated in the context algebra.

Code Fragment 6.8: Report of the execution of all property tests

```
λ> main
is asymmetric property:
+++ OK, passed 1000 tests.
is transitive property:
+++ OK, passed 1000 tests.
is least context property:
+++ OK, passed 1000 tests.
is greatest context property:
+++ OK, passed 1000 tests.
is idempotent property:
+++ OK, passed 1000 tests.
is commutative property:
+++ OK, passed 1000 tests:
56% c1 >= c2
30% c1 <= c2
12% c1 <= c2, c1 >= c2
is associative property:
+++ OK, passed 1000 tests:
21% c1 >= c2, c1 >= c3, c2 >= c3
10% c1 <= c2, c1 >= c3, c2 >= c3
 9% c1 >= c2, c1 >= c3, c2 <= c3
 9% c1 <= c2, c1 <= c3, c2 >= c3
 8% c1 <= c3, c1 >= c2, c2 <= c3
 8% c1 <= c2, c1 >= c2, c1 >= c3, c2 >= c3
 6% c1 >= c2, c1 >= c3, c2 <= c3, c2 >= c3
 4% c1 <= c2, c1 <= c3, c1 >= c2, c2 <= c3
 3% c1 <= c3, c1 >= c2, c1 >= c3, c2 >= c3
 3% c1 <= c3, c1 >= c2, c1 >= c3, c2 <= c3
 3% c1 <= c2, c1 <= c3, c2 <= c3, c2 >= c3
 3% c1 <= c2, c1 <= c3, c2 <= c3
 3% c1 <= c2, c1 <= c3, c1 >= c3, c2 >= c3
 2% c1 <= c3, c1 >= c2, c2 >= c3
 1% c1 <= c2, c1 <= c3, c1 >= c2, c1 >= c3, c2 <= c3, c2 >= c3
is consistent property:
+++ OK, passed 1000 tests.
is isotone property:
+++ OK, passed 1000 tests:
41% c1 >= c2, c1 >= c3
20% c1 <= c3, c1 >= c2
15% c1 <= c2, c1 <= c3
10% c1 <= c2, c1 <= c3, c1 >= c2
 9% c1 <= c3, c1 >= c2, c1 >= c3
 2% c1 <= c2, c1 <= c3, c1 >= c2, c1 >= c3
test combination with least context:
+++ OK, passed 1000 tests.
test combination with greatest context:
+++ OK, passed 1000 tests.
test double complement property:
+++ OK, passed 1000 tests.
test less then complement property:
+++ OK, passed 1000 tests.
test combination with complement:
+++ OK, passed 1000 tests:
86% c >= !c
13% c <= !c
```

7 Complexity analysis of the Implementation

A complexity analysis for the implementation is presented to answer questions regarding the computational complexity of the approach. The complexity class is determined regarding file size and execution time. To find a complexity class, the implementation is executed with several numbers of contexts and exemplars. Three measurements are made: (i) measurement of execution time for the context lattice generation; (ii) measurement of the time needed to calculate a prototype; and (iii) measurement of file size used to store a concept. The results are analyzed to determine bottlenecks, and are discussed to avoid worst-case scenarios. Recommendations are given for possible preprocessing steps.

A function determining the maximal number of contexts in a context lattice is presented. The function is used to determine whether the implementation has a bottleneck, and to provide a quick estimate about the number of contexts included in a context lattice. [Adomavicius and Tuzhilin, 2011] reported that a recommender system is influenced by multiple contexts at one time. For example, [Baltrunas et al., 2011] observed an influence of fourteen different contexts at one time for a recommender system about places of interest (POIs). If fourteen contexts have to be included in the context lattice, the function determines how many contexts are included in total.

7.1 Maximal number of contexts included in a context lattice

A function that describes the maximal number of contexts included in a context lattice is presented here. Hahn et al. [2016] included such a function for a context lattice and explained it as follows: the lattice consists of levels, where the number of contexts included in a level depends on the number of contexts included in a lower level starting from the atomic context representation. In summary, the number of context included in a lattice is determined by the number of atomic context representations.

The explanation by Hahn et al. [2016] is correct, but one element was accounted for too little in their formula. Including this element, the formula for the number of contexts is given by Equation 7.1 where c stands for the number of atomic context representations. The formula for a general lattice structure is given by, for example, Gill [1976, page 5] and is equal to the cardinality of power sets.

$$2^c \tag{7.1}$$

The dependency between atomic context representations and the number of contexts established from them is graphically expressed in Figure 7.1. The exponential increase of contexts included in the lattice with fourteen atomic contexts used in a recommender system for POIs [Baltrunas et al., 2011] results in 16384 contexts in total.

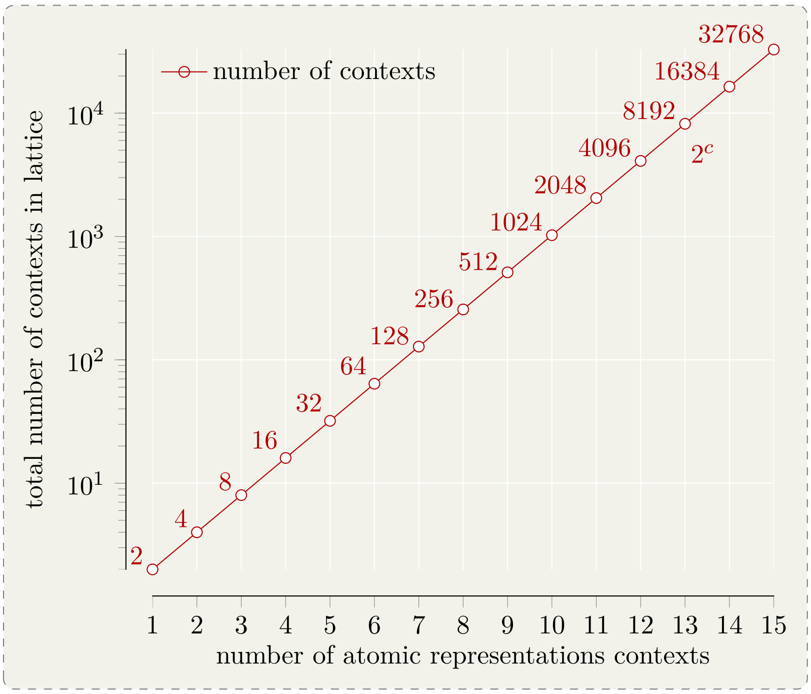


Figure 7.1: Maximal increase function of contexts included in a context lattice

The maximal increase function given in Equation 7.1 is categorized in the complexity class exponential (**EXP**). What implications follow for the implementation will be analyzed through measurements.

7.2 Complexity of used libraries

The complexity class for the implementation is also influenced by the libraries used. The implementation of the context algebra given in module `ContextAlgebraSetInstance` (Algorithm A.2) is based on the module `Data.Set`. From this library the functions `Set.subset`, `Set.singleton`, `Set.union`, `Set.intersection`, and `Set.difference` are used for the partial order relation, to establish a context, and to implement the disjunction, conjunction, and complement functions. The `Data.Set` library is implemented using the functional pearl presented by Adams [1993]. In the theoretical work, the set is implemented as a tree with worst complexity for union and intersection of $O(n + m)$ where n and m are the sizes of the elements included in the two sets. The Haskell implementation follows this complexity as documented online [1] and given in Table 7.1. The O notation of the `Data.Set` functions highlight that the functions are linear in the cardinality of the sets, aside from the `singleton` function which is constant.

context algebra function	function	complexity
`isMoreSelectiveOrEqualSel`	subset	$O(n+m)$
`createContext`	singleton	$O(1)$
`conjunction`	intersection	$O(n+m)$
`disjunction`	union	$O(n+m)$
`complement`	difference	$O(n+m)$

Table 7.1: Complexity class for used functions of the `Data.Set` implementation

A concept is implemented in module `Concept` (Algorithm A.3) using the library `MultiSet` [2] which also influences the complexity class of the context algebra. The documentation for `MultiSet` includes benchmark results that are equal to the ones given in Table 7.1.

In summary, the complexity from the used libraries is linear to the number of elements in the sets. The number of elements in the set is exponential, and depends on the atomic context representation. From a theoretical point of view, the implementation is of the exponential complexity class **EXP**.

7.3 Benchmark implementation

To measure execution time and file size, several modules are created to randomly generate contexts and exemplars, and to calculate the prototype. These additional modules are called benchmark implementation, and generate different workloads for the implementation. They differ from the evaluation implementation (Section 6) in their random generation of contexts and exemplars, and in the parametric execution of the combination functions.

The additional library *criterion* [3] is used to measure execution time. To eliminate influences originating from task scheduling or other background operations, the library criterion is used. The library repeats benchmarks independently while measuring execution time, and calculates the statistical *median* used in further analysis in this work. The functions *e.g.*`nfIO` are applied to evaluate the context algebra functions strictly.

Additional modules with functions including parameters are introduced to execute the context algebra functions with different number of contexts and exemplars. The first module `ConceptGenerator` given in Algorithm A.16 generates a context lattice and exports the result in a text file. This file is read by the second module `ProtoCalculator` given in Algorithm A.17 where prototypes for this context lattice are calculated. The third module `Main` given in Algorithm A.15 uses these two modules and implements benchmarks to measure execution time. The split into three modules is needed to measure only one functionality at a time, *e.g.* when calculating a prototype it is not of interest

[1] `https://hackage.haskell.org/package/containers-0.5.7.1/docs/Data-Set.html`

[2] `https://hackage.haskell.org/package/multiset`

[3] `http://hackage.haskell.org/package/criterion-1.1.0.0`

how long it takes to establish the context lattice.

The module ConceptGenerator (Algorithm A.16) includes the function createAndStoreConcept which creates a context lattice, including a mapping to a concept, and writes the result into a file. The number of contexts and exemplars is given by input parameters defined by the benchmarks. Measurements taken from this module are the execution time it takes to generate the context lattice, and the file size of the created file representing the concept.

The module ProtoCalculator (Algorithm A.17) includes the function calcProto to measure execution time for prototype calculation. The function reads the concept file generated earlier. The varying input parameter chooses a level in the context lattice, from which a context is selected and mapped to a concept to calculate different prototypes according to input parameters. The variations in level influence the time it takes to calculate a prototype. It is assumed that in lower levels, *e.g.* atomic contexts, less observations are considered for prototype calculation compared to higher levels in the contexts, which influences the time. The calculation time for prototypes is measured in this module.

The module Main (Algorithm A.15) creates benchmarks for the functions createAndStoreConcept and calcProto. The benchmark for the generation of the concept is given in Code Fragment 7.1, the other in Algorithm A.15.

The execution and measurements of the benchmarks is achieved by the function bench, which is equipped with a label. For the benchmark shown in Code Fragment 7.1 the input parameters are given by the expressions maxNumberContext and exemplars. These specify the maximal context number, and the number of exemplars assigned to each context. The benchmark included in Code Fragment 7.1 starts by executing the benchmarks with context lattices having one atomic context representation, and continues up to a context lattice established by 12 atomic context lattices. Each contextualized concept so created inherits one exemplar as the expression exemplars defines it. All these benchmarks are grouped together into one benchmark group, and build a full benchmark for the function createAndStoreConcept. In the same manner benchmarks are generated for the function calcProto given in Algorithm A.15.

Code Fragment 7.1: Implementation of a benchmark group

```
generationBench = defaultMain
 [bgroup "Concept generation"$
  map (\c -> bench (label c)
             $ nfIO (createAndStoreConcept c exemplars)) [1.. maxNumberContext]
 ]
  where maxNumberContext = 12
        exemplars = 1
        label c =  ("contexts-"++(show c)++"-exemplars-"++(show exe))
```

The created benchmark groups are executed via the main function. The modules are compiled into an executable file which is executed via a command line. The measurement values are returned on the standard output and a summary HTML file is generated from the criterion library. The measurement values generated in this way are used in the following analysis.

7.4 Measurements and Analysis

Three characteristics of the implementation are measured: (i) execution time for the generation of the context lattice, (ii) file size of the concept and the context lattice, (iii) execution time for the calculation of a prototype. The resulting measurement values are analyzed to identify the complexity class of the implementation.

Three benchmark groups are executed, alternating the number of exemplars included in the concept. All these groups are executed with different sizes of context lattices, ranging from one to twelve atomic context representations. The first benchmark group is established with one exemplar and is intended to show the lower bound for the benchmark. The second group is established with ten exemplars per context, where each context is equipped with the same ten exemplars. The last group uses twenty exemplars and is used to observe a trend for the analysis.

The benchmarks are executed on a notebook with the properties given in Table 7.2. It runs a Linux Mint [4] operating system, based on Ubuntu Trusty (14.04.4) executing benchmarks and compiling the implementation with the Glasgow Haskell compiler (GHC) [5] release 7.10.2.

Table 7.2: Details of the system execution benchmarks

Test system	
Brand	Lenovo Thinkpad X220
Processor	Intel Core i5 (2nd Gen) 2520M / 2.5 GHz
Ram	4GB DDR3 SDRAM
Operating System	Linux Mint 17.2 Rafaela
Kernel Version	3.16.0-38-generic
GHC	7.10.2

Execution time for the context lattice generation

The execution time for the generation of the context lattice and the concept are measured for several context lattice sizes and different numbers of exemplars. In detail, the execution time of the function `createAndStoreConcept` is measured for: one to twelve atomic context lattices each with either one, ten, or twenty equipped exemplars. The mean measurement values are given in Figure 7.2. The x-axis of this figure shows the number of atomic context representation, and the y-axis shows the logarithm of mean measurement time to generate all contexts. The red line is the maximal increase function given in Equation 7.1. The measurement values of the three different benchmark

[4] https://linuxmint.com

[5] https://www.haskell.org/ghc/

groups (one exemplar, ten exemplars, twenty exemplars) converge with eight and more atomic context representations. The measurement values for the creation time including one exemplar are given in Table 7.3.

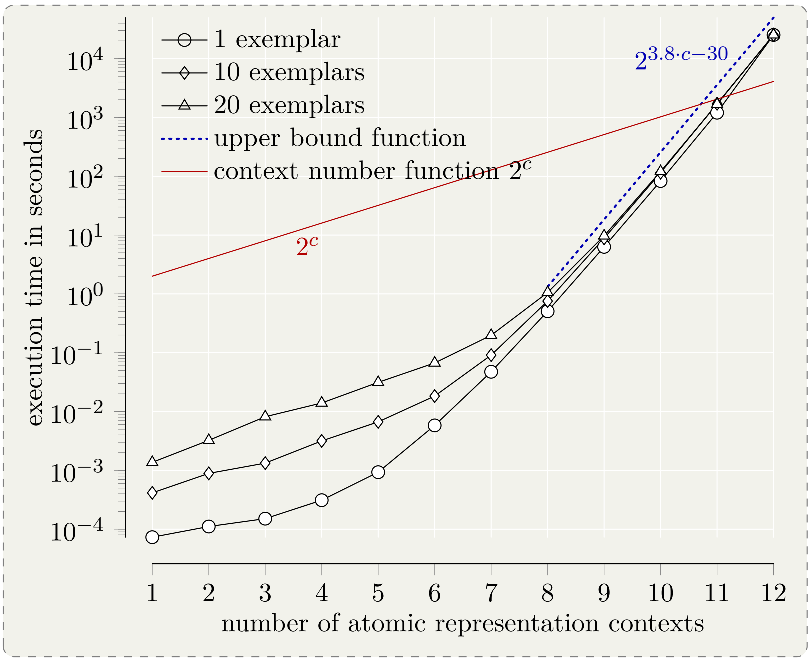


Figure 7.2: Measurements for creation time of different context lattices

atomic contexts	mean execution time in seconds
1	$70 \cdot 10^{-6}$
2	$110 \cdot 10^{-6}$
3	$150 \cdot 10^{-6}$
4	$300 \cdot 10^{-6}$
5	$900 \cdot 10^{-6}$
6	$5.8 \cdot 10^{-3}$
7	$47 \cdot 10^{-3}$
8	$500 \cdot 10^{-3}$
9	6
10	80
11	120
12	$25 \cdot 10^3$

Table 7.3: Measurements for creation time of a concept including one exemplar per context

One upper bound function for all measurements is included in dotted blue in Figure 7.2. This function is given in Equation 7.2, in the range from eight to twelve atomic context representations. The maximal function has the most impact on execution time. For example, the execution time for three contexts in mean is about 150 millionths of a second, whereas the execution time for twelve atomic contexts is about 25000 seconds (almost seven hours). The influence of the different numbers of exemplars is practically negligible for overall execution time.

$$2^{3.8 \cdot c - 30} \tag{7.2}$$

In conclusion, the execution time of a concept including the context lattice follows an exponential function which falls into the complexity class **EXP** in terms of the number of atomic context representations.

File size of a generated concept

The file size of an exported concept is measured in order to draw conclusions about the file sizes of the data structures. The exported file includes annotations necessary to build the data structure during execution time. From the file sizes, the influence of contexts, exemplars, and annotations is obtained. To reduce the influence of context names and exemplar names in the files size, they are generated in a specific pattern. The context pattern starts with a small letter `c`, followed by an underscore, and one or two randomly generated characters *e. g.*`c_ab`. The pattern for exemplars is similar, only starting with an `e` to produce identifiers such as `e_db`. This pattern guarantees that the names have a constant influence on the file size.

The file sizes of the generated context lattices for several numbers of exemplars are included in Figure 7.3. The context lattices mapped to one exemplar present a minimal file size for a concept. The measurement values for the benchmark with ten exemplars shows the same inclination, only shifted upwards. The measurement values for benchmarks with twenty exemplars also shares the inclination, and a shift upwards with smaller differences.

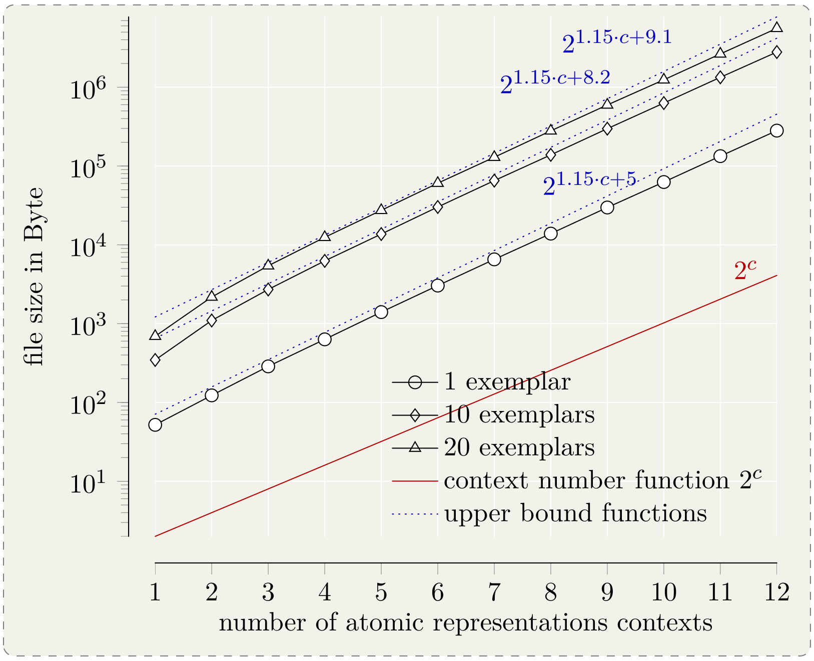


Figure 7.3: Measurements for file size of different context lattices

Possible upper bound functions are based on the maximal exponential function. Sample upper bound functions are given in dotted blue lines in Figure 7.3. The upper bound function for the context lattice mapped to one exemplar is graphically found as $2^{1.15 \cdot c+5} = 2^{1.15 \cdot c} \cdot 2^5$. Compared to the maximal function, it is increased by factor 1.15, and a constant upward shift of factor 2^5. This increase depends on the context and exemplars, and the annotation needed to describe the data structure in the exported file. File sizes for concepts established with ten and twenty exemplars show a similar pattern, having a different upward shift. These lattices mapped to 10 exemplars have the constant $2^{8.2}$. This shows that the number of exemplars has an influence on the upward shift. For twenty exemplars, a multiplier of $2^{9.1}$ is identified. Taking into account this dependency, the constant is split into a constant value a and the number of exemplars e, resulting in $a \cdot e$. Within this abstraction, a general file size function is found, as given in Equation 7.3.

$$2^{1.15 \cdot c+a \cdot e} \tag{7.3}$$

In summary, the file size of the concept depends on two factors: the number of atomic contexts and the number of exemplars mapped to the contexts. The influence of the number of exemplars is constant, and in summary dominated by the exponential function.

The general file size function also defines the complexity class. The complexity class is given by the term of the function with the biggest number of increase, determined by the exponential function $2^{1.15 \cdot c}$. In conclusion, the file size of concepts is of the **EXP** complexity class.

Execution time of prototype calculation

Execution time is measured for the calculation of prototypes from previously generated concepts. The calculation of prototypes is done for contextualized concepts that include different numbers of observations. To identify a trend, contextualized concepts from all context lattice levels are generated to measure the difference for the included number of observations. These measurements are executed for the three benchmarks including one, ten, and twenty exemplars. From the measurements, upper bound functions are depicted that lead to a conclusion about the complexity class for the execution time of prototypes.

The number of context combinations that can be achieved varies with the number of atomic context representations, which influences execution time. For example, a context lattice built with three atomic context representations includes four levels, as shown in Figure 7.4. As the $\perp$ level does not include any observation, the first contextualized concept is created by level one, *i.e.* atomic context representations. From the atomic representation level, two other levels can be reached in the context lattice from Figure 7.4. The contexts in different context levels are connected with different numbers of observations that have to be taken into account to calculate the prototype. The atomic context representations include the least number of observations. With every disjunction of atomic contexts, the number of observations increases until the $\top$ context is reached, which includes all observations and the calculation time is at its theoretically highest.

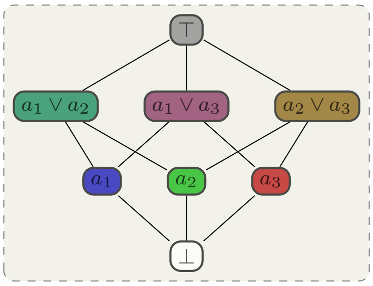


Figure 7.4: Context lattice created from three atomic context representations

The mean measurement values are shown in three figures distinguished by the number of exemplars. Measured execution time values for all possible contextualized concepts including twenty exemplars are given Figure 7.5. Measurement values including one and ten exemplars are given in Figure A.2 and Figure A.3, respectively, in Appendix A, as all share the same pattern. The axis labeled “number of atomic context rep.” lists the number of atomic context representations that are used to build the context lattice. The axis labeled “context combinations” represents on which level the context is taken to create the contextual concept. The axis labeled “mean execution time in s” shows the mean execution time to calculate the prototype. The measurement values confirm the argument that with increasing level of the depicted context the time to calculate the prototype also increases.

The upper bound functions are found and compared to obtain a general upper bound function. The prototype calculation times for context lattices built from twelve atomic context representations mapping to contextualized concepts including one, ten, and twenty exemplars are

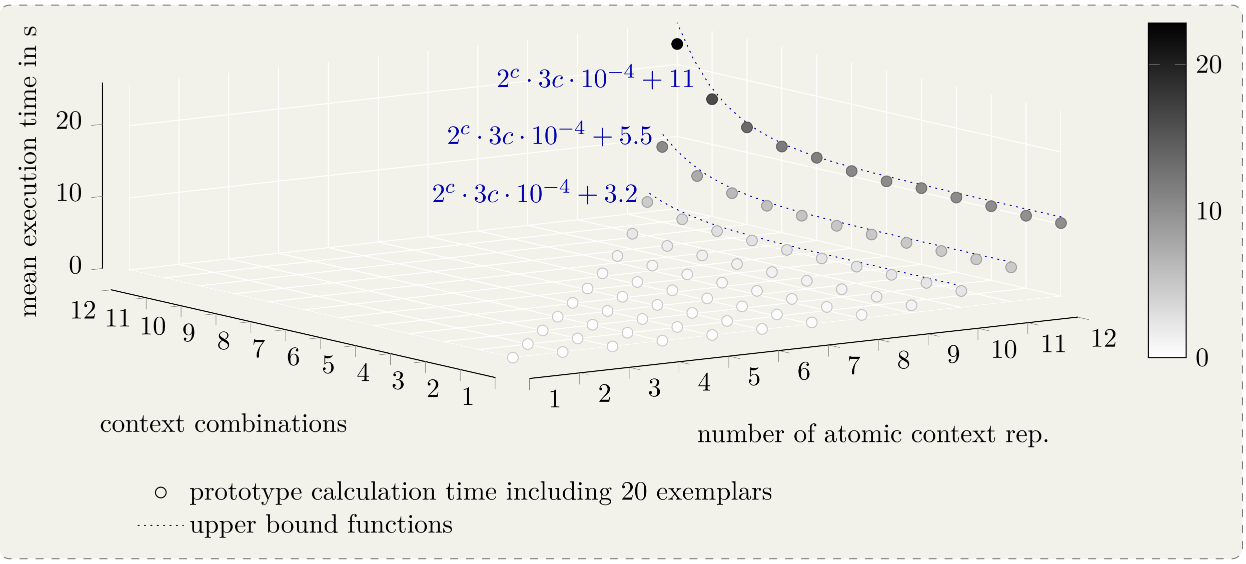


Figure 7.5: Measurements for prototype calculation time including 20 exemplars

given in Figure 7.6. All obtained upper bound functions share a pattern given in Equation 7.4. The variable a depends on the number of exemplars, and the variable b on the number of atomic contexts as well as on the number of exemplars.

$$2^c \cdot a \cdot c \cdot 10^{-6} + b \tag{7.4}$$

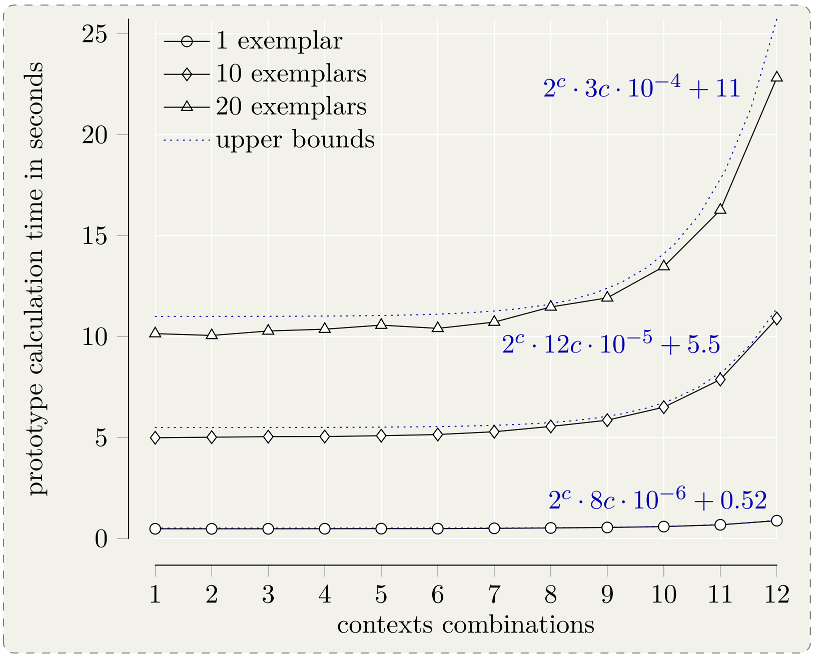


Figure 7.6: Measurements for prototype calculation time for different exemplar numbers

To confirm the pattern, the upper bound functions for context lattices built by ten, eleven, and twelve atomic context representation are given in Table 7.4 and are obtained from Figure 7.5, Figure A.3 and Figure A.2, respectively.

atomic contexts	upper bound functions		
	1 exemplar	10 exemplars	20 exemplars
10	$2^c \cdot 8c \cdot 10^{-6} + 0.12$	$2^c \cdot 120c \cdot 10^{-6} + 1.2$	$2^c \cdot 300c \cdot 10^{-6} + 3.2$
11	$2^c \cdot 8c \cdot 10^{-6} + 0.25$	$2^c \cdot 120c \cdot 10^{-6} + 2.5$	$2^c \cdot 300c \cdot 10^{-6} + 5.5$
12	$2^c \cdot 8c \cdot 10^{-6} + 0.52$	$2^c \cdot 120c \cdot 10^{-6} + 5.5$	$2^c \cdot 300c \cdot 10^{-6} + 11$

Table 7.4: Upper bound functions for prototype calculation time

Function 7.4 depicts the complexity class for the prototype calculation time. The most increasing factor is the exponential function 2^c, which results in the function $O(2^c)$ resulting in complexity class **EXP** for the time it takes to calculate the prototype.

7.5 *Discussion*

The measurements confirm an exponential complexity of the implementation driven by the maximal function 7.1. The similarity of the measurements and the maximal function provide evidence that the implementation does not include any bottleneck. The factors increasing the maximal function are created *e.g.* by data structures. The implementation can be improved, but large gains will be unlikely because of the influence of the maximal function given in Equation 7.1.

In respect to practical large-scale implementations methods to circumvent or reduce the execution of the implementation can be found given certain factors:

- Assuming the number of contexts and observations is fixed: the context lattice and prototypes can be preprocessed, storing results in a lookup table. The context lattice is fully populated, and for each context a prototype is obtained. The result of the preprocessing establishes a lookup table including the context and the prototype *e. g.* *in Vienna* ↦ Plachutta, *serving pizza* ↦ Pizza Hut etc. The amount of time needed to calculate the lookup table can be disregarded, because it must be done only once in advance. Querying the lookup table will circumvent the execution of the implementation.

- Assuming the number of contexts is fixed and the observations change: this affects only the prototype calculation, the context lattice is not affected. By adding or removing observations, the prototype may change and must be recalculated. Depending on the number of changes to the observations they can be computed in real time or be stored together with the context in a lookup table. One reduction in this method can be computing only those prototypes that are affected by the change. In this method the context lattice is processed once and then stored, while the prototype calculation is redone with every change in observations.

- Assuming the number of contexts is flexible: this affects every part in the implementation, and for each change both the context lattice and the prototypes have to be recomputed. If contexts change constantly, no preprocessing is possible.

7.6 Conclusion

The implementation is analyzed to answer the question of the computational complexity regarding file size and execution time. A major influence on the complexity comes from the number of atomic context representations. The influence is modeled by the maximal function 2^c, where c is the number of atomic context representations.

Conclusions from the memory and execution time measurements result in the **EXP** (exponential) complexity class for the implementation. The complexity classes were determined from measurements of the algorithms: concept generation time **EXP**, file size of a concept **EXP**, and prototype calculation time **EXP**. From the maximal function and the measurements, it follows that the implementation does not include a bottleneck.

For practical large-scale implementations, methods are discussed to circumvent execution and the associated exponential amounts of time. Assuming either the number of contexts or the number of observations are fixed, a preprocessing step generates results stored in a lookup table to circumvent the execution.

8
Context Algebra applied to represent "near"

In this chapter, the context algebra is initialized with data to represent the concept **near**. The contributions in this chapter are:

- requirements for data in order to initialize the context algebra
- a review of existing data acquisition approaches with an evaluation of whether they are able to acquire all needed data– which concludes that existing approaches fall short
- a manual data acquisition approach that satisfies all requirements to initialize the representation of **near**
- a data analysis and the initialization of the model
- envisioned applications for the initialized representation of **near** with natural language input and natural language output

Attempting to represent the concept **near** has two motivations: (i) **near** is a universal semantic primitive, and (ii) existing models observing context influence can be integrated by this model. Wierzbicka [1996] classifies **near** as a universal semantic primitive that is understood but cannot be defined in terms of rules or other concepts. Lakoff and Johnson [2008][page 56f] include **near** on a list of "prime candidate[s] . . . that are understood directly without metaphor . . . emerging from spatial experience". **Near** is a polysemy, used to express multiple meanings, *e.g.* near in time, near in space, or a near relationship. The spatial meaning is further investigated to translate the word near into metric distances. As former representations have concluded, such a transformation is not trivial because of the context-dependence of **near**. For example, **near** can refer to a wide range of distances – to thousands of kilometers for "the moon is near the earth", but also to a mere meter or less, as in "the chalk is near the blackboard". By establishing a representation for **near**, it is expected that references will be distinguished and different metric distances determined according to the respective context.

8.1 Meanings for "near"

To focus on the spatial meaning of **near**, all other possible meanings that **near** is interpreted to have to first be identified. The electronic lexical database *WordNet* includes several meanings for **near**, described as *synsets*. A synset is "a set of word forms that are interchangeable in some context . . . without changing the way the sentence . . ." is interpreted [Fellbaum, 1998, page 24]. "It is convenient to think of a synset as representing a lexicalized concept of English" [Fellbaum, 1998, page 24]. Synsets are accompanied by explanations similar to those included in dictionaries, which makes the meaning clearer. The synsets including descriptions for **near** are queried online with `http://wordnetweb.princeton.edu/perl/webwn?s=near`, and show five usages as an adjective, two as an adverb, and one as a verb. In Table 8.1 several synsets including their meaning are given, which have to be distinguished in order to focus on the spatial meaning of **near**.

Table 8.1: Sample of synsets for **near**

synset	explanation
time	"in the near future"
space	**"stood near the door"**
degree	"they are near equals"
circumstances	"she was near tears"
relationship	"her mother is always near"
spending with reluctance	"very close (or near) with his money"
very close in resemblance	"a dress of near satin"

8.2 *Context influence on former models for "near"*

Many studies and formalization approaches for **near** exist that highlight the influence of context.

Fisher and Orf [1991] conducted a study where participants were asked if a central building on the university campus is near or close. They identified extreme cases, where either all buildings are near or close, or "a core grouped to be not-near". Robinson [2000] conducted a study to obtain a fuzzy value for linguistic terms that are used for geographic query languages. The experiment was conducted with two different spatial contexts, where results showed that even within a small sample there exists significant semantic variation. Neighborhood relations like "Is this far?" rely on surrounding objects. Evidence is shown by studies conducted by Worboys [2001], Duckham and Worboys [2001], Worboys et al. [2004] asking if places are "near" to a reference place. The results showed that context and scale factors affect the judgment of nearness. For an extensive review and a discussion of the role of context for proximity terms such as near, refer to Burigo and Coventry [2010]. In summary, it shows that formalizations for neighborhood relations have to take context into account.

Danofsky [1976] presented a model to understand the concept **near** using weighting factors like purpose, dimensions of objects, absolute distance, etc. Danofsky established several thresholds for concrete and discrete examples of **near** that are context-dependent. Robinson [1990] presented an algorithm for data acquisition to model **near** with

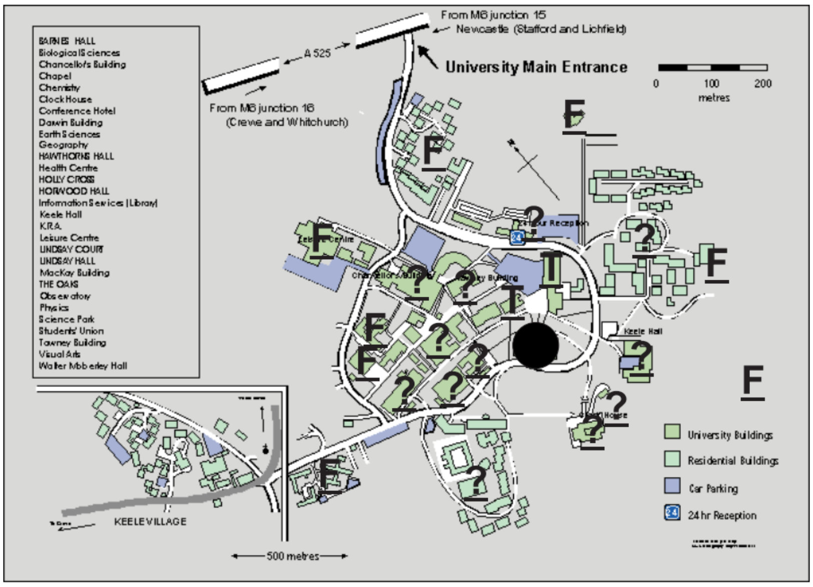


Figure 8.1: Buildings considered **near** to the library by Worboys [2001, Fig. 2]

a fuzzy number. Wang [1994] also used fuzzy numbers to model vague natural language. Wang concluded that one fuzzy distribution is not enough to model one word, and that multiple fuzzy representations are needed that change depending on the influencing context. Another approach was introduced by Frank [1992] using an algebraic approach to model proportions of distances, considering very near, near, far, and very far. Gahegan [1995] presented a formalization that includes influence of context on the fuzzy membership function. In a two-stage process first the context is taken into account, and then the fuzzy membership functions are applied. Duckham and Worboys [2001] established a three-valued (T, F, ?) nearness relation based on the survey results included in the map shown in Figure 8.1. Yao and Thill [2007] claimed that the fuzzy approaches are not accurate, and proposed a neurofuzzy system to model context-contingent distance terms. They used sample data to train a "neural network to figure out the membership functions", which shows higher prediction accuracy compared to previous fuzzy membership functions, but they conclude that the existing neural network libraries "[are] not well suited for the prediction of metric distances from a linguistic distance measure". From this review, the need to include context in a representation of **near** is apparent.

8.3 Data acquisition for "near"

This section explains the process of data acquisition for the concept **near**. Data that initialize the model have to be grounded and have to come from reality. Possible data acquisition approaches are reviewed, and an appropriate approach is proposed to acquire data. It is implemented and described in this section.

Data requirements to initialize the representation

To illustrates the requirements for data, the constituting rule from Searle is consulted. The distinction in the elements X, Y, and C is used to specify requirements for each element.

1. *X* – this element contains linguistic expressions for different meanings of **near**. For example, a temporal meaning as in "near lunchtime", a spatial meaning as in "near my home", or a relationship meaning "near to my heart".

2. *C* – this element contains linguistic expressions surrounding X: A perfect example for a context is: "walking to a near pub", where near is stated in conjunction with the context *walking*. These linguistic expressions are used to identify contexts that influence the concept. In summary, the data have to explicitly include context.

3. *Y* – this element contains objects observable in reality that are referred to with linguistic expressions. For example, the linguistic expression: "walking from central station to nearby St. Pauls Cathedral" is processed to extract a metric distance, in this case 500 meters. To derive a metric distance, the position of the *Locatum*, here central station, and the *Relatum* here St. Pauls Cathedral, have to be found in a gazetteer as illustrated in Figure 8.2. The relation from the linguistic expressions to positions on the earth also grounds the model in reality, and so avoids the symbol grounding problem (Section 2.1). The focus here is on the spatial meaning of "near" relating to distances observable in reality; similar approaches could be used for other meanings.

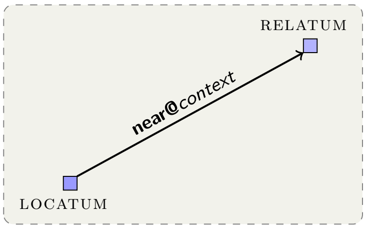


Figure 8.2: Relatum and Locatum representing **near**.

Data have to satisfy the three requirements above in order to initialize the context algebra used to represent **near**.

Data acquisition approach

Data acquisition approaches have to acquire data that satisfy the requirements stated in Section 8.3. In the following, four data acquisition methods are reviewed, and possible data sets are discussed regarding their suitability for satisfying the above requirements.

- *Sensors* that measure data can be one possible data source. The difficulty with this approach is that sensor measurements have to be categorized, where " . . . data may require different classifications (even at the same level of granularity) when used within different contexts" [Freksa and Barkowsky, 1996, page 117]. As the requirement is to extract context from data, this is not an appropriate approach to find different values in different contexts.

- *Questionnaires* produce data that can be used to initialize the context algebra. For example, Aerts and Gabora [2005a] used an email-delivered questionnaire where subjects rated the typicality of exemplars influenced by context on a seven-point Likert [1932] scale.

This approach is *reasonable* and produces useful data for calculations, but it is very time-consuming.

- *N-Grams* list occurrences of words stated in a sequence. The benefit of using natural language is that objects in reality are already recognized by an agent. This approach looks for patterns where words appear in a sequence, and counts their occurrences in huge text collections. In terms of a data acquisition approach, n-grams can be used to identify which surrounding words (context) influence a specific word. The number of surrounding words is limited, as search engine companies like Google[1] extract 2-grams ($n = 2$, two words build a sequence) and tri-grams ($n = 3$, three words build a sequence) from digitized books [Lin et al., 2012]. Derungs and Purves [2014] used tri-grams in the form of "A near B", where A and B are cities, to model **near**. The results showed that the approach falls short because additional context, *e. g.* the population of the cities, is not included in the tri-grams but affects the distance **near** refers to. Based on our requirements, the main criticism is that this method cannot determine the influencing context.

- *Natural language processing* (NLP) tools generate parsing trees and identify word types from a sentence. It could be used to determine surrounding words to extract context influence. A workflow using parsing trees was introduced by Hobel et al. [2016] and Gelernter and Balaji [2013], using machine-learning techniques to extracted geographic features from texts describing a place. What is missing, however, is an identification mechanism that can distinguish between words that act as context, and those that do not. Another problem of natural language processing is identifying the relation to reality, *e. g.* the metric distance that **near** refers to. One approach introduced by Zhang et al. [2010], Wallgrün et al. [2014], Xu et al. [2014] is to identify locatum and relatum in the text using simple patterns, and then use a gazetteer to identify their locations in reality. They concluded that in order to apply their approach to large data sets "... analysis will require advanced language processing, interpretation, and geoparsing techniques" [Wallgrün et al., 2014]. Regarding our three requirements, NLP is indeed the most promising method of acquiring the necessary data, but it does not yet satisfy all of the requirements stated above.

Neither of these approaches satisfies all requirements, nor are they easily usable for large data sets. In order to acquire all necessary data to initialize the representation for **near**, a new approach has to be devised.

Data sets including the concept **near**, the influencing context, as well as relatum and locatum are necessary to gather all data required to initialize the representation for **near**. The web is one popular source to look for data sets. For example, Wallgrün et al. [2014] used portals that rate and describe holiday trips to build a corpus for spatial relations. Like Wallgrün et al. [2014], a data set available on the web

[1] `https://books.google.com/ngrams/`

that includes real estate advertisements is chosen because it includes all the necessary data. Real estate advertisements include the relatum as the property, and the locatum as another point of interest described in relation to the relatum: *e.g.* "The flat (Gusshausstrasse 25-27, 1040 Vienna) is in near walking distance to Karlsplatz". The context that influences **near** can be determined from the text of the advertisement, *e.g.* in our example **Near** is mentioned in the context ***walking***. As they include all necessary data, real estate advertisements are chosen to initialize a representation of **near**.

Two real estate data sets available on the web are used to initialize **near**. Both data sets mainly include properties in Austria that are advertised in the German language. The first data set is obtained from the portal available at https://www.willhaben.at/iad/immobilien, referred to here as *willhaben*. The second data set is obtained from derStandard, which is available online at http://derstandard.at/Immobilien and referred to here as *derStandard*.

Data acquisition process

The data acquisition approach is inspired by natural language processing tools. It is a manual process establishing a workflow supported by a web application. To acquire the necessary data from the data sets, the real estate entries are imported into the data base of the web application.

The willhaben data are retrieved by a web crawler, and stored as comma-separated values (CSV)[2] file. The data set includes entries ranging from March 2013 to August 2014, and includes textual description fields for the address as well as explanation fields for the real estate entry. An example entry for one flat in CSV format is given in Table 8.2. Not all fields are filled with text, which leads to many missing addresses in the data set. The entries without an address are not included in the further acquisition process.

ID	address	...	description
1	Gusshausstrasse 25-27, 1040 Wien	...	Dies helle Objekt ...

Table 8.2: Example entry included in "willhaben" data set

The derStandard data set is provided by the company[3] der Standard[4] itself, in an eXtended Markup Language (XML) format and in a real-estate-specific schema (for more details see Appendix B). The entries are collected from a period from March 2007 to February 2016 and include several fields specifying the address and the description of the real estate object in natural language. An example entry for one flat, including only tags used in the processing step, is given in XML format in Algorithm 8.1. The address and the description are mandatory, which yielded a large number of data that can be used for further processing.

Database import In the data importing step, both data sets are parsed and imported in a spatial database system. The parsing and commu-

[2] special thanks to Gerhard Muggenhuber for providing the data in a personal communication

[3] special thanks to Michela Bocchi and the editorial office of derStandard Immo company for providing the data in a personal communication

[4] http://derstandard.at/

Code Fragment 8.1: Example entry included in "derStandard" data set

```
<immobilie>
  <geo>
     <plz>8010</plz>
     <ort>Graz</ort>
     <strasse>Hugo Wolf Gasse 8</strasse>
     <land iso_land="AUT" />
     <bundesland>Steiermark</bundesland>
  </geo>
  <freitexte>
     <objekttitel>PROVISIONSFREIE GARCONNIERE NAECHST UNIVERSITAET GRAZ ZU VERMIETEN
     </objekttitel>
     <objektbeschreibung> Neubau, ruhige Lage, ...
     </objektbeschreibung>
      ...
  </freitexte>
   ...
</immobilie>
```

nication with the database are achieved by Haskell programs listed in Appendix Section B.2 for the willhaben data set, and in Section B.2 for the derStandard data set. As database system (short database), the open-source object-relational database system postgresql[5] enriched with spatial algorithms[6] is used. The reason for selecting a spatial database system is to have spatial algorithms available to calculate the distance between relatum and locatum in further steps. In order to import the data in the database, a relational schema is created which is included in Figure B.1 in Appendix B. In total, 43438 entries are imported into the database, with 978 entries from willhaben and 42460 entries from derStandard, as listed in Table 8.3.

data set	entries	entries including near
derStandard	42460	9424
willhaben	978	153
$\sum$	43438	9577

Table 8.3: Total number of real estate entries

To model **near**, only entries mentioning "nahe" (adjective) or "Nähe" (noun) in the description or title are considered. A filter in the data base is executed to process only entries mentioning near. This filtering process reduced the number of entries to 9577 (distribution is included in Table 8.3).

Web Application to guide the workflow To process the real estate entries, a web application was devised that creates a workflow for extracting the required data. The architecture of the web application is split into a back-end and a front-end part, both shown in Figure 8.3. The back-end (database and `scotty`[7] web Server) is executed as a server daemon on a computer located at the Research Group Geoinformation. The back-end handles the communication with the database via several modules (Algorithms B.13 and B.14) to query and update real estate entries (Algorithm B.11) and to overview the workflow, and two modules that support further functionality in Algorithm B.12 and B.15.

[5] `http://www.postgresql.org`

[6] `http://postgis.net`

[7] `http://hackage.haskell.org/package/scotty`

The retrieved data are transferred into Haskell types in the modules shown in Algorithm B.16 and Algorithm B.17.

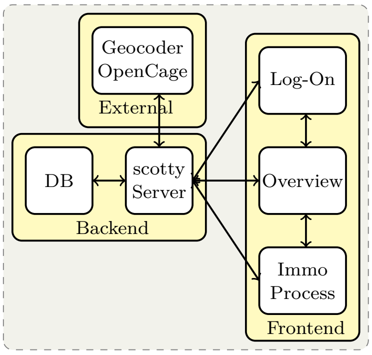


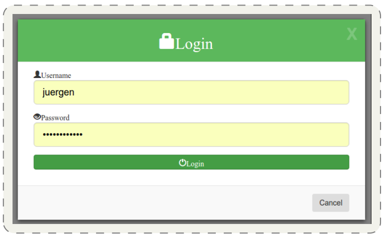
Figure 8.3: Web application architecture guiding the data acquisition process

The back-end queries an external gazetteer service (Algorithm B.18) available via http://api.opencagedata.com/geocode/ to get locations for addresses, which is used because a Haskell library exists. The front-end is implemented via several HyperText Markup Language (HTML[8]) interfaces: (Log-On Algorithm B.19, Overview Algorithms B.20, B.21 and Immo Process Algorithms B.22, B.23), which makes it accessible from various platforms (mobile, tablet, PC) and allows delivery via the scotty web server to the client.

The scotty server establishes a connection to the database and to an external gazetteer, and includes the functionality to deliver the front-end. Scotty manages functions written in Haskell [Marlow, 2010] to create web interfaces, and controls the HTTP[9] handling. Functionality to store data acquired from the user are provided via Representational State Transfer (REST) [Fielding and Taylor, 2000] architecture style services.

The front-end interfaces are based on HTML and JavaScript libraries and uses Asynchronous JavaScript + XML (Ajax) requests for retrieving additional data and sending data to the scotty server. The HTML interface uses responsive libraries to adjust the interface to mobile and non-mobile web browsers. The main interface, called ImmoProcess, includes a map that is established by OpenLayers3[10] using functionality written in JavaScript shown in Algorithm B.25, B.26, B.27, B.28, and B.29. The data acquired through this interface are sent via Ajax to the main module (given in Algorithms B.8, B.9, and B.10) and executed on the server, which fuses the functionality of all modules to store the data in the database, *e.g.* the contexts influencing **near**.

The web application provides interfaces for multiple parallel users that have to identify themselves before beginning the workflow on the Log-On interface. If an agent enters credentials that are known to the system (user and password are stored in the database) into the Log-On interface shown in Figure 8.4, a session is established and the Overview interface is shown. The authentication mechanism (Algorithm B.12 and B.8) enables the identification of user-processed real estate entries, and denies access to unknowns users. The active session is necessary in order to receive further interfaces. REST services also respond only if a session is active. A session is active for three hours, and is expanded with every completed workflow described below.


Login

Username
juergen

Password
••••••••••••

Login

Cancel

Figure 8.4: Log-On interface

The workflow includes six steps and is established by the interfaces Overview and ImmoProcess. The work flow starts with the Overview interface that is shown after a successful authentication and includes the first step of the workflow. The other five steps are processed in the ImmoProcess interface, and when completed the user is redirected to the Overview interface to start the workflow again. The six steps are as follows:

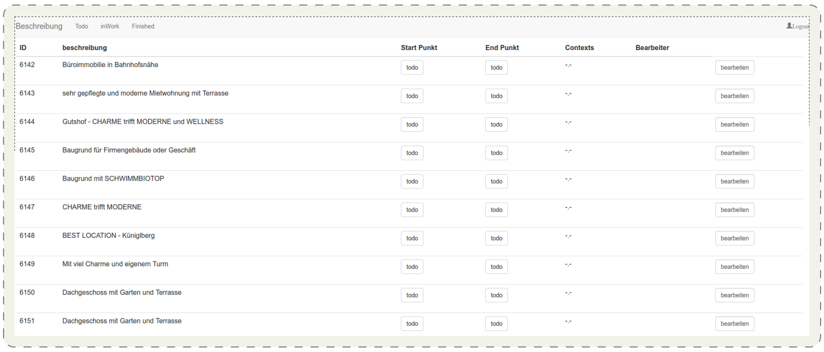
1. *Selecting a real estate entry* The user selects one of the real estate

[8] https://www.w3.org/TR/html5/

[9] https://www.w3.org/Protocols/rfc2616/rfc2616.html

[10] http://openlayers.org

entries included in the `Overview` interface (in Figure 8.5) by clicking the button labeled "bearbeiten", which redirects the user to the `ImmoProcess` interface.



ID	beschreibung	Start Punkt	End Punkt	Contexts	Bearbeiter	
6142	Büroimmobilie in Bahnhofsnähe	todo	todo	-.-		bearbeiten
6143	sehr gepflegte und moderne Mietwohnung mit Terrasse	todo	todo	-.-		bearbeiten
6144	Gutshof - CHARME trifft MODERNE und WELLNESS	todo	todo	-.-		bearbeiten
6145	Baugrund für Firmengebäude oder Geschäft	todo	todo	-.-		bearbeiten
6146	Baugrund mit SCHWIMMBIOTOP	todo	todo	-.-		bearbeiten
6147	CHARME trifft MODERNE	todo	todo	-.-		bearbeiten
6148	BEST LOCATION - Küniglberg	todo	todo	-.-		bearbeiten
6149	Mit viel Charme und eigenem Turm	todo	todo	-.-		bearbeiten
6150	Dachgeschoss mit Garten und Terrasse	todo	todo	-.-		bearbeiten
6151	Dachgeschoss mit Garten und Terrasse	todo	todo	-.-		bearbeiten

Figure 8.5: Overview interface, showing real estate entities that have to be processed

2. *Identifying context* The user identifies context influencing **near** in the title and description of the real estate entry. The title and the description are presented on the top left part in the `ImmoProcess` interface (Figure 8.6), highlighting the words *nahe* and *nähe* with a green background to aid in the identification process. From words surrounding "near", those words that (in the user's understanding) influence **near** have to be selected, and then inserted into the input field marked as "context". In this example, the user selects the sentence "Das Wiener Stadtzentrum ist leicht und schnell zu erreichen und mit dem nahegelegenen Stadtpark findet ..." and identifies "gelegenen" as an influencing context. They then insert it into the input field (shown in Figure 8.6). By clicking the button labeled "add Context", the context is stored in the database and this step is completed.

3. *Locating the real estate entity* The user locates the real estate entry on the map from possible locations that are retrieved from a gazetteer. The address specified by the real estate agency for the real estate entry is prepopulated in the input field labeled "Startpunkt" in the `ImmoProcess` interface (Figure 8.6). In the example, for the real estate entry given in Figure 8.6 the prepopulated address is "1030, Wien, Seidlgasse". To translate this address into a location, the button labeled "verorten" is pressed which queries a gazetteer and shows the returned locations as markers A on the map. The response of the gazetteer will in most cases include multiple results[11] from which the user selects the appropriate one; this will change the marker to A. This step is completed by pressing the button labeled "speichern" to store the address and the location in the database.

4. *Locating the locatum* The user locates the locatum included in the text description in the map using suggestions from a gazetteer. The address of the locatum is identified in the same sentence where the

[11] The response of the gazetteer service depends on the query data; for incomplete input data the gazetteer service responds with all possible locations the address matches to, in the entire world. This behavior is similar to what was described in the introduction example, and will be further discussed in Section 9.2 where an integration of the context algebra for gazetteers is recommended

context was identified. For the example given in Figure 8.6, “Stadtpark” is identified as the locatum and entered into the input field labeled “Endpunkt” in the `ImmoProcess` interface (Figure 8.6). With the same mechanism as in the previous step, suggested locations are proposed by a gazetteer (designated with markers E) and the user selects the appropriate marker, which will then change its color to blue E. This step is again completed by pressing the button labeled “speichern” to store location and address in the database.

5. *Categorizing the real estate entity regarding spatial meaning* The user categorizes the real estate entry regarding space meaning. The category is assigned via clicking on either of three buttons included in the bottom left part in the `ImmoProcess` interface (Figure 8.6). Which button/category to select by the user is given in Table 8.4.

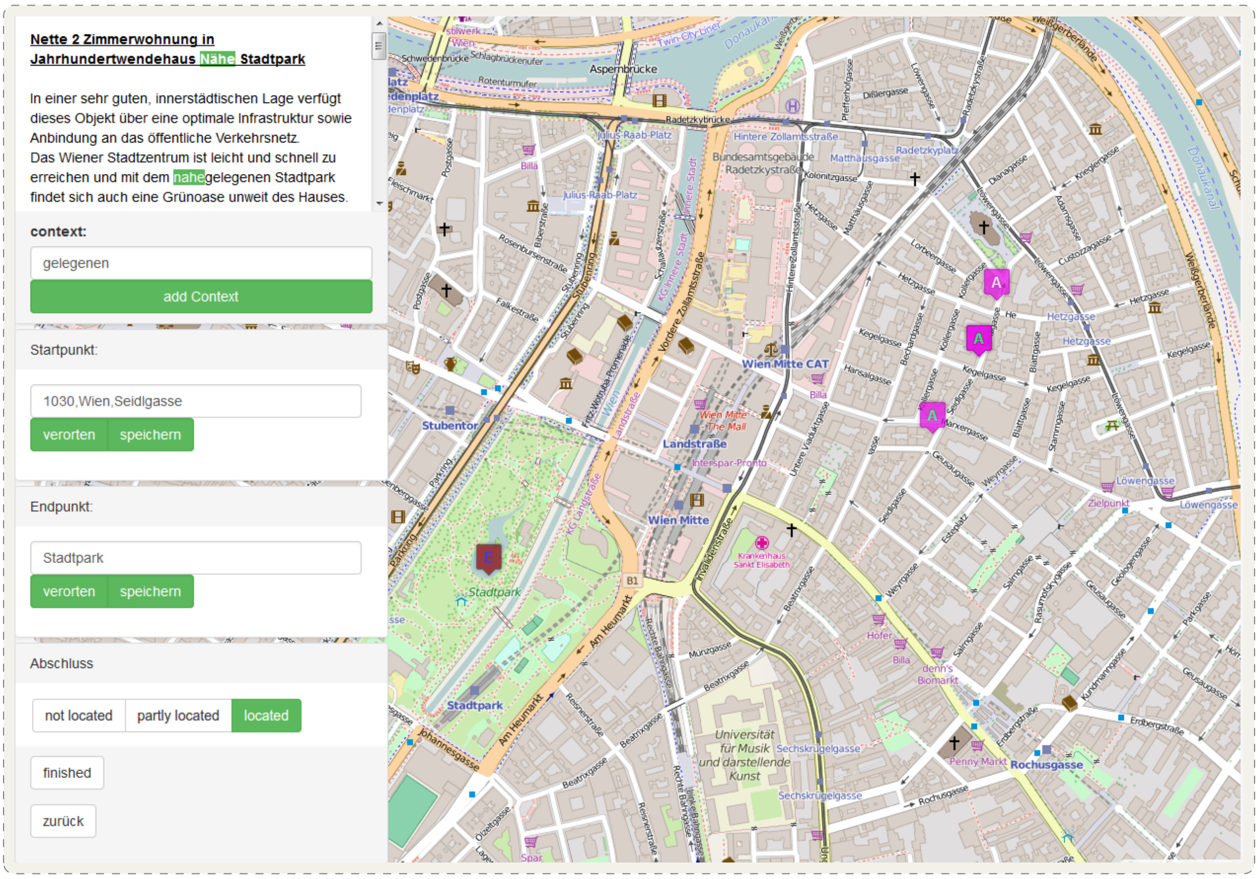
relatum location	locatum location	example	quality category
missing	missing	“. . . in näherer Zukunft” – “in near future”	not located
maybe identified	maybe identified	“flat near subway station Pilgramgasse”	partly located
identified	identified	“The flat Gusshausstrasse 25-27 is located near Karlsplatz”	located

Table 8.4: Categorization guide for spatial meaning of real estate entries

By pressing one of these buttons the category is stored in the database and this step is completed.

6. *Finishing the process for this real estate entity* The user completes or aborts the data acquisition for this real estate entry in this step. If all previous step were successfully completed, the interface looks as shown in the screenshot in Figure 8.6, and the user can complete the workflow by pressing the button labeled “finished”. If the user is not sure if there was a mistake somewhere or if one of the previous steps was not completed successfully, the user can press the “zurück” button, and the current entry can be processed again later by another user. After clicking one of the two buttons, the `Overview` interface reappears and the workflow can be started again.

The workflow was successfully completed for 6141 real estate entries by ten users. In group sessions, up to six users processed different entries simultaneously. Due to time constraints not all 9755 real estate objects were processed, but 6141 entries provide enough data to have meaningful results. The average duration to process one real estate entry is about six minutes, depending on the time spent locating the relatum and locatum. A major influence on the duration comes from the results delivered by the gazetteer service, which are sometime incorrect. In one case the query included the words “Vienna, Austria”, but the returned results were located in China. In such cases, a different gazetteer has to be queried which may yield better results.



Nette 2 Zimmerwohnung in Jahrhundertwendehaus Nähe Stadtpark

In einer sehr guten, innerstädtischen Lage verfügt dieses Objekt über eine optimale Infrastruktur sowie Anbindung an das öffentliche Verkehrsnetz.
Das Wiener Stadtzentrum ist leicht und schnell zu erreichen und mit dem nahegelegenen Stadtpark findet sich auch eine Grünoase unweit des Hauses.

context:

gelegenen

Startpunkt:

1030,Wien,Seidlgasse

Endpunkt:

Stadtpark

Abschluss

Figure 8.6: ImmoProcess interface indicating a completed workflow

Data analysis

Within the workflow described above, the real estate entries are processed and further analyzed. The analysis is separated in two parts. The first part analyzes the acquired data regarding contexts in order to generate contextualized concepts. The second part focuses on the distances between relatum and locatum, which are used as exemplars to initialize the representation of **near**.

Context identification According to the constituting rule of Searle, the contexts C influencing the interpretation of **near** are identified. In order to identify the influencing contexts, the stored contexts are lemmatized. In this preprocessing step all identified contexts are processed by a lemmatizer, which brings the word to its canonical form so that differences in tense or case are eliminated, *e. g.* "liegende, gelegene" is lemmatized to "liegen", "zu Fuß" (by foot) into "gehen" (walk), etc. This preprocessing step results in 111 different contexts given in Tables B.1, B.2, and B.3, which are then further analyzed.

The analysis distinguishes three types of contexts influence: *no context*, *one context*, and *combined context*. The type no context stands for no context influence on **near**, the type one context includes context in the form of one word, *e. g.* *walk (gehen)*. The type combined context stands for multiple contexts, *e. g.* *walk, immediate (gehen, unmittelbar)*. The majority of the processed entries, 62 % $(=\frac{2618+201+952}{6141})$, include no context, 37 % $(=\frac{978+213+1068}{6141})$ include one context and a minority of 1% $(=\frac{13+64}{6141})$ include more than one context. Their distribution within the quality classes is given in Figure 8.7. In total, 93 one-context type contexts are identified, as well as 18 combined contexts.

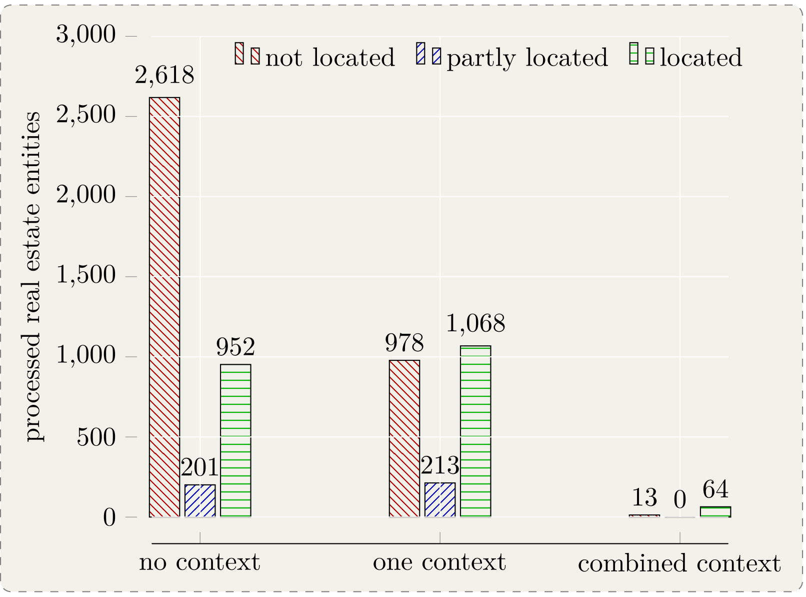


Figure 8.7: Contexts identified during data acquisition, grouped by categorization and number of context in combination

Three combined contexts are the highest number of contexts identified for **near**. These are: *located, airport, train (liegen, Flughafen, Bahn)* and *located, absolute, walk (gehen, liegen, absolut)*. The latter context, for example, was derived from the sentence: "Die naheliegende in absoluter Gehdistanz liegende Infrastruktur in der Krottenbachstrasse."

The contexts mentioned in the real estate entries for **near** established four different meanings. The spatial meaning was the main meaning that was identified. If no spatial meaning for **near** was included, other contexts indicating different meanings were identified and stored in the database. From the meanings listed by WordNet in Table 8.1 relationship, temporal, and spatial meanings are identified in the real estate data collection. Additionally, an *information* meaning for **near** is included which may be related to a German use of **near** that has no proper equivalent in English. Such language-dependent effects for **near** were already reported by van der Zee et al. [2009]. They compared English and Finnish usages of **near** and concluded that differences are related to, for example, functional relatedness. The contexts and their number of occurrences identifying the relationship, temporal, and information meanings are shown in Table 8.5.

Table 8.5: Meanings of **near** identified in the real estate data set

meaning and context	English translation	number of occurrences
relationship		
Naheverhältnis	*close relationship*	15
Angehörige	*close relatives*	3
time		
nahe Zukunft	*near future*	7
information		
Auskünfte	*information*	76
Details	*details*	8
Informationen	*information*	89
Interesse	*interest*	13

For the spatial meaning, 64 different located contexts are identified that are either one context type or combined contexts. A small sample including multiple combined contexts is given in Table 8.6, all others are included in Tables B.1, B.2 and B.3. The contexts included in Table 8.6 are translated into English and the number of occurrences in the data collection are included. The one-contexts *walk*; *located* and *immediate* occurred most often in the data collection, with 136 occurrences of (*walk*), 664 of (*located*), and 438 of (*immediate*). These are also part of many combined contexts, such as: *located, very*; *immediate, shore*; *immediate, in a direct line* etc. and occur in combination, *e. g.* *walk, located*; *immediate, walk*.

Exemplar identification According to the constituting rule of Searle, the possible interpretations (exemplars) Y of **near** are identified. As exemplars that **near** can refer to, the *direct line* between the relatum and locatum is determined. Also other determination methods for example the path on a road network can be used to extract exem-

context	English translation	number of occurrences		
		not located	partly located	located
gehen	*walk*	62	7	67
gehen, liegen	*walk, located*	0	0	4
liegen	*located*	131	66	447
liegen, Flughafen, Bahn	*located, airport, train*	0	0	1
liegen, absolut, gehen	*located, absolute, walk*	0	0	1
liegen, ganz	*located, totally*	0	0	1
liegen, sehr	*located, very*	6	0	0
unmittelbar	*immediate*	133	34	271
unmittelbar, Altstadt	*immediate, old town*	0	0	1
unmittelbar, Autobahn	*immediate, highway*	4	0	0
unmittelbar, Luftlinie	*immediate, in a direct line*	0	0	5
unmittelbar, See	*immediate, lake*	1	0	0
unmittelbar, Ufer	*immediate, shore*	0	0	28
unmittelbar, Zentrum	*immediate, center*	0	0	1
unmittelbar, gehen	*immediate, walk*	2	0	6
unmittelbar, liegen	*immediate, located*	0	0	2

Table 8.6: Sample of Contexts that influence **near**, included the number of occurrences

plars from relatum to locatum. For example, the shortest path or the fastest path are better exemplars for context *walking*, but which of these should be chosen? Does this method also make sense for context *located*? Which determination method to use is context dependent and the data set lacks of this context to decide which determination method to use and as common denominator the direct line is used here. Real estate entries with a non-spatial meaning do not have a location assigned to the relatum and the locatum, and therefore no distance/exemplar can be obtained for these meanings. Real estate entries with non-spatial meanings and entries that are classified as not located and partly located are discarded. For the 2084 as "located" classified real estate entries, the distance is determined as a direct line between relatum and locatum calculated in the database and rounded to integer precision. The minimal distance calculated is about five meters, while the maximal distance about 100 kilometers. The number of occurrences for distances in the range from zero to 1000 meters is given in Figure 8.8. The exemplars with a distance larger than 1000 meters occur between one and five times. In total, 1391 difference exemplars are identified that occur between one and eleven times, which are the possible interpretation exemplars for Y.

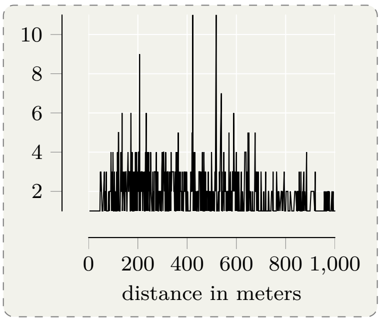


Figure 8.8: Exemplar occurrences in the range from 0 to 1000 meters

The locations for relatum and locatum are for example in Croatia, Poland, Spain, and the USA, but most locations are in Austria. Exemplars obtained from the Vienna area are designated by red lines in Figure 8.9.

Data acquisition and analysis resulted in 2084 real estate entries satisfying all requirements. They will be used to initialize the context

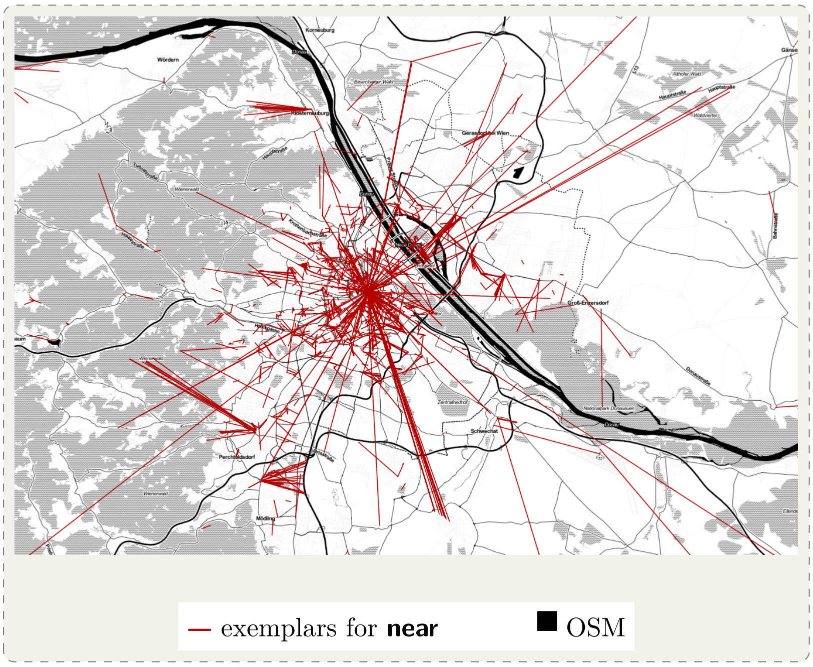


Figure 8.9: Exemplars for **near** in the Vienna area

algebra. From these entries, 72 different contexts and 1391 different exemplars are identified. With respect to the constitute rule of Searle, contexts C and exemplars Y are determined and are ready to initialize the **near** representation.

8.4 Context Algebra initialized with data

To create a representation for **near** using the context algebra, a context lattice and contextualized concepts have to be initialized with data. From the real estate data only the entries satisfying all requirements are applied here. The representation for **near** is split in two parts. In the first part, the identified contexts build the context lattice influencing **near**. In the second part, the exemplars initialize contextualized concepts.

Context lattice In order to build a context lattice, atomic representations have to be identified. The idea of atomic contexts is to build a complete lattice, where every element in the lattice can be mapped to a subset of exemplars. In this case, the identified exemplars do not map to all possible context combinations. Only for the contexts included in Tables B.1, B.2, B.3 are mappings to subsets of exemplars identified. That not all contexts that are included in a completed context lattice can be mapped to a subset of exemplars was also observed by Aerts and Gabora [2005a]. They assumed that atomic contexts may not occur in data, and that the contexts that occurred in the data (they used basic contexts) are sufficient to build the context lattice. Without atomic contexts not all possible context combinations are included in the context lattice only these that occurred in the data. This assumption is reasonable and is also applied here. With that assumption in mind, all the 72 contexts (57 are one-contexts, 15 are combined contexts) classified as located build the top part of the context lattice.

A part of the context lattice is shown in Figure 8.10 to indicate missing atomic contexts. Contexts that are written in black are mapped to subsets of exemplars. Contexts that are written in gray include no exemplars in the mapping. The points indicate that the context lattice also includes other elements.

The least and greatest context complete the context lattice. The $\bot$ context maps to the most selective context that is generated by conjunction of all contexts. This context maps to the empty set. The $\top$ context considers all contexts included in the lattice. This context maps to the whole set of exemplars.

The context lattice also includes contexts for non-spatial meanings. These contexts are *future (Zukunft)*, located on the left side of Figure 8.10, *close relationship (Naheverhältnis)*, not shown in Figure 8.10 indicated by dots ..., and context *information* (not shown in Figure 8.10), comprising two German contexts *Auskünfte* and *Informationen* that have only one English translation for both.

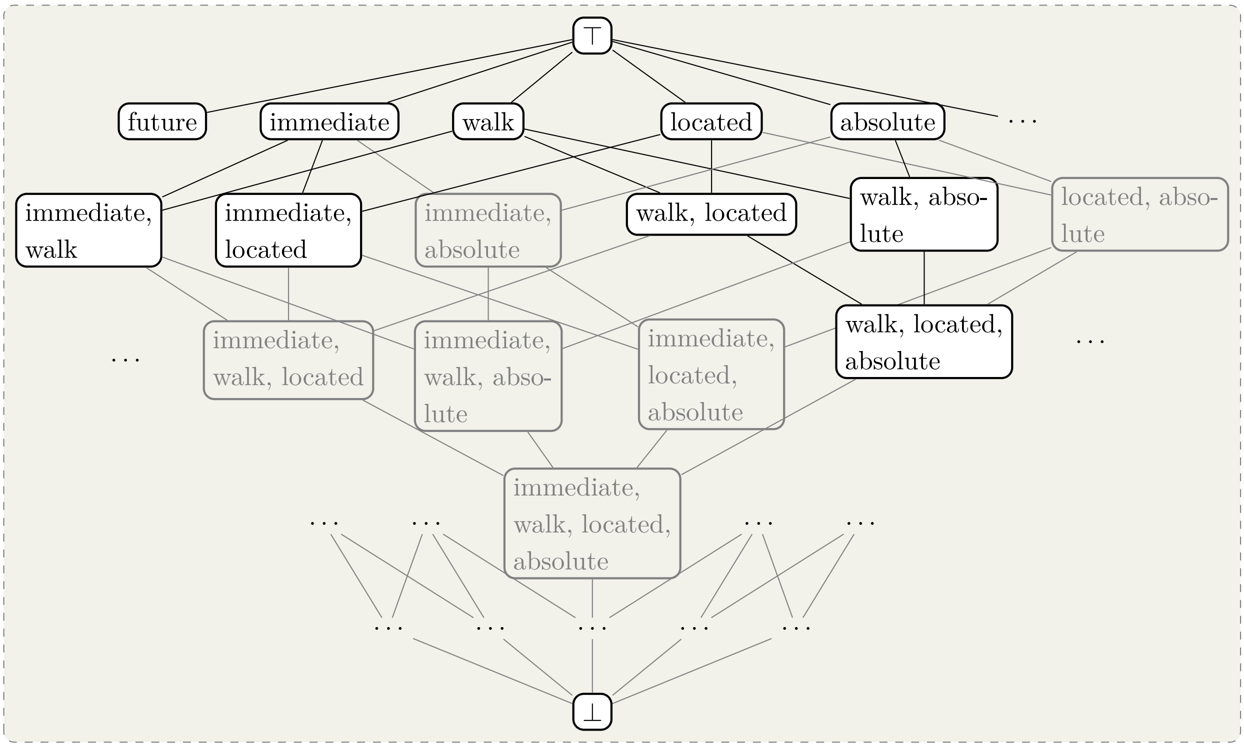


Figure 8.10: Part of the context lattice for **near** initialized with data acquired from the data analysis

Exemplars Exemplars constituting contextualized concepts for **near** are used to calculate a prototype referred to as an interpretation for **near**. For the non-spatial contexts, extra exemplars are included: time, relationship, and information, which are also considered prototypes. The prototype determination for the spatial meaning is considered further. So far, the method to calculate the prototype is creating an observation table (Table 4.4) including contextual typicalities, and selecting that exemplar as prototype that has the highest typicality. This method deals with exemplars on a nominal scale [Stevens, 1946], which are clearly distinct from each other. In the data acquisition for **near** the exemplars are scattered; having only sparse occurrences with equal numbers of occurrences means the proposed prototype determination method given by Equation 4.29 cannot be applied to determine a prototype. The exemplars acquired during the process are distances categorized on a ratio measurement scale. The feature of a ratio measurement scale is that exemplars are not clearly distinct from each other. Methods like average, median, or mode are used to reduce the distribution to a central tendency. These measures cannot be used in this case because the data are too scattered.

Kernel Density Estimation (KDE) is selected as a method to determine the prototype for exemplars on a ratio measurement scale. KDE is a generalization of a histogram generated for nominal data [Silverman, 1986]. The benefit of this method is that exemplars do not have to be categorized, which is problematic because categorization is in itself context-dependent. To establish the KDE distribution, the kernel is placed on each data point and adds up if the data point occurs more often in the data set. The established distribution is similar to an observation table used to determine contextual typicality. Within the contextual typicality, the maximum value determines the prototype.

A result for a contextual typicality distribution for **near@**$\top$ created by KDE is given in Figure 8.11 and Figure 8.12. In Figure 8.11 a snapshot of the distances in the range from zero to 1000 meters is shown, and the entire distribution is given in Figure 8.12. From the distribution in Figure 8.11 the distance with the highest KDE value is selected as prototype, included as a point in Figure 8.11. For the contextualized concept **near@**$\top$ the prototypical distance is 190 meters.

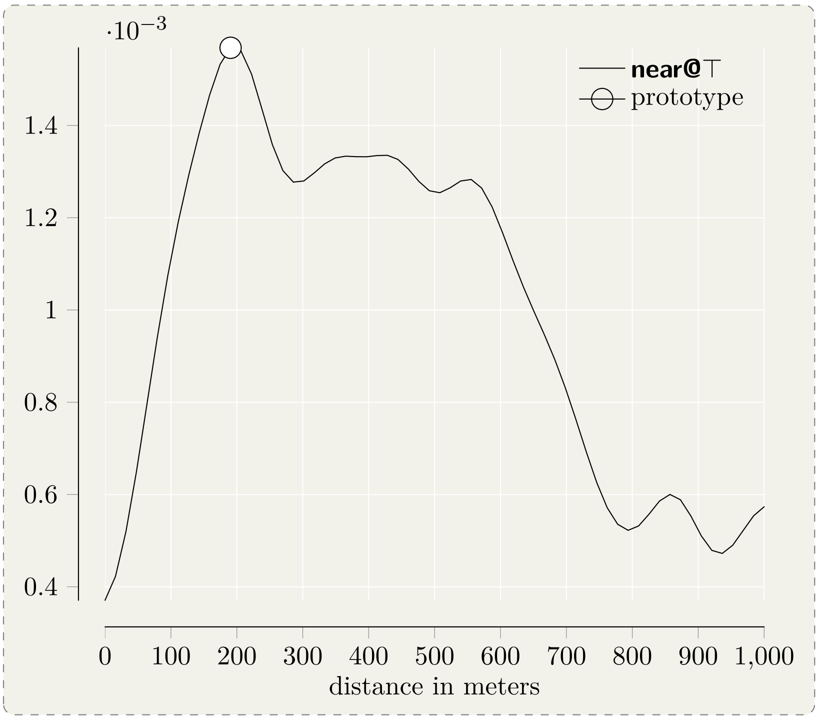


Figure 8.11: Contextual typicality of **near@**⊤ clipped from 0 to 1000 meters, the point indicates the prototype

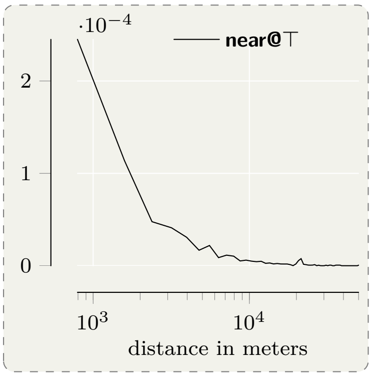


Figure 8.12: Contextual typicality of **near@**⊤ from 0 to 50.000 meters

The KDE algorithm has to be included in the implementation of the context algebra. The Haskell library statistics[12] includes a function to calculate KDE. To integrate the function, a new module `RatioExemplars` is created that instances the abstract class `ContextualizedConcept` by implementing the function `calculatePrototype`. The implementation is given in Algorithm B.30 in Appendix B.5. Here the usage of abstract classes introduced before is beneficial, because the user of the context algebra can now change between the modules `RatioExemplars` and `NominalExemplars` without changing other code.

The KDE implementation creates contextual typicalities for **near** that are illustrated in figures included in Section B.5. Those contextualized concepts having at least five "located" categorized real estate entries are included in Figures B.3, B.4, B.5, B.6, B.7, and B.8. The contexts in the Figures are given in German language, a translation for each context is included in Table B.4. In summary, eighteen contextual typicalities are determined using KDE and illustrated in six figures included in Appendix B.5.

8.5 *Applications using the context algebra representing "near"*

Possible applications for the representation of **near** with the context algebra are discussed. The possibilities the context formalization offers are given in examples usable for future applications.

[12] http://hackage.haskell.org/package/statistics

Natural Language input

The context algebra can be applied to applications that use natural language input, to transform the natural language into metric distances. From the resulting distances, a prototype is determined which results into one metric distance. This metric distance can be used as input for spatial algorithms as envisioned in the introduction in Section 1. A schema for how the context algebra can be used in an application is given in Figure 8.13. A benefit of using context algebra is that the gap between vague natural language terms and the precise metric distances needed for spatial algorithms is bridged. This is visualized in Figure 8.13, where natural language text (represented as “. . .” in Figure 8.13) is transformed into text resulting in contextualized concepts for **near** and **near@***Bern* and used as input for the context algebra. The output of the context algebra is a precise metric distance obtained from the determined prototype of the contextualized concept. As the contextualized concept can change, e.g **near** referring to the contextualized concept **near@**$\top$, resulting in 200 meters, the prototypes can also change, *e. g.* **near@***Bern* results in 1000 meters. The context algebra integrates the dynamic change of intended meaning by including the influencing context to approximate the intended meaning of a vague term. How such a context algebra could be used in applications for natural language input is outlined below.

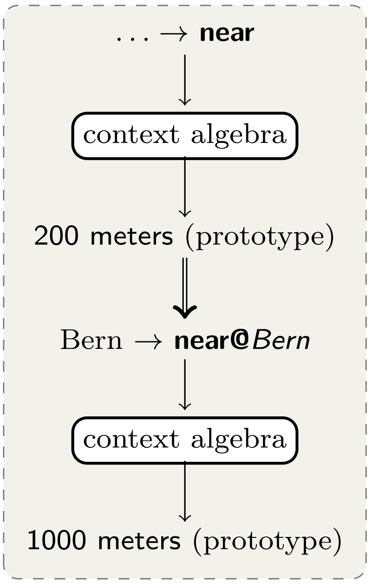


Figure 8.13: Application of the context algebra to translate natural language input (near) into distances

Distinguishing between meanings For applications dealing with natural language input it is beneficial to identify which meaning of **near** is used. As **near** is a semantic primitive, it is challenging to identify its meaning. Context algebra is able to distinguish different meanings (temporal, spatial, relationship, etc.) according to the influencing contexts.

A context algebra initialized with a data set including contexts that relate to specific meanings is key to distinguishing between multiple meanings of **near**. By following a conversation, all sentences can be parsed and those that include **near** or influencing contexts are processed by the context algebra. The gradually added context during a conversation is used to generate different contextual concepts by querying the context lattice for the found contexts. These contexts create the mapping to exemplars that are used to determine a prototype. The prototype identifies the meaning in which it is used, *e. g.* “10 minutes” for temporal usage.

In the initialized context algebra for **near**, temporal meaning, relationship meaning, and spatial meaning can be distinguished. For example, if the context *future* is included in the natural language input, the context algebra creates the contextualized concept **near@***future*. In the initialized context algebra, this contextualized concepts maps to only one exemplar that indicates use of the temporal meaning of **near**.

Translation mechanism The initialized context algebra for **near** translates the word “near” into a distance usable for spatial algorithms.

Hahn et al. [2016] described such a use of the context algebra that splits the concept **near** into contextualized concepts producing different prototypical distances according to context. The prototype of a contextualized concept is mapped from **near** onto a metric distance, as given in Figure 8.13.

For example, when describing the location of your flat to a new colleague with the sentence "My flat is located near Maria-Theresien-Platz", which distance does **near** relate to? The context algebra is able to determine the intended distance. First, the influencing contexts are identified: here, the context *located*. Second, *located* is selected from the context lattice and the contextualized concept **near@***located* is established. Third, the exemplars included in **near@***located* are processed by KDE to form a contextual typicality function given in Figure 8.14. Fourth, from this function the maximal typicality is designated as prototype, which for **near@***located* results in about 570 meters, given as point in Figure 8.14.

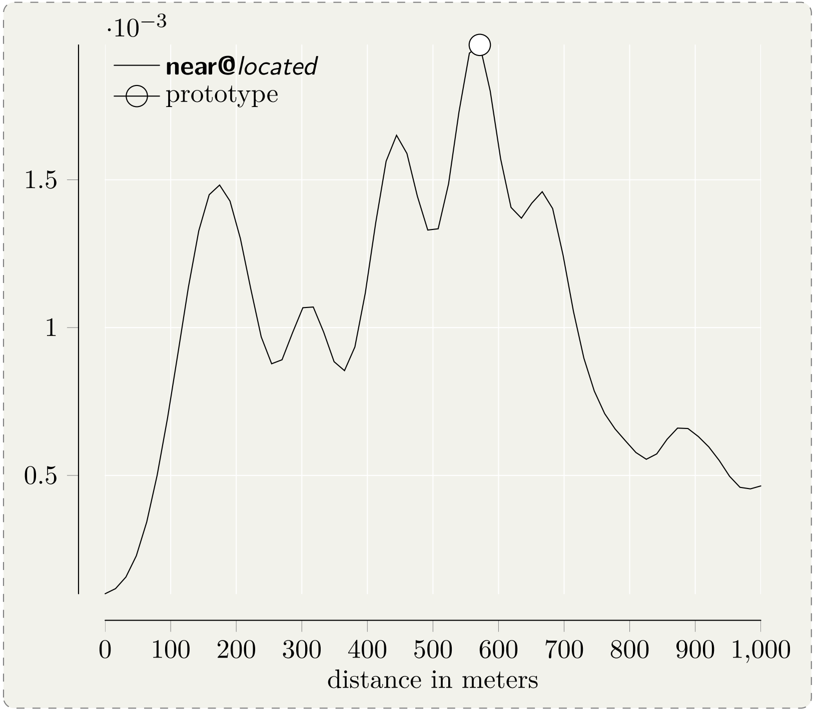


Figure 8.14: Contextual typicality for **near@***located*

The translated distance from the contextualized concept **near@***located* can be used as input for a spatial algorithm. For example, if the new colleague wants to find a flat located near Maria-Theresien-Platz he can use a spatial algorithm (*e. g.*`within`) using the 570 meters as radius to look for flats.

Natural language output

The context algebra can also be applied to applications that produce natural language output. For such applications the context algebra received a metric distance as input, and translates this distance into a natural language output. For example "200 meters" is translated into "near". A schema for this process is indicated in Figure 8.15.

The determination of the natural language output is a process following five steps: First, prototypes are determined for all contextualized concepts. This results in an array of metric distances. Second, these are compared to the input metric distance. Third, the metric distance with the least deviation is selected to be the appropriate language output. Fourth, the selected prototype is translated into the contextualized concept which links to natural language. Fifth, the contextualized concept, *e.g.* **near@**$\top$ is selected as the resulting natural language output, *e.g.* near.

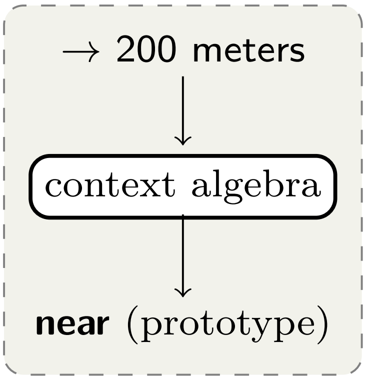


Figure 8.15: Application of the context algebra to translate distances into natural language (near)

Selecting the appropriate word to describe a distance One example of applications that produce natural language outputs are wayfinding applications. For example, Xu et al. [2014] observed from a web corpus that wayfinding systems generate route descriptions that include metric distances, whereas human-generated route descriptions usually include vague spatial terms. For example, the wayfinding system states a description as: "Turn right in 600 meters". In contrast, a human description could be: "Turn right near the statue". By equipping the wayfinding system with a context algebra, the description could include natural language derived from metric distances.

Context algebra can be used to translate a distance into a verbal description. For example, the distance of "600 meters" is to be translated into a verbal description. If no additional context is given, the prototype of the $\top$ context is selected and compared to the distance. For the established model of **near@**$\top$, the prototype is about 190 meters (prototype in Figure 8.16). This prototype, however, is not a good fit for the distance of "600 meters". To find a better match, the context lattice can be queried for more selective contexts. If the user can provide a more refined context, *e.g.* *walk*, the prototype for the contextualized concept **near@***walk* is determined and compared to the distance. In this case the prototype for **near@***walk* equals 550 meters (prototype in Figure 8.16) which fits much better than the contextualized concept **near@**$\top$. Now the context lattice can be used to check if a more selective context has a prototype with less deviation. Indeed the more selective contextualized concept **near@***immediate walk* has the prototype of about 580 meters (prototype in Figure 8.16), which fits even better with the distance input. This prototype can be used to produce the natural language description resulting in "near immediate walk".

Object influence to near

Two studies concluded that the size of the object that a spatial relation refers to influences the understanding of distance. The first study by Morrow and Clark [1988] found that "distance is judged to be larger, all else being equal, the larger the landmark, the larger the figure, ...". The second study by Carlson and Covey [2005] concluded that

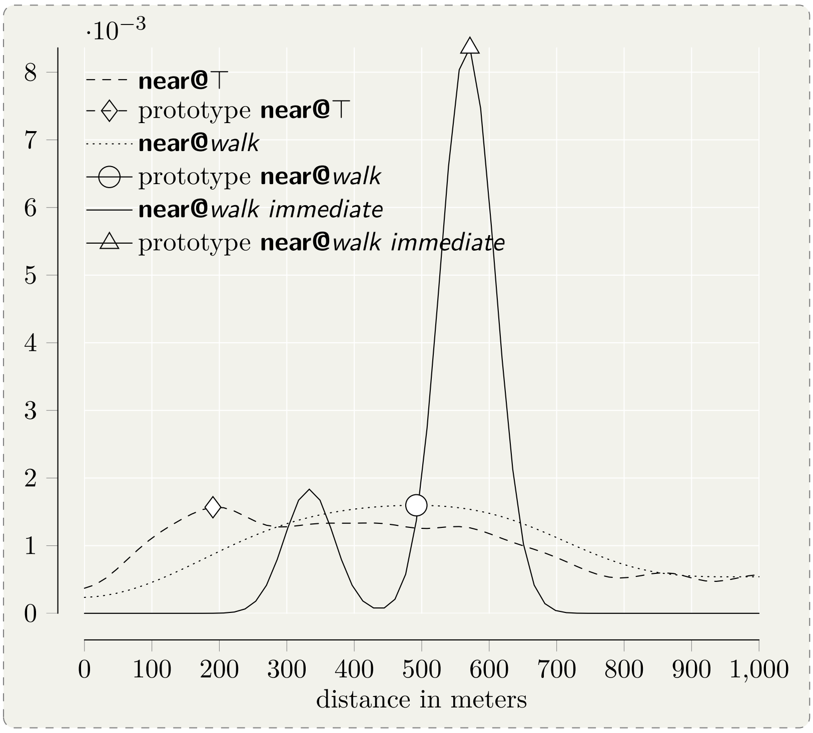


Figure 8.16: Contextual typicality for **near@**⊤, **near@***walk* and **near@***walk immediate*

distance terms (e.g near) and terms that do not convey distances (*e. g.* left) are both "systematically influenced by the size of the objects" they describe. Context algebra is able to model this influence of the size of the object. The relatum and the locatum can be included as context information for the distance relation. The established model for **near** accounts at least for the locatum of the spatial relation near.

Which flats are near the city center, the subway, or the lake? The locatum has influence on the distance that a spatial relation refers to. The data that initialize the context algebra for **near** include several locata that can be used to verify if a context algebra can model this behavior. To distinguish between distances according the size of the locatum, three different locata are selected: *center*, *lake* and *subway*. These locata are used to influence **near**, which creates the contextualized concepts **near@***center*, **near@***subway* and **near@***lake*. The contextual typicalities for these contextualized concepts are included in Figure 8.17. In this figure, the prototypes for all three contextualized concepts differ, *e. g.* the prototype for **near@***center* is about 1300 meters, the prototype for **near@***lake* is about 1050 meters, and for **near@***subway* it is about 380 meters. This demonstrates that context algebra is able to represent the object's influence on the spatial relation near.

8.6 Conclusion

In this chapter, the context algebra is initialized with acquired data, and benefits for using a context algebra for existing applications are

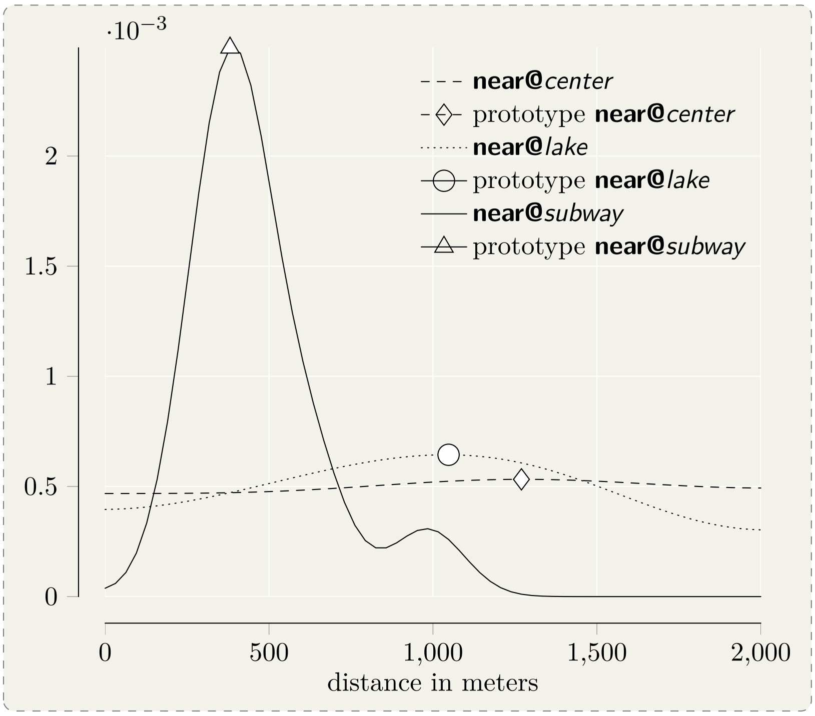


Figure 8.17: Contextual typicality for **near@***center*, **near@***lake*, and **near@***subway*.

pointed out. The context algebra is initialized to represent the concept **near**. Conclusions drawn from representing **near** can be distinguished into three parts: (i) requirements for data initializing the context algebra, (ii) how data can be acquired, and (iii) applications for the resulting model.

How can data be acquired to respect a connection between a word and reality? The fourth research question of this thesis addresses: How can data be acquired that respect a connection between a word and reality? In this chapter this question is answered by creating requirements for data:

- Context is explicitly stated in the data.
- Data have to include a reference to reality to be grounded.
- To investigate the spatial meaning, references to the earth have to be included in the data.

Only data sets that are able to satisfy all requirements can be used to initialize the context algebra. With these requirements, two real estate data collections (available in the German language) including about 40000 entries are selected. These real estate entries are filtered for the occurrences of near, which resulted in about 9500 entries that are further processed.

Existing data acquisition techniques (questionnaires, parsing tress, etc.) are unable to extract all the required data, making a new, manual process necessary. The data acquisition and extraction workflow is guided by a web application. In this acquisition process, the real estate data are manually processed to identify contexts and create references to reality on a map. Contexts are extracted from the description of the real estate object. The coordinates of a real estate object (relatum) and the coordinates of another object mentioned with a near relation (locatum) are identified on a map. At the end of processing, the object is classified by whether it is usable to investigate the spatial meaning of **near**. Context, location, and description of a relatum (the real estate object) and locatum (the object near to the real estate object), as well as the classification are stored in a spatial database which creates a knowledge base for the representation of **near**. The manual acquisition process was specially designed for this use case, but is too time-consuming for a practical application. Further research on how to automatically detect influencing contexts in texts is necessary, and NLP may offer promising methods in that regard in the future.

The stored data are analyzed, and four different meanings of **near** that can be used are extracted: temporal, relationship, information, and spatial. All these meanings have different contexts, making it possible to distinguish between them. The spatial meaning for **near** is further investigated in order to extract data for the transformation of natural language into metric values.

The data analysis showed evidence for the context structure proposed in Section 4.3. In the data set the word near was influenced by no context in 62 % of cases, while 37 % showed influence by one context, and 1 % by a combination of contexts (two or more contexts). The contexts are combined from several "basic" ones, *e.g.* *walk*, *located*, to combined contexts, *e.g.* *walk, located* and *located, very*. The maximum combination of contexts are three conjunct contexts, *e.g.* *located, airport, train*. In the analysis, only conjunct contexts were identified; no disjunct contexts were included in the data.

For the spatial meaning, contextualized concepts are established where each includes a prototype. The exemplars used to determine the prototype range from five meters to 100 kilometers. The exemplars are now given on a rational measurement scale in contrast to the exemplars given on a nominal scale considered so far. Therefore, the prototype determination algorithm has to be changed. The method of Kernel Density Estimation (KDE) is chosen to determine the prototype. The distribution of the KDE is taken as contextual typicality, where the maximum value is selected as prototype.

The representation for **near** is initialized with the located categorized real estate entries. 72 contexts are obtained from located entries that populate the context lattice. For all these contexts, a mapping to subsets created by 1391 exemplars is included.

Applications The representation of **near** can be used to translate natural language into metric distances. It is able to distinguish between several meanings in which **near** can appear, *e.g.* temporal, information, relationship and spatial meanings. It can translate natural language input to metric distances that can be used as input for spatial algorithms. For example, the natural language sentence "... my flat is located near subway station Pilgramgasse " is translated into a metric distance of 570 meters for the contextualized concept **near@***located*.

The representation of **near** can also be used to translate metric distances into natural language. For example, route descriptions can be expressed with natural language, *e.g.* "550 meters" is translated into "near walking".

The representation of **near** is able to model the results of cognitive experiments. Studies concluded that the size of objects used in combination with **near** has an influence on the distance **near** refers to. The context algebra is able to model the influence of the objects to **near**, *e.g.* **near@***lake* results in 1050 meters whereas **near@***subway* results in 380 meters.

In conclusion, the context algebra for the concept **near** is able to distinguish between contexts and generate different distances according to context influence.

Research questions arising from this application of context algebra are:

- Are there other relevant contexts influencing near? A first direction would be to determine contexts from other data sets.
- What are relevant contexts influencing spatial terms in general?
- Where to collect necessary data to initialize the context algebra?

9 Conclusion and Future Work

9.1 Conclusion

To evaluate the hypothesis, the conclusions of this work are stated with respect to the research questions. Each research questions is answered by proposing a context algebra, implementing it, testing it, and applying it to represent the concept **near**.

At the end of this section, a short summary is given highlighting the main contributions.

Research Question 1: What properties does context have that a general context operation must respect? Contexts are partially ordered with the relation "is more selective than or equally selective as", denoted $\leq$. Selectiveness makes sense in light of the definition of context: *"Context is any information that selects appropriate references from a word to objects in reality".* For that relation a converse relation "is less selective than or equally selective as", denoted as $\geq$, exists. Both relations satisfy the algebraic properties of *reflexivity* and *transitivity.*

The partial order relation includes a universal greatest context and a universal least context. The least context, denoted $\top$, is less selective than all other contexts, *e.g.* *serving pizza or in Vienna*. The greatest context, denoted $\bot$, is more selective than all other contexts, *e.g.* *serving pizza in Vienna*. $\top$ and $\bot$ context can only be defined with respect to a use case, and are not generally applicable.

Contexts are combined by the conjunction and disjunction operations, and contexts have complements. Two conjunct contexts establish another context that is a more selective than or equally selective as both constituents. Two disjunct contexts establish another context that is less selective than or equally selective as both constituents. Both operations respect the properties of *idempotency*, *commutativity*, *associativity*, *consistency*, and *isotone.* Special laws for $\top$ and $\bot$ are included for combination operations, *e.g.* $c_i \in C \qquad c_i \wedge \bot = \bot \qquad \bot \vee c_i = c_i$. A complement denoted $\overline{c}$ for context is included in the context algebra satisfying axioms, *e.g.* $\overline{\overline{c}} = c$.

Within these properties, the context algebra represents context in a lattice structure. The operations are realized as set union and set intersection including special cases for $\top$ and $\bot$ context.

In conclusion, context is partially ordered and can be complemented, and two contexts can be conjunct and disjunct. The partial order relation and complement, and the conjunct and disjunct operations respects their own properties and result in a lattice representation for context.

Research Question 2: How can a general context operation be connected to words and to reality? The semiotic triangle is used as starting point to connect words with reality. It is enriched with context to distinguish between objects in reality. The connection is established through the following approach:

1. Observations connect exemplars with context in reality, *e.g.* the exemplar 200 meters in the context walking is one observation for **near**.

2. All observations are mapped to the context lattice to establish contextualized concepts.

3. A contextualized concept corresponds to a word used in a context.

The formalization of the observations, context lattice, concepts, and contextualized concepts is split into two parts. The first part uses the formalization established in research question 1, the second part formalizes the mapping from context to concepts creating contextualized concepts.

The mapping from the context lattice to concepts is established through contextualized concepts. Contextualized concepts summarize observations of reality for a specific context. All contextualized concepts together build the concept that is connected to a word, which establishes the connection from a word to reality.

Research Question 3: What is necessary to implement the general context operation, assuring all context properties are respected, and what class of complexity can be determined? In order to implement the general context operation, the context algebra has to be implemented first. It is implemented with the functional language Haskell using abstract classes, instantiating them using set functions. The implementation is available online at hackage via the URL: `https://hackage.haskell.org/package/ContextAlgebra`. This implementation supports the implementation of the general context operation. Its use is described for the restaurant example introduced in the introduction.

To ensure that the implementation satisfies all laws stated in the context algebra, algebraic property tests are executed. For each function. *e.g.* partial order relation, conjunction, and disjunction, property tests are established and executed with 1000 randomly generated input contexts. The execution results in a report showing that each property test was successfully executed which assures us that the claimed laws are satisfied.

As many contexts may influence a word at one time, the implementation is executed with different number of inputs to identify its complexity class. Benchmarks for context lattice generation time and memory consumption as well as the time used to calculate a prototype are executed. All benchmarks show a unique result for the implementation, resulting in an exponential complexity class.

Research Question 4: How can data be acquired that respect a connection between a word and reality? Methods to extract context and references to reality from data are needed to create representations of real world entities. A review of existing approaches concluded that they fall short when it comes to identify influencing context. In order to acquire the necessary data, a web application is created that users can use to acquire the necessary data. With this application the data acquisition process is outlined, which is complex and requires more automation if it is to be applied on larger scales. The key is that the application outlines which data must be acquired to establish a connection between words and reality.

Data have to meet the following requirements to initialize the context algebra:

- Context is explicitly stated in the data.
- Data have to include a reference to reality to be grounded.
- To investigate spatial meaning, references to the earth have to be included in the data.

For the representation of the concept **near**, media including real estate advertisements meet all requirements. Real estate data include descriptions mentioning the word near for multiple meanings, *e. g.* temporal, information, relationship, spatial. Additionally, the descriptions include contexts that influence the concept **near**. To create a reference to reality, the real estate entry is located on a map and connected with its coordinates. To investigate the spatial meaning, the distance that **near** mentions from the entry to another object on earth is included in the database. This is done by locating the other object on the map.

The acquired data are processed to be usable for the context algebra. The contexts are lemmatized, which brings them into the canonical form of the word. Distances are calculated to be usable as observations for building contextualized concepts.

Research Question 5: What services can be generated by using a general context operation for natural language? Problems solved from the representation of **near** with a context algebra are shown in several use cases.

One class of use cases translates natural language input into metric distances. Depending on the context, different metric distances are calculated. For example, the natural language input “My flat is located near subway station Pilgramgasse” is translated into the contextualized concept **near@***located*, resulting in 570 meters. The benefit is that the general context operation bridges the gap from vague natural language terms to metric distances, as is required for their use as input for spatial algorithms.

Another class of use cases produces natural language output from metric distances. For example, wayfinding systems internally calculate route directions based on metric distances which are presented to the user as-is, *e.g.* "Turn right in 600 meters". As such descriptions are different from descriptions usually given by humans, context algebra can translate the metric distance into vague spatial terms. The benefit here is that the output of spatial algorithms can be transformed into vague spatial terms better understandable by users.

Regarding **near**, cognitive studies demonstrated the influence of the sizes of objects to the intended distance **near** refers to. For example, the distance for **near@***lake* is larger compared to **near@***subway*. This effect can be modeled by context algebra, resulting in 1050 meters for **near@***lake* and 380 meters for **near@***subway*. The benefit here is to have a model for cognitive effects influencing **near**.

Evaluation of the hypothesis Based on the answers to the research questions and the applications for **near**, the hypothesis is confirmed.

A GENERAL CONTEXT OPERATION SELECTS REFERENCES FROM A WORD TO OBJECTS IN REALITY.

The thesis can be summarized into four main results. Several results use the perspective of the constitutive rule of Searle [1995]: "X counts as Y in context C".

1. A context formalization is created that acts on a form of mapping between $X \to Y$ that is composable and respects properties that are assumed from context.

2. The elements X and Y can represent different things, *e.g.* *words* → *things*, *words* → *properties*, *pictures* → *things*.

3. Two models for context formalization are presented, a mathematical one and one implemented in the functional programming language Haskell.

4. A proposal for how context formalization can be used for spatial information processing, *e.g.* data collection, user query, and user instructions is included.

9.2 Future Work

There are several directions for future work. One direction is to invest in context research, another direction is data acquisition, and yet another one is to identify possible applications for context algebra.

Context

This work presented a formalization for context that can be pushed further as cognitive research provides new insights.

The number of context included in the context lattice shows an exponential increase, which may limit the usability of context algebra. The number of atomic context representations is determined as a critical factor for this increase. Future work has to find mechanisms to keep the number of atomic context representations low in order to avoid an exponential explosion. Perhaps distinctions of different kinds of contexts could be beneficial. For example, Janowicz [2008] distinguished context into six kinds that have an influence on similarity judgments. It remains a question for future work which distinction could be beneficial for context algebra.

Here context is used to distinguish between meanings of a word. In data analysis, it is found that contexts used in conjunction with concept can be used to detect the meanings in which the concept is used. For example, the context *walking* represents a spatial meaning of the concept near. Maybe a distinction of context within the usage of the concept can lead to a classification of different kinds of contexts? One direction for investigation could be combining context algebra with WordNet or a similar collection that includes meanings for words. The explanations and synsets describing meanings can be used to calculate a correlation between used contexts and synset of the word.

By how much context are we influenced? The data analysis of real estate entries concluded that no more than three contexts are used in one sentence. Future research has to point out by how many contexts we can be influenced. Perhaps existing contexts are dropped if a new context is? Maybe contexts are grouped together?

What happens if the receiver did not understand the message from the sender? How much of the previous context is dropped? A first direction is the formalization of Weiser and Frank [2013], building a communication model.

Does the order of context influence matter? The context algebra assumes that two contexts are commutative, which means the order of influence has no effect on the result. Experiments by Meyer and Schvaneveldt [1971] became prominent as a priming effect, where the first word influences (primes) the response of a second word. If this effect also occurs with context, it would raise new research questions.

Mobile map applications should adapt to user needs using context to minimize user interaction. Raubal and Panov [2009] presented an “AdaptationModel” based on formal models for context, users, and tasks which is applied to a pedestrian navigation service. They concluded that user interaction and cognitive load for the user was reduced by including context. The aspect to select map features according to context was addressed by Hahn and Frank [2014]. They applied the state context property model (review in Section 3) to select features for the concept of “map” combined with other concepts. Their focus was on the guppy effect, which may occur also for the concept map. To model the guppy effect, they entangled the concept “map” with the concept “buy” to build a “buy map”. Comparing the features for these three concepts they observed that map features are different, following the guppy effect pattern. Context algebra is also able to model the guppy effect, but future research has to be conducted to identify such geographical guppy effects in experiments with subjects.

Data

Data are necessary to initialize the context algebra. In this thesis, a time-consuming manual data acquisition process was presented. Use of context algebra will be limited by this time-consuming manual data acquisition process. Future research has to come up with automatic data acquisition methods.

Two types of automated procedures are possible. The first method extracts data from sensor data. The second method extracts data from natural language input.

Data extracted from sensors involves a categorization step to build a concept and faces the symbol grounding problem. The classification process is also influenced by context, which contexts are influencing the classification is the key for classification. A possible approach was presented by Keßler et al. [2007] that looked on the impact of different context parameters on semantic similarity measures. If this approach is useful in overcoming the categorization issue, then the data obtained from sensors can be used to initialize the context algebra. This will be beneficial because the data can be updated and results will adapt in real time. Another data source could be life logging [O'Hara et al., 2009], where persons collect data about their life and make them publicly available. Data extracted from sensors, for example included in a mobile phone, face the symbol grounding problem that has to be addressed.

Data extracted from natural language involves natural language processing tools. Natural language given as written text is a possible data source. What is needed are techniques to extract the necessary data from the written text. A starting point is to detect the meaning of a written text, a promising method using machine learning applied to detect spatial-temporal information from narrative discourses was presented by Howald and Katz [2011] and may apply here too. From written text, the context influencing the entity has to be identified. A possible method how to identify contexts are parsing trees built by natural language processing tools [Manning et al., 2014]. Furthermore, the entities have to be identified and linked to reality. As many entities link to regions in reality these regions have to be identified. An approach to automatize the identification of regions was introduced by Hobel et al. [2016]. Hobel et al. presented an algorithm based on machine learning on how to identify regions. To link entities given by addresses to reality, gazetteers provide the desired functionality. The main research question is how to extract exemplars and context in natural language.

Prototype calculation Prototype calculation methods are presented for exemplars categorized on a nominal and on a rational measurement scale in this work. In the same manner all other Stevens's measurement scales have to be considered and methods implemented how to calculate the prototype.

Applications

Future applications for the context algebra are sketched.

Personalized knowledge base Personalized applications can benefit from an integration of context algebra. This can result in a personalized context algebra using observations valid for a single person only.

A personalized context algebra is based on data from a single person. How data can be collected for a single person could generate challenges. To get a data set for a single person that covers all the experiences the person has had is, of course, impossible. Maybe online social networks already collect enough data? McKenzie and Raubal [2011] presented a framework to extract "spatial and temporal location[s]" that may be fit to initialize the context algebra.

Questions arising from a personalized context algebra are: is a personalized context algebra able to interpret inputs in a similar fashion to the owner of the personalized context algebra? If this is true, it could be beneficial for other users or companies, for example for the advertisement industry. Personalized context algebra could *e. g.* predict how a user interprets an advertisement. If the interpretation is not the intended interpretation it could be adjusted. How to deal with data privacy issues? Where will the data be stored, analyzed, communicated?

Dynamic context algebra In this work a contextualized concept was initialized with observations made in the past. Future work can change this static data to dynamic data. One application can be a personalized context algebra for the case that the user relocates. The addition would be that observations and contexts can be edited after initializing the model. If the context algebra is dynamic, new observations can lead to generalization, specialization, and dynamic weighting of the context lattice as proposed for spatial theories by Twaroch and Frank [2005].

With the dynamic change of the context lattice functions to add and remove contexts have to be introduced. Adding a context to the context lattice implies a complete new organization of the lattice. New atomic context representations have to be identified and all possible combinations have to be included in the context lattice. Removing a context from the context lattice is not necessary, there can be synonyms to the $\perp$ context.

The dynamic change of observations building contextualized concepts requires more functionality of the implementation. Similar to the approach of Raubal [2008], adding observations to the concept can be interpreted as including new experiences in the model. This can affect the prototype calculation, and result in another prototype. Removing observations from the concept can be interpreted as forgetting. This may affect the prototype as contextual typicality changes. An idea is that not renewed observations are dropped with a function of time as proposed by Ebbinghaus [Leipzig 1885].

Context algebra to mediate between users Humans take into account the addressee of the information, and adjust the information appropriately. For example, Hölscher et al. [2011] studied if participants would follow their own route description and concluded that the assumed conceptualization of an addressee systematically affects the planning and description of route description. Context algebra could be used to adjust the descriptions according to two knowledge bases.

If the knowledge bases of all contributors are known, an additional context algebra system can adjust the descriptions. *First*, knowledge bases can be compared if all contexts and entities are known to all contributors. If, for example, one knowledge base does not include St. Stephans Cathedral, another expression has to be found for a route description. Another possibility is to provide the information to the user that lacks the knowledge, or to ask the user if they are really not aware of this entity. All possible cases to consider for wayfinding instructions are presented by Weiser [2014] in his dissertation. *Second*, context algebra uses exemplars referring to reality to mediate between the two knowledge bases. For example, using the corresponding word for contributor A and using another word for contributor B. Hahn and Weiser [2015] already investigated this line of research which can be used as starting point to solve communication coordination problem. *Third*, context algebra could signal to a user that another user is not aware of the used entity.

How such a mediation process between two users for the spatial meaning of **near** can look is outlined in the following. The prerequisite for a mediation process is that for all users a knowledge base is present. If an additional system can reach both knowledge bases, the system can take descriptions made from the first person and translate it for the second person that uses the second knowledge base. This system uses the natural language input from the first person, translates it into a distance, and uses the second knowledge base to produce appropriate descriptions for the second person.

Gazetteer application A gazetteer translates descriptions from locations into coordinates. The process is to parse the descriptions that are given in words, and to look in a knowledge base if an entry is included. If a matching entry is found it is returned. The input words can refer to many objects in the world, *e.g.* the input “near subway station” refers to thousands stations over the world.

If context algebra is included in such a system, it can be used to distinguish between descriptions that are worth processing and those that are too unspecific; *e.g.* subway station will not be processed. The decision if something is worth processing can be made by counting the number of references for the description. Another approach for the decision is to use the position in the context lattice. If the context is too little selective, more context has to be provided by the user.

Context algebra can also be used to decide which information is most beneficial to query the user for. If the user transmits a description that is too unspecific, context algebra can be used to determine which further contexts will select the most appropriate references and these contexts can be requested from the user.

The dissertation has answered a few questions and contributed to our understanding of context – as a result, more and more precisely formulated questions are posed and await new answers.

Bibliography

Benjamin Adams and Martin Raubal. *A Metric Conceptual Space Algebra*, pages 51–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-03832-7. DOI: 10.1007/978-3-642-03832-7_4. URL http://dx.doi.org/10.1007/978-3-642-03832-7_4.

Stephen Adams. Functional pearls efficient sets—a balancing act. *Journal of Functional Programming*, Volume 3:553–561, 1993. ISSN 1469-7653. DOI: 10.1017/S0956796800000885. URL http://journals.cambridge.org/article_S0956796800000885.

Gediminas Adomavicius and Alexander Tuzhilin. *Recommender Systems Handbook*, chapter Context-Aware Recommender Systems, pages 217–253. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_7. URL http://dx.doi.org/10.1007/978-0-387-85820-3_7.

Diederik Aerts and Sven Aerts. Applications of quantum statistics in psychological studies of decision processes. In BasC. van Fraassen, editor, *Topics in the Foundation of Statistics*, pages 85–97. Springer Netherlands, 1997. ISBN 978-90-481-4792-2. DOI: 10.1007/978-94-015-8816-4_11. URL http://dx.doi.org/10.1007/978-94-015-8816-4_11.

Diederik Aerts and Liane Gabora. A theory of concepts and their combinations i: The structure of the sets of contexts and properties. *Kybernetes*, 34(1/2):167–191, 2005a. DOI: 10.1108/03684920510575799. URL http://dx.doi.org/10.1108/03684920510575799.

Diederik Aerts and Liane Gabora. A theory of concepts and their combinations ii: A hilbert space representation. *Kybernetes*, 34(1/2): 192–221, 2005b. DOI: 10.1108/03684920510575807. URL http://dx.doi.org/10.1108/03684920510575807.

Varol Akman and Mehmet Surav. Steps toward formalizing context. *AI magazine*, 17(3):55, 1996. DOI: http://dx.doi.org/10.1609/aimag.v17i3.1231.

Harald Atmanspacher and Hartmann Römer. Order effects in sequential measurements of non-commuting psychological observables. *Journal of Mathematical Psychology*, 56(4):274 – 280, 2012. ISSN 0022-2496. DOI: http://dx.doi.org/10.1016/j.jmp.2012.06.003.

URL http://www.sciencedirect.com/science/article/pii/S0022249612000727.

Harald Atmanspacher, Hartmann Römer, and Harald Walach. Weak quantum theory: Complementarity and entanglement in physics and beyond. *Foundations of Physics*, 32(3):379–406, 2002. ISSN 0015-9018. DOI: 10.1023/A:1014809312397.

A Baddeley. Working memory. *Science*, 255(5044):556–559, 1992. DOI: 10.1126/science.1736359. URL http://www.sciencemag.org/content/255/5044/556.abstract.

Linas Baltrunas, Bernd Ludwig, Stefan Peer, and Francesco Ricci. *Design, User Experience, and Usability. Theory, Methods, Tools and Practice: First International Conference, DUXU 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011, Proceedings, Part I*, chapter Context-Aware Places of Interest Recommendations for Mobile Users, pages 531–540. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21675-6. DOI: 10.1007/978-3-642-21675-6_61. URL http://dx.doi.org/10.1007/978-3-642-21675-6_61.

Lawrence W. Barsalou, Wenchi Yeh, Barbara J. Luka, Karen L. Olseth, Kelly S. Mix, and Ling-Ling Wu. Concepts and meaning. *Beals, K., et al. (eds.) Parasession on conceptual representations*, University of Chicago, Chicago Linguistics Society:23–61, 1993.

Irina Basieva and Andrei Khrennikov. *Quantum Interaction: 8th International Conference, QI 2014, Filzbach, Switzerland, June 30 – July 3, 2014. Revised Selected Papers*, chapter Quantum(-like) Formalization of Common Knowledge: Binmore-Brandenburger Operator Approach, pages 93–104. Springer International Publishing, Cham, 2015. ISBN 978-3-319-15931-7. DOI: 10.1007/978-3-319-15931-7_8. URL http://dx.doi.org/10.1007/978-3-319-15931-7_8.

Mary Bazire and Patrick Brézillon. Understanding context before using it. In Anind Dey, Boicho Kokinov, David Leake, and Roy Turner, editors, *Modeling and Using Context*, volume 3554 of *Lecture Notes in Computer Science*, pages 29–40. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26924-3. DOI: 10.1007/11508373_3. URL http://dx.doi.org/10.1007/11508373_3.

Enrico G. Beltrametti and Gianni Cassinelli. *The logic of quantum mechanics.* Encyclopedia of mathematics and its applications, v. 15.; Encyclopedia of mathematics and its applications., Section, Mathematics of physics. Addison-Wesley Advanced Book Program, 1981.

Paolo Bouquet, Chiara Ghidini, Fausto Giunchiglia, and Enrico Blanzieri. Theories and uses of context in knowledge representation and reasoning. *Journal of Pragmatics*, 35(3):455 – 484, 2003. ISSN 0378-2166. DOI: http://dx.doi.org/10.1016/S0378-2166(02)00145-5. URL http://www.sciencedirect.com/science/article/pii/S0378216602001455. ContextContext.

Andrew Brennan. Necessary and sufficient conditions. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2012 edition, 2012.

Patrick Brézillon, Laurent Pasquier, and Jean-Charles Pomerol. Reasoning with contextual graphs. *European Journal of Operational Research*, 136(2):290 – 298, 2002. ISSN 0377-2217. DOI: http://dx.doi.org/10.1016/S0377-2217(01)00116-3. URL http://www.sciencedirect.com/science/article/pii/S0377221701001163. Human Centered Processes.

Rodney Allen Brooks. *Cambrian intelligence: the early history of the new AI*, volume 1. MIT press Cambridge, MA, 1999.

Michele Burigo and Kenny Coventry. Context affects scale selection for proximity terms. *Spatial Cognition & Computation*, 10(4):292–312, 2010. DOI: 10.1080/13875861003797719. URL http://dx.doi.org/10.1080/13875861003797719.

Jerome R. Busemeyer and Peter D. Bruza. *Quantum models of cognition and decision*. Cambridge University Press, 2012.

Jerome R. Busemeyer, Emmanuel M. Pothos, Riccardo Franco, and Jennifer S. Trueblood. A quantum theoretical explanation for probability judgment errors. *Psychological review*, 118(2):193–218, April 2011. DOI: 10.1037/a0022542.

Guoray Cai. Contextualization of geospatial database semantics for human–gis interaction. *GeoInformatica*, 11(2):217–237, 2007. ISSN 1573-7624. DOI: 10.1007/s10707-006-0001-0. URL http://dx.doi.org/10.1007/s10707-006-0001-0.

Susan Carey. *Conceptual change in childhood*. MIT press, 1985.

Laura A. Carlson and Eric S. Covey. How far is near? inferring distance from spatial descriptions. *Language and Cognitive Processes*, 20 (5):617–631, 2005. DOI: 10.1080/01690960400023501. URL http://dx.doi.org/10.1080/01690960400023501.

Daniel Chandler. *Semiotics: the basics*. Routledge, 2007.

Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not.*, 46(4):53–64, May 2011. ISSN 0362-1340. DOI: 10.1145/1988042.1988046. URL http://doi.acm.org/10.1145/1988042.1988046.

Collins Cobuild. *English language dictionary*. 1987. URL http://www.collinsdictionary.com/dictionary/english/context.

Paul Crease. *Representation of geographic relevance in mobile applications*. PhD thesis, University Zürich, 2013.

Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax, February 2014. URL https://www.w3.org/TR/rdf11-concepts/.

Murray E. Danofsky. How near is near? *AI Memo No. 344, Artificial Intelligence Lab, MIT*, 1976. URL http://hdl.handle.net/1721.1/6245.

Ferdinand de Saussure. *Cours de linguistique generale*. Paris: Payot, 1916.

Curdin Derungs and Ross S. Purves. Where's near? using web n-grams to explore spatial relations. *GIScience 2014, 8th International Conference on Geographic Information Science, September 23-26, 2014, Vienna, Austria*, 2014.

Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, Januar 2001. ISSN 1617-4909. DOI: 10.1007/s007790170019. URL http://dx.doi.org/10.1007/s007790170019.

Pasquale di Donato. Geospatial semantics: A critical review. In David Taniar, Osvaldo Gervasi, Beniamino Murgante, Eric Pardede, and BernadyO. Apduhan, editors, *Computational Science and Its Applications – ICCSA 2010*, volume 6016 of *Lecture Notes in Computer Science*, pages 528–544. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12155-5. DOI: 10.1007/978-3-642-12156-2_40. URL http://dx.doi.org/10.1007/978-3-642-12156-2_40.

Belko Abdoul Aziz Diallo, Thierry Badard, Frédéric Hubert, and Sylvie Daniel. An owl-based mobile geobi context ontology enabling location-based and context-based reasoning and supporting contextual business analysis. *International Journal of Geosciences*, 6(1): 88–108, 2015. DOI: 10.4236/ijg.2015.61007.

Paul AM Dirac. A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 35:416–418, 7 1939. ISSN 1469-8064. DOI: 10.1017/S0305004100021162. URL http://journals.cambridge.org/article_S0305004100021162.

Matt Duckham and Michael F. Worboys. Computational structure in three-valued nearness relations. In Daniel R. Montello, editor, *Spatial Information Theory*, volume 2205 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42613-4. DOI: 10.1007/3-540-45424-1_6. URL http://dx.doi.org/10.1007/3-540-45424-1_6.

Hermann Ebbinghaus. *Über das Gedächtnis: Untersuchungen zur experimentellen Psychologie*. Duncker & Humblot, Leipzig 1885.

Umberto Eco. *A theory of semiotics*, volume 217. Indiana University Press, 1976.

Max J. Egenhofer and David M. Mark. Naive geography. In Andrew U. Frank and Werner Kuhn, editors, *Spatial Information Theory A Theoretical Basis for GIS*, volume 988 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 1995.

ISBN 978-3-540-60392-4. DOI: 10.1007/3-540-60392-1_1. URL http://dx.doi.org/10.1007/3-540-60392-1_1.

Max J Egenhofer, Keith C Clarke, Song Gao, Teriitutea Quesnot, W Randolph Franklin, May Yuan, and David Coleman. Contributions of giscience over the past twenty years. *Advancing Geographic Information Science: The Past and Next Twenty Years*, pages 9–34, 2015.

Gilles Fauconnier. *Mental spaces: Aspects of meaning construction in natural language*. Cambridge University Press, 1994.

Christiane Fellbaum, editor. *WordNet: an electronitc lexical database*. MIT Press, second printing, 1999 edition, 1998. ISBN 0-262-06197-X.

Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 407–416, New York, NY, USA, 2000. ACM. ISBN 1-58113-206-9. DOI: 10.1145/337180.337228. URL http://doi.acm.org/10.1145/337180.337228.

Peter F. Fisher and Thomas M. Orf. An investigation of the meaning of near and close on a university campus. *Computers, Environment and Urban Systems*, 15(1–2):23 – 35, 1991. ISSN 0198-9715. DOI: http://dx.doi.org/10.1016/0198-9715(91)90043-D. URL http://www.sciencedirect.com/science/article/pii/019897159190043D.

Jerry Fodor. Concepts: a potboiler. *Cognition*, 50(1–3):95 – 113, 1994. ISSN 0010-0277. DOI: http://dx.doi.org/10.1016/0010-0277(94)90023-X. URL http://www.sciencedirect.com/science/article/pii/001002779490023X.

Andrew U. Frank. Qualitative spatial reasoning about distances and directions in geographic space. *Journal of Visual Languages & Computing*, 3(4):343 – 371, 1992. ISSN 1045-926X. DOI: http://dx.doi.org/10.1016/1045-926X(92)90007-9. URL http://www.sciencedirect.com/science/article/pii/1045926X92900079.

Andrew U. Frank. Distinctions produce a taxonomic lattice: Are these the units of mentalese? In eds. B. Bennett, C. Fellbaum, editor, *Formal Ontology in Information Systems*, volume FAIA Vol 150, pages 27–38. IOS Press, November 2006.

Christian Freksa and Thomas Barkowsky. On the relation between spatial concepts and geographic objects. *Geographic objects with indeterminate boundaries*, pages 109–121, 1996.

Liane Gabora. *Cognitive mechanisms underlying the origin and evolution of culture*. PhD thesis, Brussels: Center Leo Apostel for Interdisciplinary Studies, Vrije Universiteit Brussels, 2001.

Liane Gabora, Eleanor Rosch, and Diederik Aerts. Toward an ecological theory of concepts. *Ecological Psychology*, 20(1):84–116, 2008. DOI: 10.1080/10407410701766676. URL http://dx.doi.org/10.1080/10407410701766676.

Mark Gahegan. Proximity operators for qualitative spatial reasoning. In Andrew U. Frank and Werner Kuhn, editors, *Spatial Information Theory A Theoretical Basis for GIS*, volume 988 of *Lecture Notes in Computer Science*, pages 31–44. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-60392-4. DOI: 10.1007/3-540-60392-1_3. URL http://dx.doi.org/10.1007/3-540-60392-1_3.

Peter Gärdenfors. *Conceptual spaces: The geometry of thought*. MIT press, 2004.

Judith Gelernter and Shilpa Balaji. An algorithm for local geoparsing of microtext. *GeoInformatica*, 17(4):635–667, 2013. ISSN 1573-7624. DOI: 10.1007/s10707-012-0173-8. URL http://dx.doi.org/10.1007/s10707-012-0173-8.

Arthur Gill. *Applied algebra for the computer sciences*. Prentice-Hall, 1976.

Fausto Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, 16:345–364, 1993.

Michael F. Goodchild. Keynote address: spatial information science. In *Fourth International Symposium on Spatial Data Handling (Zurich, July 1990)*, volume 1, pages 3–14, 1990.

Alison Gopnik and Andrew N. Meltzoff. *Words, thoughts, and theories*, volume 1. Mit Press Cambridge, MA, 1997.

Ramanathan V. Guha. *Contexts: a formalization and some applications*. PhD thesis, Stanford University Stanford, CA, 1991.

Juergen Hahn and Andrew U. Frank. Select the appropriate map depending on context in a hilbert space model (scop). In Harald Atmanspacher, Emmanuel Haven, Kirsty Kitto, and Derek Raine, editors, *Quantum Interaction*, volume 8369 of *Lecture Notes in Computer Science*, pages 122–133. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-54942-7. DOI: 10.1007/978-3-642-54943-4_11. URL http://dx.doi.org/10.1007/978-3-642-54943-4_11.

Juergen Hahn and Paul Weiser. A quantum formalization for communication coordination problems. In Harald Atmanspacher, Claudia Bergomi, Thomas Filk, and Kirsty Kitto, editors, *Quantum Interaction*, volume 8951 of *Lecture Notes in Computer Science*, pages 177–188. Springer International Publishing, 2015. ISBN 978-3-319-15930-0. DOI: 10.1007/978-3-319-15931-7_14. URL http://dx.doi.org/10.1007/978-3-319-15931-7_14.

Juergen Hahn, Paolo Fogliaroni, Andrew U. Frank, and Gerhard Navratil. A computational model for context and spatial concepts.

In Tapani Sarjakoski, Yasmina Maribel Santos, and Tiina L. Sarjakoski, editors, *Geospatial Data in a Changing World: Selected papers of the 19th AGILE Conference on Geographic Information Science*, pages 3–19. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33783-8. DOI: 10.1007/978-3-319-33783-8_1. URL http://dx.doi.org/10.1007/978-3-319-33783-8_1.

James A. Hampton. Conceptual combination: Conjunction and negation of natural concepts. *Memory & Cognition*, 25(6):888–909, 1997. ISSN 1532-5946. DOI: 10.3758/BF03211333. URL http://dx.doi.org/10.3758/BF03211333.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1):335–346, 1990.

Albert Held, Sven Buchholz, and Alexander Schill. Modeling of context information for pervasive computing applications. In *Procceding of the World Multiconference on Systemics, Cybernetics and Informatics*, 2002.

Douglas L. Hintzman and Genevieve Ludlam. Differential forgetting of prototypes and old instances: Simulation by an exemplar-based classification model. *Memory & Cognition*, 8(4):378–382, 1980. ISSN 0090-502X. DOI: 10.3758/BF03198278. URL http://dx.doi.org/10.3758/BF03198278.

Heidelinde Hobel, Paolo Fogliaroni, and Andrew U. Frank. *Deriving the Geographic Footprint of Cognitive Regions*, pages 67–84. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33783-8. DOI: 10.1007/978-3-319-33783-8_5. URL http://dx.doi.org/10.1007/978-3-319-33783-8_5.

Robin M Hogarth and Hillel J Einhorn. Order effects in belief updating: The belief-adjustment model. *Cognitive Psychology*, 24(1):1 – 55, 1992. ISSN 0010-0285. DOI: http://dx.doi.org/10.1016/0010-0285(92)90002-J. URL http://www.sciencedirect.com/science/article/pii/001002859290002J.

Blake Stephen Howald and E. Graham Katz. *On the Explicit and Implicit Spatiotemporal Architecture of Narratives of Personal Experience*, pages 434–454. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-23196-4. DOI: 10.1007/978-3-642-23196-4_23. URL http://dx.doi.org/10.1007/978-3-642-23196-4_23.

Richard I.G. Hughes. *The structure and interpretation of quantum mechanics*. Harvard university press, 1992.

Christoph Hölscher, Thora Tenbrink, and Jan M. Wiener. Would you follow your own route description? cognitive strategies in urban route planning. *Cognition*, 121(2):228 – 247, 2011. ISSN 0010-0277. DOI: http://dx.doi.org/10.1016/j.cognition.2011.06.005. URL http://www.sciencedirect.com/science/article/pii/S0010027711001521.

Krzysztof Janowicz. Kinds of contexts and their impact on semantic similarity measurement. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 441–446, March 2008. DOI: 10.1109/PERCOM.2008.35.

Krzysztof Janowicz, Benjamin Adams, and Martin Raubal. Semantic referencing – determining context weights for similarity measurement. In SaraIrina Fabrikant, Tumasch Reichenbacher, Marc van Kreveld, and Christoph Schlieder, editors, *Geographic Information Science*, volume 6292 of *Lecture Notes in Computer Science*, pages 70–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15299-3. DOI: 10.1007/978-3-642-15300-6_6. URL http://dx.doi.org/10.1007/978-3-642-15300-6_6.

Krzysztof Janowicz, Martin Raubal, and Werner Kuhn. The semantics of similarity in geographic information retrieval. *Journal of Spatial Information Science*, 2011(2):29–57, 2011.

Mark P. Jones. Computing with lattices: An application of type classes. *Journal of Functional Programming*, 2:475–503, 10 1992. ISSN 1469-7653. DOI: 10.1017/S0956796800000514. URL http://journals.cambridge.org/article_S0956796800000514.

Carsten Keßler. Conceptual spaces for data descriptions. In *The cognitive approach to modeling environments (CAME), workshop at GIScience*, pages 29–35, 2006.

Carsten Keßler, Martin Raubal, and Krzysztof Janowicz. *The Effect of Context on Semantic Similarity Measurement*, chapter On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops: OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, OTM Academy Doctoral Consortium, MONET, OnToContent, ORM, PerSys, PPN, RDDS, SSWS, and SWWS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II, pages 1274–1284. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-76890-6. DOI: 10.1007/978-3-540-76890-6_55. URL http://dx.doi.org/10.1007/978-3-540-76890-6_55.

Andrei Khrennivov. Classical and quantum mechanics on information spaces with applications to cognitive, psychological, social, and anomalous phenomena. *Foundations of Physics*, 29(7):1065–1098, 1999. ISSN 0015-9018. DOI: 10.1023/A:1018885632116. URL http://dx.doi.org/10.1023/A3A1018885632116.

Kirsty Kitto. Why quantum theory? In Peter D. Bruza, W. Lawless, C. van Rjisbergen, D. Sofge, B. Coecke, and S. Clark, editors, *Proceedings Second Quantum Interaction Symposium*, pages 11–18. College Publications, 2008.

Andrej Kolmogoroff. *Grundbegriffe der wahrscheinlichkeitsrechnung*, volume 3. Berlin, 1933.

Lloyd K. Komatsu. Recent views of conceptual structure. *Psychological bulletin*, 112(3):500, 1992.

Werner Kuhn. Geospatial semantics: Why, of what, and how? In Stefano Spaccapietra and Esteban Zimányi, editors, *Journal on Data Semantics III*, volume 3534 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26225-1. DOI: 10.1007/11496168_1. URL http://dx.doi.org/10.1007/11496168_1.

George Lakoff and Mark Johnson. *Metaphors we live by*. University of Chicago press, 2008.

Stephen Laurence and Eric Margolis. Concepts and cognitive science. *Concepts: core readings*, pages 3–81, 1999.

Derek F. Lawden. *The Mathematical Principles of Quantum Mechanics*. Dover Publications, 2005.

G. N. Leech. *Semantics: The Study of Meaning*. Penguin: Harmondsworth, 1981.

Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

Yuri Lin, Jean-Baptiste Michel, Erez Lieberman Aiden, Jon Orwant, Will Brockman, and Slav Petrov. Syntactic annotations for the google books ngram corpus. In *Proceedings of the ACL 2012 System Demonstrations*, ACL '12, pages 169–174, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=2390470.2390499.

Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. *Specification of abstract data types*. John Wiley & Sons, Inc., 1997.

William Loyola. Comparison of approaches toward formalising context: Implementation characteristics and capacities. *Electronic Journal of Knowledge Management*, 5(2):203–214, 2007.

S Mac Lane and G Birkhoff. *Algebra Third Edition. Providence, Rhode Island*. AMS Chelsea Publishing, 1991.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-5010.

Eric Margolis and Stephen Laurence. Concepts. http://plato.stanford.edu/archives/spr2014/entries/concepts/, 2014.

David M. Mark. Toward a theoretical framework for geographic entity types. In Andrew U. Frank and Irene Campari, editors, *Spatial Information Theory A Theoretical Basis for GIS*, volume 716 of *Lecture*

Notes in Computer Science, pages 270–283. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-57207-7. DOI: 10.1007/3-540-57207-4_18. URL http://dx.doi.org/10.1007/3-540-57207-4_18.

David M. Mark, Christian Freksa, Stephen C. Hirtle, Robert Lloyd, and Barbara Tversky. Cognitive models of geographical space. *International Journal of Geographical Information Science*, 13(8):747–774, 1999. DOI: 10.1080/136588199241003. URL http://dx.doi.org/10.1080/136588199241003.

Simon Marlow. Haskell 2010 language report, 2010.

John McCarthy. Generality in artificial intelligence. *Commun. ACM*, 30(12):1030–1035, December 1987. ISSN 0001-0782. DOI: 10.1145/33447.33448. URL http://doi.acm.org/10.1145/33447.33448.

John McCarthy. Notes on formalizing context, 1993. URL http://cogprints.org/418/.

John McCarthy. A logical ai approach to context. *Unpublished note*, 6, 1996.

John McCarthy and Sasa Buvac. Formalizing context (expanded notes), 1997. URL http://cogprints.org/419/.

Grant McKenzie and Martin Raubal. Adding social constraints to location based services. extended abstract. In *8th Symposium on Location-Based Services, Vienna, Austria.*, 2011.

David E. Meyer and Roger W. Schvaneveldt. Facilitation in recognizing pairs of words: evidence of a dependence between retrieval operations. *Journal of experimental psychology*, 90(2):227, 1971.

John P. Minda and J. David Smith. Prototype models of categorization: Basic formulation, predictions, and limitations. *Formal approaches in categorization*, pages 40–64, 2011.

Daniel R. Montello, Michael F. Goodchild, Jonathon Gottsegen, and Peter Fohl. Where's downtown?: Behavioral methods for determining referents of vague spatial queries. *Spatial Cognition & Computation*, 3(2-3):185–204, 2003. DOI: 10.1080/13875868.2003.9683761. URL http://dx.doi.org/10.1080/13875868.2003.9683761.

Daniel R. Montello, Alinda Friedman, and Daniel W. Phillips. Vague cognitive regions in geography and geographic information science. *International Journal of Geographical Information Science*, 28(9):1802–1820, 2014. DOI: 10.1080/13658816.2014.900178. URL http://dx.doi.org/10.1080/13658816.2014.900178.

David W. Moore. Measuring new types of question-order effects: Additive and subtractive. *The Public Opinion Quarterly*, 66(1):80–91, 2002. ISSN 0033362X, 15375331. URL http://www.jstor.org/stable/3078697.

Daniel G. Morrow and Herbert H. Clark. Interpreting words in spatial descriptions. *Language and Cognitive Processes*, 3(4):275–291, 1988. DOI: 10.1080/01690968808402091. URL http://dx.doi.org/10.1080/01690968808402091.

Robert M. Nosofsky. Attention, similarity, and the identification–categorization relationship. *Journal of experimental psychology: General*, 115(1):39, 1986.

Robert M. Nosofsky. The generalized context model: An exemplar model of classification. *Formal approaches in categorization*, pages 18–39, 2011.

Charles K. Ogden and Ivor A. Richards. *The meaning of meaning*. Harcourt, Brace & World New York, 8th edition edition, 1946.

Kieron O'Hara, Mischa M. Tuffield, and Nigel Shadbolt. Lifelogging: Privacy and empowerment with memories for life. *Identity in the Information Society*, 1(1):155–172, 2009. ISSN 1876-0678. DOI: 10.1007/s12394-009-0008-4. URL http://dx.doi.org/10.1007/s12394-009-0008-4.

Daniel N. Osherson and Edward E. Smith. On the adequacy of prototype theory as a theory of concepts. *Cognition*, 9(1):35 – 58, 1981. ISSN 0010-0277. DOI: http://dx.doi.org/10.1016/0010-0277(81)90013-5. URL http://www.sciencedirect.com/science/article/pii/0010027781900135.

Wolfgang Pauli. Zur quantenmechanik des magnetischen elektrons. *Zeitschrift für Physik*, 43(9):601–623, 1927. ISSN 0044-3328. DOI: 10.1007/BF01397326. URL http://dx.doi.org/10.1007/BF01397326.

Charles S. Peirce. *Collected papers of Charles Sanders Peirce*, volume 5. Harvard University Press, 1931.

Asher Peres. *Quantum Theory: Concepts and Methods*, volume 72 of *Fundamental Theories of Physics*. Springer, 1934. ISBN 0-7923-2549-4.

Emmanuel M. Pothos and Jerome R. Busemeyer. Can quantum probability provide a new direction for cognitive modeling? *Behavioral and Brain Sciences*, 36:255–274, 6 2013. ISSN 1469-1825. DOI: 10.1017/S0140525X12001525. URL http://journals.cambridge.org/article_S0140525X12001525.

Martin Raubal. Formalizing conceptual spaces. In A. Varzi and L. Vieu, editors, *Formal ontology in information systems, proceedings of the third international conference (FOIS 2004). Frontiers in Artificial Intelligence and Applications 114*, volume 114, pages 153–164. IOS Press, Amsterdam, NL, 2004.

Martin Raubal. *Representing Concepts in Time*, pages 328–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-87601-4. DOI: 10.1007/978-3-540-87601-4_24. URL http://dx.doi.org/10.1007/978-3-540-87601-4_24.

Martin Raubal and Ilija Panov. *A Formal Model for Mobile Map Adaptation*, pages 11–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-87393-8. DOI: 10.1007/978-3-540-87393-8_2. URL http://dx.doi.org/10.1007/978-3-540-87393-8_2.

Vincent B. Robinson. Interactive machine acquisition of a fuzzy spatial relation. *Computers & Geosciences*, 16(6):857 – 872, 1990. ISSN 0098-3004. DOI: 10.1016/0098-3004(90)90008-H. URL http://www.sciencedirect.com/science/article/pii/009830049090008H.

Vincent B. Robinson. Individual and multipersonal fuzzy spatial relations acquired using human–machine interaction. *Fuzzy Sets and Systems*, 113(1):133–145, 2000. ISSN 0165-0114. DOI: http://dx.doi.org/10.1016/S0165-0114(99)00017-2. URL http://www.sciencedirect.com/science/article/pii/S0165011499000172.

M. Andrea Rodríguez and Max J. Egenhofer. Putting similarity assessments into context: Matching functions with the user's intended operations. In Paolo Bouquet, Massimo Benerecetti, Luciano Serafini, Patrick Brézillon, and Francesca Castellani, editors, *Modeling and Using Context*, volume 1688 of *Lecture Notes in Computer Science*, pages 310–323. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66432-1. DOI: 10.1007/3-540-48315-2_24. URL http://dx.doi.org/10.1007/3-540-48315-2_24.

M. Andrea Rodríguez and Max J. Egenhofer. Comparing geospatial entity classes: an asymmetric and context-dependent similarity measure. *International Journal of Geographical Information Science*, 18(3):229–256, 2004. DOI: 10.1080/13658810310001629592. URL http://dx.doi.org/10.1080/13658810310001629592.

M. Andrea Rodríguez, Max J. Egenhofer, and Robert D. Rugg. Assessing semantic similarities among geospatial feature class definitions. In Andrej Včkovski, KurtE. Brassel, and Hans-Jörg Schek, editors, *Interoperating Geographic Information Systems*, volume 1580 of *Lecture Notes in Computer Science*, pages 189–202. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-65725-5. DOI: 10.1007/10703121_16. URL http://dx.doi.org/10.1007/10703121_16.

Eleanor Rosch and Carolyn B. Mervis. Family resemblances: Studies in the internal structure of categories. *Cognitive psychology*, 7(4): 573–605, 1975.

Robert E. Sanders. Dan sperber and deirdre wilson, relevance: Communication and cognition, oxford: Basil blackwell, 1986. pp. 265.

Language in Society, 17:604–609, 12 1988. ISSN 1469-8013. DOI: 10.1017/S004740450001318X. URL http://journals.cambridge.org/article_S004740450001318X.

Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90, Dec 1994. DOI: 10.1109/WMCSA.1994.16.

Erwin Schrödinger. Die gegenwärtige situation in der quantenmechanik. *Naturwissenschaften*, 23(49):823–828, 1935.

Angela Schwering and Werner Kuhn. A hybrid semantic similarity measure for spatial information retrieval. *Spatial Cognition & Computation*, 9(1):30–63, 2009. DOI: 10.1080/13875860802645087. URL http://dx.doi.org/10.1080/13875860802645087.

Angela Schwering and Martin Raubal. Measuring semantic similarity between geospatial conceptual regions. In M. Andrea Rodríguez, Isabel Cruz, Sergei Levashkin, and Max J. Egenhofer, editors, *GeoSpatial Semantics*, volume 3799 of *Lecture Notes in Computer Science*, pages 90–106. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-30288-9. DOI: 10.1007/11586180_7. URL http://dx.doi.org/10.1007/11586180_7.

John R. Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(03):417–424, 1980.

John R. Searle. *The construction of social reality*. Simon and Schuster, 1995.

John R. Searle. *The MIT encyclopedia of the cognitive sciences*, chapter The Chinese Room, pages 115–116. MIT press, 1999.

Yoav Shoham. Varieties of context. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, pages 393–408, 1991.

Robert S. Siegler. Microgenetic studies of self-explanation. *Microdevelopment: Transition processes in development and learning*, pages 31–58, 2002.

Bernard W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

Barry Smith and David M. Mark. Ontology and geographic kinds. *Proceedings, International Symposium on Spatial Data Handling, Vancouver, Canada.*, 1998.

Edward E. Smith and Douglas L. Medin. *Categories and concepts*. Harvard University Press Cambridge, MA, 1981.

Stanley S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):pp. 677–680, 1946. ISSN 00368075. URL http://www.jstor.org/stable/1671815.

Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Applications of a context ontology language. In *Proceedings of SoftCOM 2003, 11th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2003)*, pages 14–18, 2003. URL http://elib.dlr.de/7326/.

J. S. Trueblood and Jerome R. Busemeyer. A comparison of the belief-adjustment model and the quantum inference model as explanations of order effects in human inference. In In S. Ohlsson & R. Catrambone (Eds.), editor, *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 1166–1171, 2010.

Amos Tversky and Itamar Gati. Similarity, separability, and the triangle inequality. *Psychological review*, 89(2):123, 1982.

Florian A. Twaroch and Andrew U. Frank. Sandbox geography — to learn from children the form of spatial concepts. In *Developments in Spatial Data Handling*, pages 421–433. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-22610-9. DOI: 10.1007/3-540-26772-7_32. URL http://dx.doi.org/10.1007/3-540-26772-7_32.

Emile van der Zee, Karen Adams, and Jussi Niemi. The influence of geometrical and nongeometrical features on the use of the lexical concepts near and far in english and finnish. *Spatial Cognition & Computation*, 9(4):305–317, 2009. DOI: 10.1080/13875860903219212. URL http://dx.doi.org/10.1080/13875860903219212.

Francisco J. Varela, Evan Thompson, and Eleanor Rosch. *The embodied mind; cognitive science and human experience.* MIT Press, Cambridge, Mass. [u.a.], 6. print. edition, 1997. ISBN 0-262-22042-3; 0-262-72021-3.

John von Neumann. *Mathematische Grundlagen der Quantenmechanik.* Number 38. Berlin Springer, 1932.

Jan Oliver Wallgrün, Alexander Klippel, and Timothy Baldwin. Building a corpus of spatial relational expressions extracted from web documents. In *Proceedings of the 8th Workshop on Geographic Information Retrieval*, GIR '14, page 6, New York, NY, USA, 2014. ACM, ACM. ISBN 978-1-4503-3135-7. DOI: 10.1145/2675354.2675702. URL http://doi.acm.org/10.1145/2675354.2675702.

Fangju Wang. Towards a natural language user interface: an approach of fuzzy query. *International Journal of Geographical Information Systems*, 8(2):143–162, 1994. DOI: 10.1080/02693799408901991. URL http://dx.doi.org/10.1080/02693799408901991.

Zheng Wang, Jerome R. Busemeyer, Harald Atmanspacher, and Emmanuel M. Pothos. The potential of using quantum theory to build models of cognition. *Topics in Cognitive Science*, 5(4):672–688, 2013a. ISSN 1756-8765. DOI: 10.1111/tops.12043. URL http://dx.doi.org/10.1111/tops.12043.

Zheng Wang, Tyler Solloway, and Jerome R. Busemeyer. New empirical tests of a quantum model for question order effects. In *35th Annual Cognitive Science Conference*, 2013b.

Mark Weiser. The computer for the 21st century. *Scientific American*, 9(3):94–104, 1991. DOI: 10.1038/scientificamerican0991-94.

Mark Weiser. Hot topics-ubiquitous computing. *Computer*, 26(10): 71–72, Oct 1993. ISSN 0018-9162. DOI: 10.1109/2.237456.

Paul Weiser. *A pragmatic communication model for way-finding instructions*. PhD thesis, Technische Universität Wien, 2014.

Paul Weiser and Andrew U. Frank. Cognitive transactions - a communication model. In Thora Tenbrink, John Stell, Antony Galton, and Zena Wood, editors, *Spatial Information Theory*, volume 8116 of *Lecture Notes in Computer Science*, pages 129–148. Springer International Publishing, 2013. ISBN 978-3-319-01789-1. DOI: 10.1007/978-3-319-01790-7_8. URL http://dx.doi.org/10.1007/978-3-319-01790-7_8.

Anna Wierzbicka. *Semantics: Primes and Universals: Primes and Universals*. Oxford University Press, UK, 1996.

Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors, *Formal Concept Analysis*, volume 3626 of *Lecture Notes in Computer Science*, pages 1–33. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-27891-7. DOI: 10.1007/11528784_1. URL http://dx.doi.org/10.1007/11528784_1.

Ludwig Wittgenstein. *Philosophical investigations*. Oxford: Blackwell, 1953.

Michael F. Worboys. Nearness relations in environmental space. *International Journal of Geographical Information Science*, 15(7): 633–651, 2001. DOI: 10.1080/13658810110061162. URL http://dx.doi.org/10.1080/13658810110061162.

Michael F. Worboys, Matt Duckham, and Lars Kulik. Commonsense notions of proximity and direction in environmental space. *Spatial Cognition & Computation*, 4(4):285–312, 2004. DOI: 10.1207/s15427633scc0404_1. URL http://dx.doi.org/10.1207/s15427633scc0404_1.

Sen Xu, Alexander Klippel, Alan M. MacEachren, and Prasenjit Mitra. Exploring regional variation in spatial language using spatially stratified web-sampled route direction documents. *Spatial Cognition & Computation*, 14(4):255–283, 2014. DOI: 10.1080/13875868.2014.943904. URL http://dx.doi.org/10.1080/13875868.2014.943904.

Xiaobai Yao and Jean-Claude Thill. Neurofuzzy modeling of context–contingent proximity relations. *Geographical Analysis*, 39(2):169–194, 2007. ISSN 1538-4632. DOI: 10.1111/j.1538-4632.2007.00700.x. URL http://dx.doi.org/10.1111/j.1538-4632.2007.00700.x.

Lotfi A. Zadeh. A fuzzy-algorithmic approach to the definition of complex or imprecise concepts. In *Systems Theory in the Social Sciences*, Interdisciplinary Systems Research / Interdisziplinäre Systemforschung, pages 202–282. Birkhäuser Basel, 1976. ISBN 978-3-7643-0822-3. DOI: 10.1007/978-3-0348-5495-5_11. URL http://dx.doi.org/10.1007/978-3-0348-5495-5_11.

Xiao Zhang, Prasenjit Mitra, Alexander Klippel, and Alan MacEachren. *Automatic Extraction of Destinations, Origins and Route Parts from Human Generated Route Directions*, pages 279–294. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15300-6. DOI: 10.1007/978-3-642-15300-6_20. URL http://dx.doi.org/10.1007/978-3-642-15300-6_20.

Index

A
Haskell Code for Context Algebra

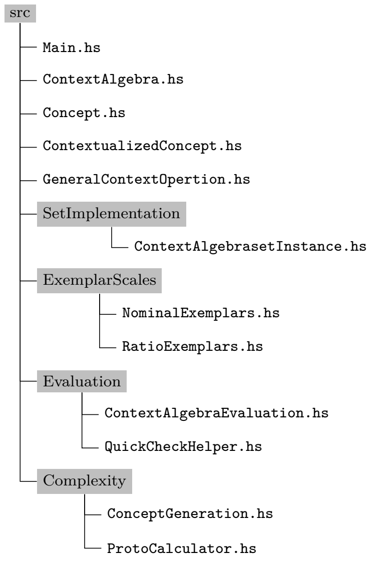


Figure A.1: Directory structure of the Haskell source code

This chapter includes source code and measurement results for the context algebra. The Haskell modules implementing context algebra are included in Section A.1. This implementation is used to produce the general context operation and demonstrate it's usage by example in Section A.2. The source code to evaluate the context algebra is given in Section A.3. In the last Section A.4 the source code necessary for the complexity analysis and the measurement results are included.

The functionality of the source code is split into thematically coherent directories illustrated in Figure A.1. The `Main` module and the abstract classes `ContextAlgebra`, `Concept`, `ContextualizedConcept` and the `GeneralContextOperation` are placed in `src` directory. The specific instances for the abstract classes *e. g.*`ContextAlgebraInstance`, `NominalExemplar` and `RatioExemplars` are stored in according sub directories `SetImplementation` and `ExemplarScales`. The necessary code for evaluation and complexity analysis is placed in a separated directory `Evaluation` and `Complexity`.

The source code is aligned to fit to the line size of this LaTeXtemplate which sometimes clashes with the Haskell formatting strategy, please take care. Modules that consume a lot lines of code are split into two or more listings fitting on one page labeled part 1 and part 2 etc.

A.1 Context Algebra

The full source code of the Haskell modules shown in Figure 6.1 are given in this section. The abstract classes defined in module `ContextAlgebra` are highlighted in red font in all algorithms.

Algorithm A.1: Module ContextAlgebra including abstract classes

```
module ContextAlgebra where

import Test.QuickCheck

-- * Classes that build the ContextAlgebra
class PartialOrder c where
  -- | partial order relation
  isMoreSelectiveOrEqualSel :: c    -- ^ first input context
                            -> c    -- ^ second input context
                            -> Bool -- ^ true if the first context is more selective or
                                    --   equal selective compared to the second context

  -- | partial order relation
  isLessSelectiveOrEqualSel :: c    -- ^ first input context
                            -> c    -- ^ second input context
                            -> Bool -- ^ true if the first context is less selective or
                                    --   equal selective compared to the second context

  -- | equal selective relation
  equals :: c    -- ^ first input context
         -> c    -- ^ second input context
         -> Bool -- ^ true if the two input context are equal
  equals c1 c2 = c1 `isMoreSelectiveOrEqualSel` c2
                 &&
                 c1 `isLessSelectiveOrEqualSel` c2

class Universe c where
  -- | includes all atomic contexts
  atomicRepresentation :: [c]

class  Bound c where
  -- | response the universal least context
  leastContext :: c
  -- | response the universal greatest context
  greatestContext :: c

class (PartialOrder c, Bound c, Complement c) => ContextLattice c where
  -- | conjunction of two context
  conjunction :: (Bound c, Complement c, PartialOrder c)=>
                 c -- ^ first input context
              -> c -- ^ second input context
              -> c -- ^ resulting in a more selective context than both constituents

  -- | disjunction of two context
  disjunction :: (Bound c, Complement c, PartialOrder c)=>
                 c -- ^ first input context
              -> c -- ^ second input context
              -> c -- ^ resulting in a less selective context than both constituents

class Complement c where
  -- | complements the input context
  complement :: c -- ^ input context
             -> c -- ^ complement of the input context
```

Algorithm A.2: Module ContextAlgebraSetInstace

```haskell
{-# LANGUAGE TypeSynonymInstances #-}
module SetImplementation.ContextAlgebraSetInstance where

import qualified Data.Set as Set

import ContextAlgebra

type Context c = Set.Set c

createContext :: c
              -> Context c
createContext  = Set.singleton

instance  (Ord c) => PartialOrder (Context c) where
  isMoreSelectiveOrEqualSel = Set.isSubsetOf

  isLessSelectiveOrEqualSel c1 c2
   | c2 == c1 = True
   | otherwise = not $ c1 `isMoreSelectiveOrEqualSel` c2

instance (Universe c, Ord c) => Bound (Context c) where
  greatestContext = Set.unions . map createContext $ atomicRepresentation

  leastContext =emptyContext
    where emptyContext = Set.empty

instance (Universe c, Ord c)=> Complement (Context c) where
  complement = Set.difference greatestContext

instance (Universe c, Complement c, Ord c)=> ContextLattice (Context c) where
  disjunction c1 c2
   | c1 `equals` leastContext = c2
   | c2 `equals` leastContext = c1
   | c1 `equals` greatestContext = greatestContext
   | c2 `equals` greatestContext = greatestContext
   | c1 `equals` c2 = c1
   | complement c1 `equals` c2 = greatestContext
   | complement c2 `equals` c1 = greatestContext
   | c1 `isMoreSelectiveOrEqualSel` c2 = c2
   | otherwise =  c1 `Set.union` c2

  conjunction c1 c2
   | c1 `equals` leastContext = leastContext
   | c2 `equals` leastContext = leastContext
   | c1 `equals` greatestContext = c2
   | c2 `equals` greatestContext = c1
   | c1 `equals` c2 = c1
   | complement c1 `equals` c2 = leastContext
   | complement c2 `equals` c1 = leastContext
   | c1 `isMoreSelectiveOrEqualSel` c2 = c1
   | otherwise = c1 `Set.intersection` c2
```

Algorithm A.3: Module Concept

```haskell
module Concept where

import qualified Data.MultiSet as Mset
import ContextAlgebra

data Observation c e = O (c,e) deriving (Show,Eq,Ord)

getContext :: Observation c e ->  c
getContext (O (c,_))= c

getExemplar :: Observation c e -> e
getExemplar (O (_,e))= e

data Concept c e= C (Mset.MultiSet (Observation c e)) deriving (Show)

createConcept :: Observation c e -> Concept c e
createConcept = C . Mset.singleton

addConcept :: (Ord c,Ord e)=>Concept c e -> Concept c e -> Concept c e
addConcept (C a) (C b) = C $ Mset.union  a  b

nullConcept :: Concept c e -> Bool
nullConcept (C e) = Mset.null e

emptyConcept :: Concept c e
emptyConcept = C Mset.empty

unionConcept:: (Ord c,Ord e)=>[Concept c e] -> Concept c e
unionConcept = foldl addConcept emptyConcept

toObservationList :: Concept c e -> [Observation c e]
toObservationList (C e) = Mset.toList e

fromObservationList :: (Ord c,Ord e) => [Observation c e] -> Concept c e
fromObservationList = C . Mset.fromList

toOccurrenceList :: Concept c e -> [(Observation c e ,Int)]
toOccurrenceList (C obs)= Mset.toOccurList obs

numberExemplars :: Concept c e -> Int
numberExemplars (C obs) = Mset.size obs
```

Algorithm A.4: Module Mapping

```haskell
{-# LANGUAGE FlexibleContexts, FlexibleInstances, MultiParamTypeClasses,
             TypeSynonymInstances #-}
module Mapping where

import ContextAlgebra
import Concept

class Mapping c e where -- data type e equals this of the lattice data type
  m :: (Ord c, Ord e, Bound c, PartialOrder c) =>
       c            -- ^ more selective context or equal selective context
    -> Concept c e  -- ^ knowledge base including facts influenced by context
    -> Concept c e  -- ^ knowledge base including facts valid for context c

instance Mapping c (Concept c e) where
 m ctx concept
   | leastContext `equals` ctx = emptyConcept
   | otherwise =  fromObservationList
                   [obs | obs <- toObservationList concept
                        , getContext obs `isMoreSelectiveOrEqualSel` ctx]
```

Algorithm A.5: Module ContextualizedConcept

```
{-# LANGUAGE MultiParamTypeClasses,FlexibleInstances #-}
module ContextualizedConcept where

import Concept

class ContextualizedConcept c e where
  calculatePrototype :: (Show e, Ord e) =>
                        Concept c e -- ^ observations representing the
                                    --   contextualized concept
                     -> (e,Double)  -- ^ calculated prototype for the
                                    --   contextualized concept,
                                    --   with contextual typicality
```

Algorithm A.6: Module NominalExemplars

```
{-# LANGUAGE MultiParamTypeClasses, FlexibleInstances #-}
module ExemplarScales.NominalExemplars where

import qualified Data.MultiSet as Mset
import qualified Data.List as List
import Data.Function
import ContextualizedConcept
import Concept

instance ContextualizedConcept c e where
  calculatePrototype  = List.maximumBy (compare `on` snd ) . rateObservations

rateObservations :: (Show e, Ord e) =>
                    Concept c e  -- ^ contextualized concept
                 -> [(e,Double)] -- ^ contextual typicality for all exemplars
rateObservations con = map (\(exemplar, occurences) ->
                          (exemplar,fromIntegral occurrences /
                                    fromIntegral totalOccurrences) )
                      . Mset.toAscOccurList . Mset.fromList
                      . concatMap (\(O (_,e) ,o)-> replicate o e).
                      toOccurenceList $ con
  where totalOccurrences = numberExemplars con
```

A.2 General Context Operation

Algorithm A.7: Formalization for the restaurant example using the Generalized Context Operation

```haskell
{-# LANGUAGE ScopedTypeVariables, TypeSynonymInstances,
             FlexibleInstances, MultiParamTypeClasses #-}
module GeneralContextOperation where

import ContextAlgebra
import Concept
import Mapping
import SetImplementation.ContextAlgebraSetInstance
import ContextualizedConcept
import ExemplarScales.NominalExemplars

generalContextOperation :: ( Ord c, Ord e
                           , Bound c, PartialOrder c, ContextLattice c, Mapping c e) => c
                        -> c
                        -> Concept c e
                        -> Concept c e
generalContextOperation actualCtx newCtx con = m moreSelCtx con
 where moreSelCtx = actualCtx `conjunction` newCtx

a1Name = "(in Vienna) and (serving pizza)"
a2Name = "(in Vienna) and not (serving pizza)"
a3Name = "not(in Vienna) and (serving pizza) "

a1 = createContext a1Name
a2 = createContext a2Name
a3 = createContext a3Name
inVienna = a1 `disjunction` a2
servingPizza = a1 `disjunction` a3
inViennaOrServingPizza = a2 `disjunction` a3

instance Complement String
instance Bound String
instance Universe String where
  atomicRepresentation = [a1Name
                         ,a2Name
                         ,a3Name]
instance Mapping (Context String) String where
  m ctx entity
    | leastContext `equals` ctx = emptyConcept
    | otherwise = fromObservationList
                  [obs | obs <- toObservationList entity
                       , getContext obs `isMoreSelectiveOrEqualSel` ctx]

plachutta1 = O (a1, "Plachutta")
plachuttaV = replicate 5 $ O (a2, "Plachutta")
pizzahut = replicate 11 $ O (a3, "Pizza Hut")
candinetta = replicate 3 $ O (a1, "Candinetta")
vapianoV = replicate 5 $ O (a1, "Vapiano")
vapianoP = replicate 5 $ O (a3, "Vapiano")

restaurant = unionConcept . map createConcept $ obs
 where obs =[plachutta1]++plachuttaV++pizzahut++candinetta++vapianoV++vapianoP

prototypeRestaurants = calculatePrototype restaurant
restsInVienna = generalContextOperation greatestContext inVienna restaurant
prototypeInVienna = calculatePrototype restsInVienna
restsInViennaServingPizza = generalContextOperation inVienna a1 restsInVienna
protoInViennaServingPizza = calculatePrototype restsInViennaServingPizza
```

A.3 Evaluation

The Algorithms A.8, A.9, A.10, A.12 and A.13 are part of the `Main` module.

Algorithm A.8: Property tests for Bound class

```haskell
-- * Context lattice bounds test

-- | Check if leastContext is the least context
prop_isLeastContext :: (PartialOrder c,Bound c) =>
                       c
                    -> Bool
prop_isLeastContext c= leastContext `isMoreSelectiveOrEqualSel` c

-- | Check if greatestContext is the least context
prop_isGreatestContext :: (PartialOrder c,Bound c) =>
                          c
                       -> Bool
prop_isGreatestContext c =  greatestContext `isLessSelectiveOrEqualSel` c
```

Algorithm A.9: Property tests for ContextLattice class

```haskell
-- * Combination of contexts
-- | idempotent test of two context
prop_isIdempotent :: (PartialOrder c, ContextLattice c) =>
                     c
                     -> Bool
prop_isIdempotent c =  (c `conjunction` c) `equals` c
                       &&
                       (c `disjunction` c) `equals` c

-- | commutative test  of two contexts
prop_isCommutative :: (PartialOrder c, Bound c, Complement c, ContextLattice c) =>
                ┬
```

```haskell
-- * Combination of contexts
-- | idempotent test of two context
prop_isIdempotent :: (PartialOrder c, ContextLattice c) =>
                     c
                     -> Bool
prop_isIdempotent c =  (c `conjunction` c) `equals` c
                       &&
                       (c `disjunction` c) `equals` c

-- | commutative test  of two contexts
prop_isCommutative :: (PartialOrder c, Bound c, Complement c, ContextLattice c) =>
                c           -- ^ input context
                -> c        -- ^ input context
                -> Property -- ^ true if both contexts are commutative
prop_isCommutative c1 c2 =  classify (c1 `isMoreSelectiveOrEqualSel` c2) "c1 <= c2" $
                            classify (c1 `isLessSelectiveOrEqualSel` c2) "c1 >= c2" $
                            (c1 `conjunction` c2) `equals` (c2 `conjunction` c1)
                            &&
                            (c1 `disjunction` c2) `equals` (c2 `disjunction` c1)

-- | associative test of three context
prop_isAssociative :: (PartialOrder c, Bound c, Complement c, ContextLattice c) =>
                 c           -- ^ input context
                 -> c        -- ^ input context
                 -> c        -- ^ input context
                 -> Property -- ^ True if three contexts are associative
prop_isAssociative c1 c2 c3 = classify (c1 `isMoreSelectiveOrEqualSel` c2) "c1 <= c2" $
                              classify (c1 `isLessSelectiveOrEqualSel` c2) "c1 >= c2" $
                              classify (c1 `isMoreSelectiveOrEqualSel` c3) "c1 <= c3" $
                              classify (c1 `isLessSelectiveOrEqualSel` c3) "c1 >= c3" $
                              classify (c2 `isMoreSelectiveOrEqualSel` c3) "c2 <= c3" $
                              classify (c2 `isLessSelectiveOrEqualSel` c3) "c2 >= c3" $
                              m `equals` n
                              &&
                              o `equals` p
   where m = c1 `conjunction` (c2 `conjunction` c3)
         n = (c1 `conjunction` c2) `conjunction` c3
         o = c1 `disjunction` (c2 `disjunction` c3)
         p = (c1 `disjunction` c2) `disjunction` c3

-- | consistency test of two contexts
prop_isConsistent  :: (PartialOrder c, Bound c, ContextLattice c) =>
                 c
                 -> c
                 -> Property
prop_isConsistent c1 c2 = c1 `isMoreSelectiveOrEqualSel` c2 ==>
                          (c1 `conjunction` c2 `equals` c1)
                          &&
                          (c1 `disjunction` c2 `equals` c2)

-- | isotone test of three context
prop_isIsotone :: (PartialOrder c, Bound c, Complement c, ContextLattice c) =>
             c           -- ^ input context
             -> c        -- ^ input context
             -> c        -- ^ input context
             -> Property -- ^ True if all three context are isotone
prop_isIsotone c1 c2 c3 = c2 `isMoreSelectiveOrEqualSel` c3  ==>
                          classify (c1 `isMoreSelectiveOrEqualSel` c2) "c1 <= c2" $
                          classify (c1 `isLessSelectiveOrEqualSel` c2) "c1 >= c2" $
                          classify (c1 `isMoreSelectiveOrEqualSel` c3) "c1 <= c3" $
                          classify (c1 `isLessSelectiveOrEqualSel` c3) "c1 >= c3" $
                          ((c1 `disjunction` c2) `isMoreSelectiveOrEqualSel`
                           (c1 `disjunction` c3))
                          &&
                          ((c1 `conjunction` c2) `isMoreSelectiveOrEqualSel`
                           (c1 `conjunction`c3))
```

Algorithm A.10: Property tests for Greatest and Least context

```
-- ** Combination with Bound elements

-- | properties for combining with leastContext context
prop_withCombLeastContext::(PartialOrder c, Bound c, Complement c, ContextLattice c) =>
                           c
                           -> Bool
prop_withCombLeastContext c =  c `conjunction` leastContext `equals` leastContext
                               &&
                               leastContext `disjunction` c `equals` c

-- | properties for combining with leastContext context
prop_withCombGreatestContext::(PartialOrder c, Bound c, Complement c,ContextLattice c) =>
                      c
                      -> Bool
prop_withCombGreatestContext c = c `conjunction` greatestContext `equals` c
                                 &&
                                 greatestContext `disjunction` c `equals` greatestContext
```

Algorithm A.11: Property tests for Complement class

```
-- * Complement tests

-- | test if complement of complement equals the input context
prop_isDoubleComp :: (PartialOrder c, Complement c) =>
                     c
                     -> Bool
prop_isDoubleComp c = equals c . complement .complement $ c

-- | test if the complement property holds
prop_isLessThenComp :: (PartialOrder c, Complement c) =>
                       c
                       -> c
                       -> Property
prop_isLessThenComp c1 c2=  c1 `isMoreSelectiveOrEqualSel` c2 ==>
                            complement c2 `isMoreSelectiveOrEqualSel` complement c1


-- | complement of greatestContext and leastContext context
prop_withCombBoundComp ::(Complement c,ContextLattice c) =>
                          c           -- ^ input context
                          -> Property -- ^ True if property is fulfilled
prop_withCombBoundComp c =
  classify (c `isMoreSelectiveOrEqualSel` complement c) "c <= !c" $
  classify (c `isLessSelectiveOrEqualSel` complement c) "c >= !c" $
  (c `conjunction`  complement c `equals` leastContext)
 &&
  (c `disjunction ` complement c `equals` greatestContext)
```

Algorithm A.12: Module ContextAlgebraEvaluation, part 1

```haskell
{-# LANGUAGE TypeSynonymInstances, FlexibleInstances #-}
module Evaluation.ContextAlgebraEvaluation where

import Test.QuickCheck
import ContextAlgebra
import Evaluation.QuickCheckHelper
import SetImplementation.ContextAlgebraSetInstance
import GeneralContextOperation
-- * Function to generate all possible Inputs for QuickCheck
instance Arbitrary CtxType where
  -- | input contexts for the QuickCheck tests, imported from GeneralContextOperation
  arbitrary = elements [leastContext
                       ,greatestContext
                       ,a1
                       ,a2
                       ,a3
                       ,inVienna
                       ,servingPizza
                       ,inViennaOrServingPizza]

testisAsymetric = qCheckBinaryP
                  (prop_isAsymetric ::  CtxType-> CtxType-> Property)

testisTransitive = qCheckTernaryP
                   (prop_isTransitive :: CtxType-> CtxType-> CtxType->Property)

testisLeastContext = qCheckUnary
                      (prop_isLeastContext:: CtxType-> Bool)

testisGreatestContext = qCheckUnary
                         (prop_isGreatestContext :: CtxType-> Bool)

testisIdempotent = qCheckUnary
                    (prop_isIdempotent :: CtxType-> Bool)

testisCommutative = qCheckBinaryP
                     (prop_isCommutative :: CtxType-> CtxType-> Property)

testisAssociative = qCheckTernaryP
                      (prop_isAssociative :: CtxType-> CtxType-> CtxType-> Property)

testisConsistent = qCheckBinaryP
                    (prop_isConsistent :: CtxType-> CtxType-> Property)

testisIsotone = qCheckTernaryP
                 (prop_isIsotone :: CtxType-> CtxType-> CtxType-> Property)

testwithCombLeastContext = qCheckUnary
                            (prop_withCombLeastContext:: CtxType-> Bool)

testwithCombGreatestContext = qCheckUnary
                               (prop_withCombGreatestContext:: CtxType-> Bool)

testisDoubleComp = qCheckUnary
                    (prop_isDoubleComp:: CtxType-> Bool)

testisLessThenComp = qCheckBinaryP
                      (prop_isLessThenComp :: CtxType-> CtxType-> Property)

testwithCombBoundComp = qCheckUnaryP
                         (prop_withCombBoundComp:: CtxType-> Property)
```

Algorithm A.13: Module ContextAlgebraEvaluation, part 2

```
main = sequence_ [putStrLn "test asymmetric:"
                 ,testisAsymetric
                 ,putStrLn "test transitive:"
                 ,testisTransitive
                 ,putStrLn "test is least context:"
                 ,testisLeastContext
                 ,putStrLn "test is greatest context:"
                 ,testisGreatestContext
                 ,putStrLn "test is idempotent:"
                 ,testisIdempotent
                 ,putStrLn "test is commutative:"
                 ,testisCommutative
                 ,putStrLn "test is associative:"
                 ,testisAssociative
                 ,putStrLn "test is consistent:"
                 ,testisConsistent
                 ,putStrLn "test is isotone:"
                 ,testisIsotone
                 ,putStrLn "test combination with least context:"
                 ,testwithCombLeastContext
                 ,putStrLn "test combination with greatest context:"
                 ,testwithCombGreatestContext
                 ,putStrLn "test double complement property:"
                 ,testisDoubleComp
                 ,putStrLn "test less then complement property:"
                 ,testisLessThenComp
                 ,putStrLn "test combination with complement:"
                 ,testwithCombBoundComp]
```

Algorithm A.14: Module QuickCheck-Helper

```haskell
{-# LANGUAGE TypeSynonymInstances, FlexibleInstances #-}

module Evaluation.QuickCheckHelper where
import Test.QuickCheck

-- * Helper Functions for QuickCheck
numberTests = 10000

qCheckUnary :: (Arbitrary c, Show c) =>  (c -> Bool) -> IO()
qCheckUnary  = quickCheckWith stdArgs { maxSuccess = numberTests }

vCheckUnary ::  (Arbitrary c, Show c) => (c -> Bool) -> IO()
vCheckUnary  = verboseCheckWith stdArgs { maxSuccess = numberTests }

qCheckBinary :: (Arbitrary c, Show c) => (c -> c -> Bool) -> IO()
qCheckBinary  = quickCheckWith stdArgs { maxSuccess = numberTests }

vCheckBinary ::  (Arbitrary c, Show c) =>(c -> c -> Bool) -> IO()
vCheckBinary  = verboseCheckWith stdArgs { maxSuccess = numberTests }

qCheckTernary ::  (Arbitrary c, Show c) => (c -> c-> c -> Bool) -> IO()
qCheckTernary  = quickCheckWith stdArgs { maxSuccess = numberTests }

qCheckUnaryP ::  (Arbitrary c, Show c) => (c -> Property) -> IO()
qCheckUnaryP  = quickCheckWith stdArgs { maxSuccess = numberTests }

vCheckUnaryP ::  (Arbitrary c, Show c) => (c -> Property) -> IO()
vCheckUnaryP  = verboseCheckWith stdArgs { maxSuccess = numberTests }

qCheckBinaryP ::  (Arbitrary c, Show c) => (c -> c -> Property) -> IO()
qCheckBinaryP  = quickCheckWith stdArgs { maxSuccess = numberTests }

vCheckBinaryP ::  (Arbitrary c, Show c) =>(c -> c -> Property) -> IO()
vCheckBinaryP  = verboseCheckWith stdArgs { maxSuccess = numberTests }

qCheckTernaryP ::  (Arbitrary c, Show c) => (c -> c-> c -> Property) -> IO()
qCheckTernaryP  = quickCheckWith stdArgs { maxSuccess = numberTests }

vCheckTernaryP :: (Arbitrary c, Show c) => (c -> c-> c -> Property) -> IO()
vCheckTernaryP  = verboseCheckWith stdArgs { maxSuccess = numberTests }
```

A.4 Complexity

This section presents the full source code necessary to execute the complexity benchmarks introduced in Section 7 and the results illustrated by Figures A.2 and A.3.

Complexity benchmark

The complexity benchmark is executed by `Main` (Algorithm A.15) module which relies on modules `ProtoCalculator` (Algorithm A.16) and `ConceptGenerator` (Algorithm A.17).

Algorithm A.15: Module Main for benchmarks

```
module Main where

import Criterion.Main
import System.IO

import Benchmark.ProtoCalculator
import Benchmark.ConceptGenerator

main :: IO()
main = do
 mapM_ makeSafe [stdout, stdin, stderr]
 generationBench

-- ^ needed because on terminal unicode characters abort the execution,
--   the workaround removes the Unicode characters
makeSafe h = do
  ce' <- hGetEncoding h
  case ce' of
    Nothing -> return ()
    Just ce -> mkTextEncoding ((takeWhile (/= '/') $ show ce) ++ "//TRANSLIT") >>=
      hSetEncoding h

prototypeBench = defaultMain
 [bgroup "prototypeCalculation"$
  map (\c ->bench (label c) $ nfIO (calcProto c)) [1..ctx]
 ]
  where ctx = fromInteger testNumberContexts
        exe = testNumberExemplars
        label c = "meets-"++(show c)
                   ++"-contexts-"++(show testNumberContexts)
                   ++"-exemplars-"++(show exe)

generationBench = defaultMain
 [bgroup "Concept generation"$
  map (\c -> bench (label c)
            $ nfIO (createAndStoreConcept c exemplars)) [1.. maxNumberContext]
 ]
  where maxNumberContext = 12
        exemplars = 1
        label c =  ("contexts-"++(show c)++"-exemplars-"++(show exe))
```

Algorithm A.16: Module ConceptGeneration for benchmarks

```haskell
{-# LANGUAGE TypeSynonymInstances, FlexibleInstances #-}
module Benchmark.ConceptGenerator (createAndStoreConcept) where
import Test.QuickCheck
import Data.List (permutations)
import qualified Data.Set as Set
import ContextAlgebra
import Concept
import SetImplementation.ContextAlgebraSetInstance
type CtxType = Context String
type Exemplar = String
maxNumberContextsForArbitrary = 500
maxNumberExemplarsForArbitrary = 500

createAndStoreConcept :: Int -> Int -> IO ()
createAndStoreConcept numberContexts numberExemplars = do
   exemplars <- getNumberElements getrandomExemplar numberExemplars
   atomicContexts <- getNumberElements getrandomContext numberContexts
   let allContextCombin = establishContextLattice atomicContexts
   concepts <- sequence $! map (genConcept4Ctx exemplars) allContextCombin
   let concept = unionConcept concepts
       filepath = "./src/Benchmark/TestConcepts/Concept-"++(show numberContexts)++"-"
                  ++(show numberExemplars)++".txt"
   writeConceptFile filepath concept

getNumberElements ::(Ord a)=> (IO a)-> Int -> IO [a]
getNumberElements f number =  do
 elements <- sequence $! replicate (toAbsAndZeroToOne number) f
 let a = Set.fromList elements
 if Set.size a == number then return elements -- check if all elements are distinct
   else getNumberElements f number

getrandomContext =generate (arbitrary::Gen CtxType)
getrandomExemplar =generate (arbitrary::Gen Exemplar)
getrandomInt = generate ( arbitrary::Gen Int)

establishContextLattice :: [CtxType] -> [CtxType]
establishContextLattice atomicRep = ordNub createdContexts
 where
  disjunctions start ctxlist = scanl Set.union start ctxlist
  permutateAtomics = permutations atomicRep
  createdContexts =  concatMap (\v -> disjunctions (head v) (tail v)) permutateAtomics

genConcept4Ctx :: [Exemplar] -> CtxType  -> IO (Concept CtxType Exemplar)
genConcept4Ctx  exemplars ctx  = do
  numbers <- getNumberElements getrandomInt (length exemplars)
  let concepts = zipWith (\e a ->fromObservationList $! replicate
                          (toAbsAndZeroToOne a) (O (ctx,e))) exemplars numbers
  return $! unionConcept concepts

writeConceptFile :: FilePath -> Concept CtxType Exemplar -> IO()
writeConceptFile filename concept = writeFile filename (show concept)

instance {-# OVERLAPS #-} Arbitrary CtxType where
  arbitrary = elements $ take maxNumberContextsForArbitrary generateContextsUniverse
generateContextsUniverse  = map createContext $ generateString  "c_"
instance {-# OVERLAPS #-} Arbitrary Exemplar where
  arbitrary = elements $ take maxNumberExemplarsForArbitrary generateExemplars
generateExemplars  = generateString "e_"
generateString:: String -> [String]
generateString prefix =  [x++[a] | x <- prefix : strings, a <- ['a'..'z']]
 where strings = [x++[a] | x <- prefix : strings, a <- ['a'..'z']]
toAbsAndZeroToOne a = if a == 0 then 1 else abs a
```

Algorithm A.17: Module ProtoCalculator for benchmarks

```haskell
{-# LANGUAGE TypeSynonymInstances, FlexibleInstances, MultiParamTypeClasses #-}
module Benchmark.ProtoCalculator(testNumberContexts,testNumberExemplars, calcProto) where
import System.IO.Unsafe
import qualified Data.Set as Set
import ContextAlgebra
import SetImplementation.ContextAlgebraSetInstance
import Concept
import Mapping
import ContextualizedConcept
import ExemplarScales.NominalExemplars
type CtxType = Context String
type Exemplar = String

-- has to be set
testNumberContexts = 10
testNumberExemplars = 1

calcProto :: Int -> IO (Exemplar, Double)
calcProto numberDisjunctions = do
  concept <-readConcept  testNumberContexts testNumberExemplars
  let extractContext = ordNub $ map getContext $ toObservationList concept
      toCombineContexts = take numberDisjunctions extractContext
      contextualizedCon =  m (automaticDisjunction toCombineContexts) concept
  return $ calculatePrototype contextualizedCon

readConcept testNumberContexts testNumberExemplars = readConceptFile filePath
 where
  filePath = "./src/Benchmark/TestConcepts/Concept-"
             ++(show testNumberContexts)++"-"
             ++(show testNumberExemplars)++".txt"

readConceptFile :: FilePath -> IO (Concept CtxType Exemplar)
readConceptFile filename = do
  file<-readFile filename
  let parsedConcept = read file
  return parsedConcept

automaticDisjunction ::  [CtxType] -> CtxType
automaticDisjunction = foldl Set.union leastContext -- changed

instance Universe String where
 atomicRepresentation = map (Set.elemAt 0) extractAtomicRepresentation

extractAtomicRepresentation = filter ((==) 1 . length)  allContexts
 where concept = unsafePerformIO $ readConcept  testNumberContexts testNumberExemplars
       allContexts = ordNub . map getContext . toObservationList $ concept

instance Mapping CtxType Exemplar where
 m ctx concept
  | leastContext `equals` ctx = emptyConcept
  | otherwise =  fromObservationList
                  [obs | obs <- toObservationList concept
                       , getContext obs `isMoreSelectiveOrEqualSel` ctx]

-- Taken From Yi
ordNub :: (Ord a) => [a] -> [a]
ordNub l = go Set.empty l
  where
    go _ []     = []
    go s (x:xs) = if x `Set.member` s then go s xs
                                      else x : go (Set.insert x s) xs
```

Measurement results

The measurement values for the time needed to calculate a prototype for different numbers of exemplars and context lattice sizes are illustrated in Figures A.2 and A.3..

The measurement values for context lattices each contextualized concept mapped to one exemplar is given in Figure A.2. For the measurement including ten, eleven and twelve contexts the upper bound Functions A.1, A.2 and A.3 are determined.

$$2^c \cdot 8c \cdot 10^{-6} + 0.12 \tag{A.1}$$

$$2^c \cdot 8c \cdot 10^{-6} + 0.25 \tag{A.2}$$

$$2^c \cdot 8c \cdot 10^{-6} + 0.52 \tag{A.3}$$

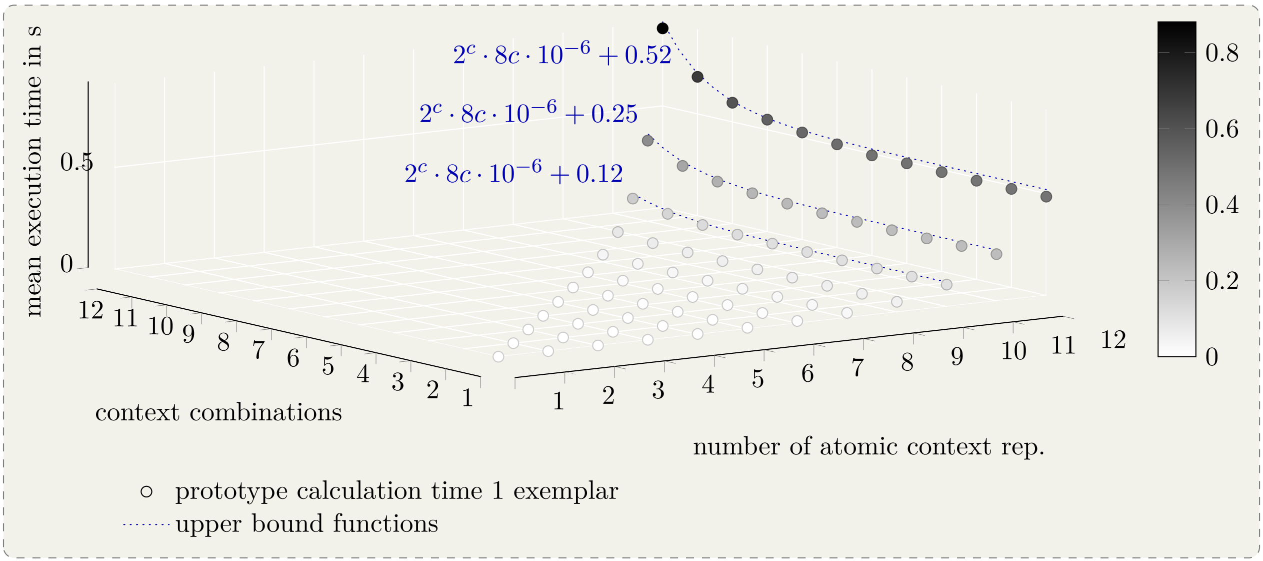


Figure A.2: Measurements for prototype calculation time including one exemplar

The measurement values for context lattices each contextualized concept mapped to ten exemplars is given in Figure A.3. For the measurement including ten, eleven and twelve contexts the upper bound Functions A.4, A.5 and A.6 are determined.

$$2^c \cdot 12c \cdot 10^{-5} + 1.2 \tag{A.4}$$

$$2^c \cdot 12c \cdot 10^{-5} + 2.5 \tag{A.5}$$

$$2^c \cdot 12c \cdot 10^{-5} + 5.5 \tag{A.6}$$

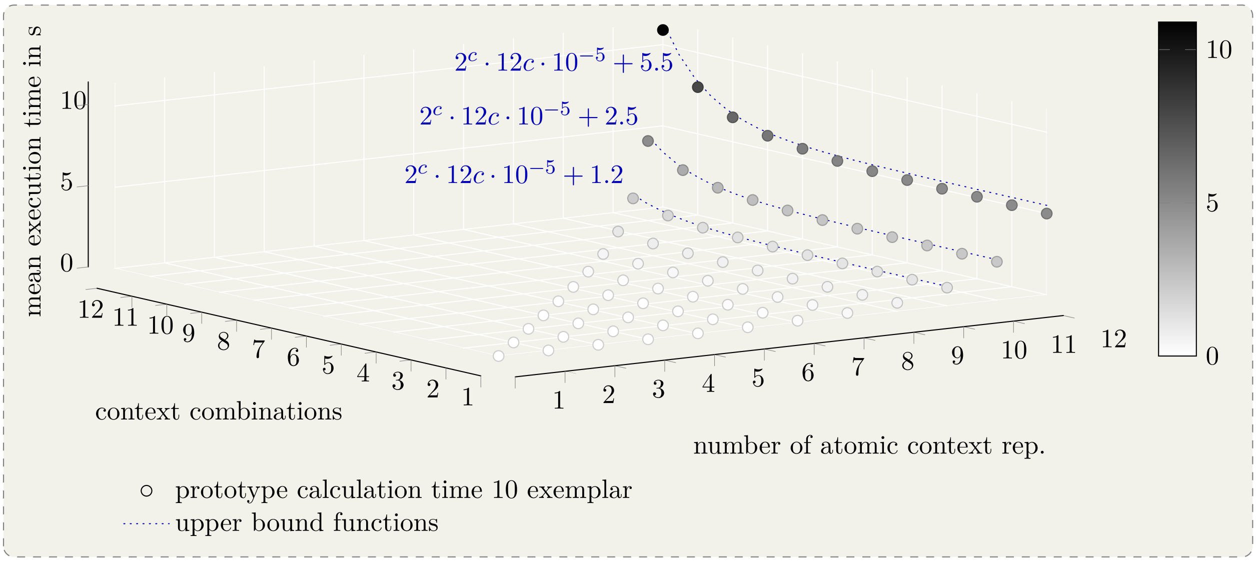


Figure A.3: Measurements for prototype calculation time including 10 exemplars

B
Representation of "near"

The representation of **near** is initialized with data extracted in a manual work flow. They are analyzed in order to build realistic contextual typicalities. The core of the manual work flow is a web application using a database. The relational data model used in the database to store the data is given in Section B.1. Two data collections retrieved from online platforms are imported into the database using the method presented in Section B.2. The web application is introduced in Section B.3 displaying data, enabling determination of context, location of locatum and relatum etc. and storing it in the database. The processed data are analyzed in the database where results are given in Section B.4. The results are retrieved from the database to create contextual typicalities where the method and results are given in Section B.5.

B.1 Relational data model

The data needed to represent **near** and to build the web application are stored into four relations illustrated in Figure B.1. The two relations `willhaben` and `derStandard` include all data imported from the provided data files. These relations include all attributes that are given in the data files. In Figure B.1 these two relations are on the left where points *e.g.* ... indicate more attributes that are not further used. Only those attributes are listed that are used in the further process. From both relations, those entries that include "nahe" or "nähe" in their description or title are copied into the relation `near`. Relation `near` stores all information that are determined while the data acquisition process. Relation `users` includes attributes to identify users.

`Near` is the main relation storing data and extracting several views to calculate contextual typicality and prototypes. The attributes are mostly self explanatory starting with an unique identifier `id` for each real estate entry. `StartText` and `endText` include the addresses of the relatum and locatum in natural language. Attribute `beschreibung` is filled with one of these three categories: `'not located'`, `'partly located'` and `located'` according to the categorization made by the user for this real estate entry. Attributes `Contexts` includes the determined contexts, attribute `start` the location of the relatum and attribute `end` the location of the relatum in coordinates. Attribute `route` stores the calculated direct line between relatum and locatum. The following attributes `idderstandard` and `idwillhaben` include foreign keys to reference entries in relations `willhaben` and `derStandard`. Attributes `StartStatus` and `EndStatus` were designed for another classification which was not applied and are left empty. According to the processing status of an entry the attribute `status` is `''` (empty) for todo, `'inWork'` for in progress and `'finished'` for entries that are completely processed. For each status an own view is created. Attribute `userid` stores the identifier of the user that set this entry to status `'finished'`.

The locations of the relatum and locatum are determined on a map shown in reference system EPSG code 4326[1] not having defined a projection to measure metric distances. To enable metric distance measurements view `near3857` is created that projects the locations into the projection system EPSG code 3857[2]. From this view other views are created that categorize the processed real estate entries. According the categorization the views `unbrauchbar3857` for not located entries, `brauchbar3957` for partly located entries and `perfekt3957` for entries that can be further processed are established. To distinguish these views the attribute `beschreibung` is used. The entries included in the view `perfekt3857` are analyzed and used to initialize the context algebra for **near**.

B.2 Data import

The two data sets (willhaben and derStandard) are given in two different formats *e.g.* CSV and XML. Each data set is imported with different functionality. Both import implementations share the usage of Haskell library `HDBC` to communicate with the database. Algorithms used to import the willhaben data set into the relation willhaben are given in following section. Algorithms used to import the derStandard data set into relation derStandard also follow.

Data import from "willhaben" data set

Willhaben data set is provided in one single file formatted as comma separated values (CSV). `Main` module (Algorithm B.1) includes all other modules and is executed reading the file "willhaben.csv". The "willhaben.csv" file is parsed by functions included in module `Willhaben` (Algorithm B.2) using library cassava `http://hackage.haskell.org/package/cassava` where each entry is transformed into a Haskell type. This data are saved into relation "willhaben" (Figure B.1) in the database with functionality included in module `Database` (Algorithm B.3).

[1] `http://epsg.io/4326`

[2] `http://epsg.io/3857`

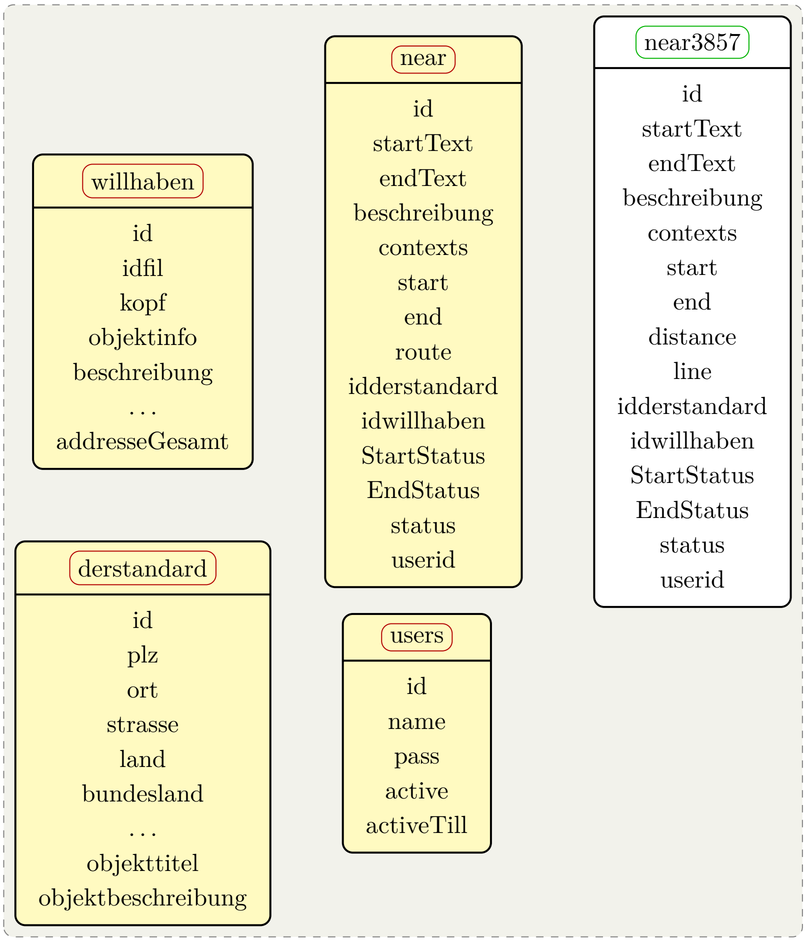


Figure B.1: Relational data base schema used by the Web Application

Algorithm B.1: Module Main for importing “willhaben” data set

```
module Main where

import Willhaben
import Database

main = do
  let willhabenfile = "willhaben.csv"
  willhabenentries <-readWillhabenFile willhabenfile
  insertWillhabenValuesInDB willhabenentries
  putStrLn $ "willhaben File: "++ willhabenfile++" processed"
```

Algorithm B.2: Module Willhaben

```haskell
{-# LANGUAGE DeriveGeneric ,OverloadedStrings #-}
module Willhaben (readWillhabenFile, WillhabenEntry(..),processEveryEntry,checklength)
  where

import Data.Csv
import qualified Data.ByteString.Lazy as BL
import qualified Data.Vector as V
import Data.Text
import GHC.Generics
import qualified Data.Char as C

data WillhabenEntry = WillhabenEntry {
 idfile :: !Text,  xufeffId :: !Text, link2detail :: !Text, kopf :: !Text, code :: !Text,
 objektinfo :: !Text, energie :: !Text,  ausstattung :: !Text, beschreibung :: !Text,
 preis :: !Text,preisdetails :: !Text, flaechen :: !Text,  zusatzinfos :: !Text,
 kontakt :: !Text,lage :: !Text, image :: !Text, adrclean :: !Text,summe :: !Text,
 code2 :: !Text, updated :: !Text, objekttyp :: !Text, bautyp :: !Text, wohnm2 :: !Text,
 nutzm2 :: !Text, grundm2 :: !Text, gartenm2 :: !Text, gesamtm2 :: !Text,
 terassenm2 :: !Text,balkonm2 :: !Text, lift :: !Text, zimmer :: !Text,
 badezimmer :: !Text, wc :: !Text, availableAt :: !Text, heizung :: !Text,floor :: !Text,
 mietkauf :: !Text, kaufpreis :: !Text, gesamtmiete :: !Text, nettomiete :: !Text,
 bk :: !Text, compensation :: !Text, maklerprovision :: !Text, baujahr :: !Text,
 hwb :: !Text, plz :: !Text,ort :: !Text,strasse :: !Text, wohnnutzm2 :: !Text,
 objektBautyp :: !Text, addresseGesamt :: !Text, addressIndex :: !Text, lat :: !Text,
 long :: !Text, popRate :: !Text, preism2 :: !Text, mietpreism2 :: !Text
 }
 deriving (Generic, Show)

instance FromRecord WillhabenEntry where
 parseRecord v
 | Prelude.length v > 0 =
   WillhabenEntry <$> v .! 0 <*> v .! 1 <*> v .! 2 <*> v .! 3 <*> v .! 4  <*> v .! 5
                  <*> v .! 6 <*> v .! 7 <*> v .! 8 <*> v .! 9 <*> v .! 10 <*> v .! 11
                  <*> v .! 12 <*> v .! 13 <*> v .! 14 <*> v .! 15 <*> v .! 16 <*> v .! 17
                  <*> v .! 18 <*> v .! 19 <*> v .! 20 <*> v .! 21 <*> v .! 22 <*> v .! 23
                  <*> v .! 24 <*> v .! 25 <*> v .! 26 <*> v .! 27 <*> v .! 28 <*> v .! 29
                  <*> v .! 30 <*> v .! 31 <*> v .! 32 <*> v .! 33 <*> v .! 34 <*> v .! 35
                  <*> v .! 36 <*> v .! 37 <*> v .! 38 <*> v .! 39 <*> v .! 40 <*> v .! 41
                  <*> v .! 42 <*> v .! 43 <*> v .! 44 <*> v .! 45 <*> v .! 46 <*> v .! 47
                  <*> v .! 48 <*> v .! 49 <*> v .! 50 <*> v .! 51 <*> v .! 52 <*> v .! 53
                  <*> v .! 54 <*> v .! 55 <*> v .! 56
 | otherwise = fail "failed to parse"

myOptions = defaultDecodeOptions {
      decDelimiter = fromIntegral (C.ord '\t')
    }

readWillhabenFile :: String -> IO ([WillhabenEntry])
readWillhabenFile filename= do
 csvData <- BL.readFile filename
 case (decodeWith myOptions HasHeader csvData::Either String(V.Vector WillhabenEntry))of
        Left err -> do Prelude.putStrLn err; return [];
        Right v ->  return $ V.toList  v;

checklength :: [WillhabenEntry] ->[[Int]]
checklength  =  Prelude.map (processEveryEntry Data.Text.length)

processEveryEntry :: (Text -> b) -> WillhabenEntry -> [b]
processEveryEntry f we = Prelude.map f [idfile we, xufeffId we, link2detail we, kopf we,
 code we, objektinfo we, energie we, ausstattung we, beschreibung we, preis we,
 preisdetails we, flaechen we, zusatzinfos we, kontakt we, lage we,  image we,
 adrclean we, summe we, code2 we,  updated we,  objekttyp we, bautyp we, wohnm2 we,
 nutzm2 we, grundm2 we, gartenm2 we, gesamtm2 we, terassenm2 we, balkonm2 we, lift we,
 zimmer we, badezimmer we, wc we, availableAt we, heizung we, Willhaben.floor we,
 mietkauf we, kaufpreis we, gesamtmiete we, nettomiete we, bk we, compensation we,
 maklerprovision we, baujahr we, hwb we, plz we, ort we, strasse we,  wohnnutzm2 we,
 objektBautyp we, addresseGesamt we,  addressIndex we, lat we, long we, popRate we,
 preism2 we, mietpreism2 we]
```

Algorithm B.3: Module Database for the “willhaben” data set

```
module Database (createWillhabenTable, insertWillhabenValuesInDB) where

import Control.Exception
import Database.HDBC
import Database.HDBC.PostgreSQL
import Willhaben

connectionString = "host=127.0.0.1 dbname=spatialConceptNear user=postgres password=t"

createConnection =  connectPostgreSQL connectionString

createWillhabenTable = do
  conn <- createConnection
  dropstatement <- prepare conn "drop table if exists willhaben;"
  let createstatement = "create table willhaben (id serial, idfile varchar(5000),
    xufeffId varchar(5000), link2detail varchar(5000), kopf varchar(5000),
    code varchar(5000),  objektinfo varchar(5000), energie varchar(5000),
    ausstattung varchar(5000), beschreibung varchar(5000), preis varchar(5000),
    preisdetails varchar(5000), flaechen varchar(5000),  zusatzinfos varchar(5000),
    kontakt varchar(5000),lage varchar(5000),image varchar(5000), adrclean varchar(5000),
    summe varchar(5000), code2 varchar(5000), updated varchar(5000),
    objekttyp varchar(5000), bautyp varchar(5000), wohnm2 varchar(5000),
    nutzm2 varchar(5000), grundm2 varchar(5000), gartenm2 varchar(5000),
    gesamtm2 varchar(5000), terassenm2 varchar(5000), balkonm2 varchar(5000),
    lift varchar(5000), zimmer varchar(5000), badezimmer varchar(5000), wc varchar(5000),
    availableAt varchar(5000), heizung varchar(5000), floor varchar(5000),
    mietkauf varchar(5000), kaufpreis varchar(5000), gesamtmiete varchar(5000),
    nettomiete varchar(5000), bk varchar(5000), compensation varchar(5000),
    maklerprovision varchar(5000), baujahr varchar(5000), hwb varchar(5000),
    plz varchar(5000), ort varchar(5000), strasse varchar(5000),
    wohnnutzm2 varchar(5000), objektBautyp varchar(5000), addresseGesamt varchar(5000),
    addressIndex varchar(5000),lat varchar(5000),long varchar(5000),popRate varchar(5000),
    preism2 varchar(5000),mietpreism2 varchar(5000));"
  sqlstatement <- prepare conn createstatement
  execute dropstatement []
  execute sqlstatement []
  commit conn
  disconnect conn
  return ()

insertWillhabenValuesInDB :: [WillhabenEntry]-> IO()
insertWillhabenValuesInDB  willhabenentries= do
  conn <- createConnection
  insertstatement <- prepare conn "insert into willhaben ( idfile, xufeffId, link2detail,
    kopf, code,  objektinfo, energie,  ausstattung, beschreibung, preis, preisdetails,
    flaechen,  zusatzinfos, kontakt, lage, image, adrclean, summe, code2, updated,
    objekttyp, bautyp, wohnm2, nutzm2, grundm2, gartenm2, gesamtm2, terassenm2, balkonm2,
    lift, zimmer,  badezimmer, wc,  availableAt, heizung, floor, mietkauf, kaufpreis,
    gesamtmiete, nettomiete, bk, compensation, maklerprovision, baujahr, hwb, plz, ort,
    strasse, wohnnutzm2, objektBautyp, addresseGesamt, addressIndex, lat, long, popRate,
    preism2, mietpreism2) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,
    ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);"
  executeMany insertstatement $! toSqlValues willhabenentries
  commit conn
  disconnect conn

toSqlValues :: [WillhabenEntry]  -> [[SqlValue]]
toSqlValues  = map (processEveryEntry toSql)
```

Data import from “derStandard” data set

DerStandard data is provided in XML format in schema Openimmo[3]. The schema groups the information into pieces which supports extraction of necessary data for data acquisition. For example the Geo tag

[3] http://www.openimmo.de

(<Geo>) includes the sub tags: <plz> for the Austrian abbreviation for zip code, <ort> for city, <strasse> for street, <bundesland> for federal state, and a user defined simplified tag which can be used to find the location of the relatum. The tag <freitexte> (translated as free text) includes a description of the real estate object. It is split into a <objekttitel> (title of the object) and a <objektbeschreibung> (object description). This tags are used to extract the location of the locatum and the context.

Module `Main` includes the functionality to parse the XML file, create Haskell types from it and saves them into the database. The `main` (Algorithm B.4) function combines the functions starting with parsing the file "export.XML" using functions included in module `ParseOpenImmo` (Algorithm B.5) which relies on the library xml-conduit http://hackage.haskell.org/package/xml-conduit. The parsed data are transformed into Haskell types created in module `Openimmo` (Algorithm B.6). The data are then saved into relation "derStandard" (Figure B.1) via the module `Database` (Algorithm B.7).

Algorithm B.4: Module Main for importing "derStandard" data set

```haskell
module Main where

import Openimmo
import ParseOpenimmo
import Database

main :: IO ()
main = do
    let filename = "export.xml"
    immobilien <- extractImmobilien filename
    insertDerStandardValuesinDB $! immobilien
```

Algorithm B.5: Module ParseImmo

```
{-# LANGUAGE OverloadedStrings, BangPatterns #-}
module ParseOpenimmo  (extractImmobilien) where

import Openimmo
import qualified Text.XML as X
import Text.XML.Cursor
import qualified Data.Text as T

extractImmobilien :: String -> IO [Immobilie]
extractImmobilien filename = do
    doc <- X.readFile X.def filename
    let cursor = fromDocument doc
    let immobilien = map extractImmobilie (cursor $/ element "anbieter"
                                                  &/ element "immobilie")
    return immobilien

extractImmobilie :: Cursor -> Immobilie
extractImmobilie c= Immobilie {geo= extractGeo c
                              ,freitexte =  extractFreitexte c
                              ,objektkat =  Objektkategorie "" "" "" "" "" ""
                              ,preise = Preise "" "" ""
                              ,verwaltungTech =VerwaltungTechn "" "" "" ""
                              }

extractGeo :: Cursor -> Geo
extractGeo c = Geo {plz = nplz, ort = nort, strasse = nstrasse, land= nland,
                    bundesland = nbundesland, etage = netage,
                    user_defined_simplified= nuser_defined_simplified}
 where nplz  = T.strip $ T.concat $ c $/ element "geo" &/ element "plz" &// content
       nort = T.strip $ T.concat $ c $/ element "geo" &/ element "ort" &// content
       nstrasse = T.strip $ T.concat $ c $/ element "geo" &/ element "strasse"
                          &// content
       nland = T.strip $ T.concat $ c $/ element "geo" &/ element "land"
                       >=> attribute "iso_land"
       nbundesland = T.strip $ T.concat $ c $/ element "geo" &/ element "bundesland"
                             &// content
       netage= T.strip $ T.concat $ c $/ element "geo" &/ element "etage" &// content
       nuser_defined_simplified= T.strip $ T.concat $ c $/ element "geo"
                                        &/ element "user_defined_simplefield" &// content

extractFreitexte :: Cursor -> Freitext
extractFreitexte c = Freitexte {objekttitel = nObjekttitel,
                                objektbeschreibung = nObjektbeschreibung}
 where nObjekttitel =T.strip $ T.concat $ c $/ element "freitexte"
                             &/ element "objekttitel" &// content
       nObjektbeschreibung = T.strip $ T.concat $ c $/ element "freitexte"
                             &/ element "objektbeschreibung" &// content
```

Algorithm B.6: Module OpenImmo

```
module Openimmo where

import  Data.Text hiding (map, concat)

data Immobilie = Immobilie {geo :: Geo, objektkat :: Objektkategorie, preise :: Preise,
                            freitexte :: Freitext, verwaltungTech :: VerwaltungTechn}
                           deriving Show;

applyfImmobilie ::(Text -> b) -> Immobilie -> [b]
applyfImmobilie f i = concat [applyfGeo f $ geo i, applyfObjektkategorie f
                              $ objektkat i,applyfPreise f $ preise i, applyfFreitexte f
                              $ freitexte i,applyfVerwaltungTechn f $ verwaltungTech i ]


data Geo = Geo {plz :: !Text,ort :: !Text, strasse :: !Text, land :: !Text,
                bundesland:: !Text, etage :: !Text, user_defined_simplified :: !Text }
              deriving Show;

applyfGeo  ::(Text -> b) -> Geo -> [b]
applyfGeo f g = map f [plz g, ort g, strasse g, land g, bundesland g, etage g,
                       user_defined_simplified g]

data Objektkategorie = Objektkategorie  {miete_pacht :: !Text, kauf :: !Text,
                                         wohnen:: !Text, gewerbe:: !Text,anlage:: !Text,
                                         objektart:: !Text} deriving Show;

applyfObjektkategorie ::(Text -> b) -> Objektkategorie -> [b]
applyfObjektkategorie f o = map f [miete_pacht o, kauf o, wohnen o, gewerbe o
                                  ,anlage o, objektart o]

data Preise = Preise {kaltmiete:: !Text, kaution:: !Text,
                      provisionspflichtig:: !Text} deriving Show;

applyfPreise :: (Text -> b) -> Preise -> [b]
applyfPreise f p = map f [kaltmiete p, kaution p, provisionspflichtig p]


data Freitext = Freitexte {objekttitel :: !Text, objektbeschreibung :: !Text}
                         deriving Show;

applyfFreitexte :: (Text -> b) -> Freitext -> [b]
applyfFreitexte f text = map f [objekttitel text, objektbeschreibung text]


data VerwaltungTechn = VerwaltungTechn {objektnrExtern ::!Text, oppenimmoObid :: !Text
                                       ,standVom :: !Text, aktion:: !Text}
                                     deriving Show;

applyfVerwaltungTechn :: (Text -> b) -> VerwaltungTechn -> [b]
applyfVerwaltungTechn f v = map f [objektnrExtern v, oppenimmoObid v, standVom v
                                  , aktion v]
```

Algorithm B.7: Module Database for “derStandard” data set

```
module Database where

import Control.Exception
import Database.HDBC
import Database.HDBC.PostgreSQL
import Openimmo

connectionString = "host=127.0.0.1 dbname=spatialConceptNear user=postgres password=t"
createConnection = connectPostgreSQL connectionString

createDerStandardtable = do
  conn <- createConnection
  dropStatement <- prepare conn "drop table if exists derstandard ;"
  let cS = "create table derstandard (id serial , plz varchar(1000), ort varchar(1000),
            strasse varchar(1000), land varchar(1000), bundesland varchar(1000),
            etage varchar(1000), userDefinedSimplified varchar(1000),
            mietePacht varchar(1000), kauf varchar(1000), wohnen varchar(1000),
            gewerbe varchar(1000), anlage varchar(1000), objektart varchar(1000),
            kaltmiete varchar(1000), kaution varchar(1000),
            provisionspflichtig varchar(1000), objekttitel varchar(10000),
            objektbeschreibung varchar(10000), objektnrExtern varchar(1000),
            openimmoObid varchar(1000), standVom varchar(1000), aktion varchar(1000))"
  createStatement <- prepare conn cS
  execute dropStatement []
  execute createStatement []
  commit conn
  disconnect conn
  return ()

insertDerStandardValuesinDB :: [Immobilie] -> IO()
insertDerStandardValuesinDB entries = do
  conn <- createConnection
  let in= "insert into derstandard (plz, ort, strasse, land, bundesland, etage,
           userDefinedSimplified, mietePacht, kauf, wohnen, gewerbe, anlage, objektart,
           kaltmiete, kaution, provisionspflichtig, objekttitel, objektbeschreibung,
           objektnrExtern, openimmoObid, standVom, aktion)
           values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
  insertstatement <- prepare conn in
  executeMany insertstatement $! toSqlValues entries
  commit conn
  a <- quickQuery' conn "select * from users" []
  print a
  return()

toSqlValues :: [Immobilie]  -> [[SqlValue]]
toSqlValues = map (applyfImmobilie toSql)
```

B.3 Web Application

The Web application is implemented using Haskell and web technologies distinguished into several directories. The application uses a postgres database and is executed via scotty (http://hackage.haskell.org/package/scotty) webserver with the Glasgow Haskell Compiler (ghc). The source code directory for this web application is illustrated by Figure B.2. Directory js includes the JavaScript libraries and the file myquery.js that is necessary for the web map. Directory css includes the file mycss.css including styling instructions in Cascading Style Sheet (CSS) format. Directory images includes the images used as markers on the web map *e. g.*, letter_a.png A, letter_e.png E etc. Directory src includes all Haskell modules executed by the web server. The Haskell modules offering the functionality is split into the four directories Database, Service, View, DataTransferObject.

Scotty web server handles HTTP requests and offers functions for easy HTTP parameter parsing. Scotty executes the Main module (given in Algorithms B.8, B.9, B.10) that integrates all other modules needed to create the processing work flow of real estate entries.

More external libraries are used to establish the necessary functionality. For example, the external library GeocoderOpenCage(http://hackage.haskell.org/package/GeocoderOpenCage) is included to query an online gazetteer. In order to interact with this package the module Geocode given in Algorithm B.18 is created. Another library scotty-session(http://hackage.haskell.org/package/scotty-session) enables a session managing using cookies. A session is only created for already known users in the database where the functionality to interact with the database is included in module SessionHandling given in Algorithm B.12. All database related functions rely on the external library HDBC(http://hackage.haskell.org/package/HDBC-postgresql) where shared functions for all modules are included in module DBUtils given in Algorithm B.15. The data retrieved for real estate entries from modules StatusProcessing (Algorithm B.11) and ImmobilieProcessing (Algorithm B.13, B.14) are translated into Haskell objects via the modules Immobilie (Algorithm B.16) and Status (Algorithm B.17). The Haskell objects are marked up using library blaze (http://hackage.haskell.org/package/blaze-html) in the modules ProcessView (Algorithms B.20, B.21) and ImmobilieView (Algorithm B.22, B.23) to establish HTML interfaces. Also the module LogInView (Algorithm B.19) uses blaze to generate the Log-On interface.

The front end interfaces (ProcessView, ImmobilieView and LogInView) are using JavaScript libraries and style the interface with HTML and CSS. The interfaces are responsible which is enabled by CSS library bootstrap (http://www.w3schools.com/bootstrap/). For individual styling of the elements own defined CSS file mycss.css (Algorithm B.24) is created. The web map uses the library OpenLayers3 (http://openlayers.org) and is customized and enriched with further functionality with the own defined JavaScript Algorithm (Algorithms B.25, B.26, B.27, B.28 and B.29) extensively using JQuery (http://jquery.com) for DOM processing and AJAX requests. To highlight the words “nahe” and “nähe” the library highlight (http://goo.gl/hDPYA) is used. The input field for contexts auto completes contexts by showing existing contexts in the database using library autocomplete (https://github.com/devbridge/jQuery-Autocomplete).

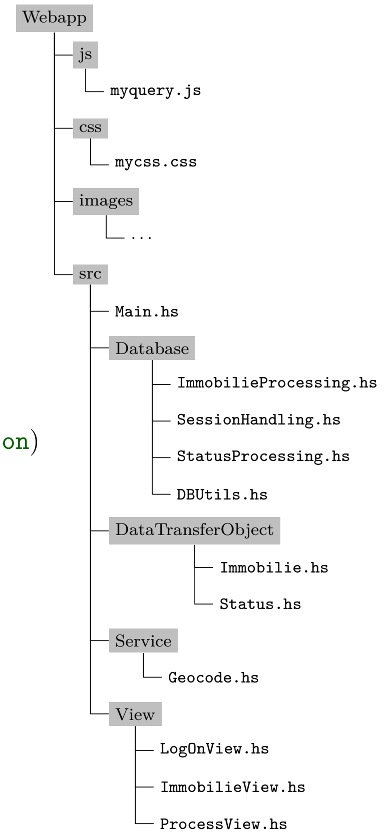


Figure B.2: Directory structure of the Haskell Web Application

Algorithm B.8: Module Main for the Web Application, part 1

```haskell
{-# LANGUAGE ScopedTypeVariables, OverloadedStrings #-}
module Main where
import Network.Wai.Middleware.RequestLogger
import Text.Blaze.Html.Renderer.Text
import Control.Monad.IO.Class
import Control.Monad
import Web.Scotty
import Web.Scotty.Cookie
import qualified Data.Text as T
import Data.Maybe
import Data.Text.Read
import qualified Data.List as L
import Data.Monoid
import View.LogOnView
import View.ProcessView
import View.ImmobilieView
import qualified View.Analysis as Analysis
import qualified Service.Geocode as G
import qualified Database.ImmobilieProcessing as IP
import qualified Database.StatusProcessing as SP
import qualified Database.SessionHandling as SH
import qualified Database.Analysis as A

main = scotty 3000 $ do
  middleware logStdoutDev
-- Sessions
  get "/" $ do html.renderHtml $ renderLoginPage
  post "/login" $ do
    username <- param "username"
    pass <- param "password"
    isUserknown <-liftIO $ checkIfknown username pass
    case isUserknown of
      Just a -> do
       session <-liftIO $ SH.createSession username
       setSimpleCookie "hits" $ T.pack $ show session
       redirect "/overview"
      Nothing -> redirect "/"
  get "/logout" $ do
    name <- liftM toUserSession $ getCookie "hits"
    let usersession = fromJust name
    deleteCookie "hits"
    liftIO $ SH.deleteSession usersession
    redirect "/"
-- workflows
  get "/overview" $ do
   hits<- liftM toUserSession $ getCookie "hits"
   isActive <- liftIO $ isCookieActive hits
   case isActive of
    Nothing -> redirect "/"
    Just session ->  do
              setSimpleCookie "hits" $ T.pack $ show session
              renderOverviewPage -- overview page
  get "/inprocess" $ do
              id <- param "id"
              renderStandardEntry id
  get "/inWork" $ do
              id <- param "id"
              liftIO $ SP.updateProcess id SP.Work
  get "/finished" $ do
   hits<- liftM toUserSession $ getCookie "hits"
   isActive <- liftIO $ isCookieActive hits
   case isActive of
    Nothing -> redirect "/"
    Just session ->  do
              setSimpleCookie "hits" $ T.pack $ show session
              id <- param "id"
              liftIO $ IP.processedByUser id (fst session)
              processfinishButton id
  get "/geocode" $ do
              id <- param "id"
              geocodeString <-param "toGeocode"
              geocode id geocodeString
```

Algorithm B.9: Module Main for the Web Application, part 2

```
-- additional information
   get "/addContext" $ do
                id <-param "id"
                ctx <- param "context"
                addContext id ctx
   get "/addBewertung" $ do
                id <-param "id"
                bewertung<- param "bewertung"
                addBeschreibung id bewertung
-- points
   get "/saveStartPoint" $ do
                id <- param "id"
                x <- param "X"
                y <- param "Y"
                savePoint id x y IP.Start
   get "/saveEndPoint"  $ do
                id <- param "id"
                x <- param "X"
                y <- param "Y"
                savePoint id x y IP.End
   get "/getStartPoint" $ do
                id <- param "id"
                getPoint id IP.Start
   get "/getEndPoint" $ do
                id <- param "id"
                getPoint id IP.End
   get "/saveGeocodeString" $ do
                id <- param "id"
                stringToSave <- param "elementString"
                ptype <- param "pointtype"
                saveGeocodingString id stringToSave (toPointType ptype)
   get "/autocomplete/contexts" $ do
                toJSONAllContext
   get "/analysis/start" $ do
     createAnalysisPage
   get "/analysis/contexts" $ do
     ctx <- param "context"
     geojson <- liftIO $ A.retrieveGeomForContext ctx
     json $ T.pack geojson
   -- get the ressources
   get "/images/:bar" $ do
    v <- param "bar"
    file $ "./images/"++ v
   get "/css/:bar" $ do
    v <- param "bar"
    file $"./css/"++ v
   get "/css/images/:bar" $ do
    v <- param "bar"
    file $"./css/images/"++ v
   get "/js/:bar" $ do
    v <- param "bar"
    file $"./js/"++ v

checkIfknown :: String -> String -> IO (Maybe String)
checkIfknown user pass =  SH.isUserPassValid user pass

createSessionForUser :: String -> IO SH.UserSession
createSessionForUser user = SH.createSession user

isCookieActive :: Maybe SH.UserSession -> IO (Maybe SH.UserSession)
isCookieActive maybeSession = case maybeSession of
  Just session ->  SH.isSessionActive session
  Nothing -> return Nothing

toUserSession :: Maybe T.Text -> Maybe SH.UserSession
toUserSession t = case t of
  Nothing -> Nothing
  Just text -> Just$ read $ T.unpack text
```

Algorithm B.10: Module Main for the Web Application, part 3

```
-- * Progress
renderOverviewPage :: ActionM()
renderOverviewPage = do
  todo <- liftIO SP.selectTodo
  progress <- liftIO SP.selectInWork
  finished <- liftIO SP.selectFinished
  html.renderHtml $ createOverview todo progress finished

renderStandardEntry :: Int-> ActionM ()
renderStandardEntry id = do
  liftIO $ SP.updateProcess id SP.Work
  naheEntry <- liftIO $ IP.retrieveImmobilie id
  contexts <- liftIO $ IP.retrieveContext id
  beschreibung<- liftIO $ IP.retrieveBeschreibung id
  html.renderHtml $ renderImmobilie naheEntry contexts beschreibung

processfinishButton :: Int -> ActionM()
processfinishButton id = do
   saveProcess id SP.Finished
   redirect "/overview#todo" -- renderOverviewPage

saveProcess :: Int -> SP.ProcessStatus -> ActionM()
saveProcess  id progress = liftIO $ SP.updateProcess id progress

-- * Point manipulation
getPoint ::Int->  IP.Point -> ActionM()
getPoint id pointtype = do
  point <- liftIO $ IP.retrievePoint id pointtype
  json $ T.pack point

savePoint :: Int -> String -> String -> IP.Point -> ActionM()
savePoint id x y pointtype = do
  liftIO $ IP.updatePoint id x y pointtype
  json $ T.pack x

geocode :: Int -> String -> ActionM()
geocode id geocodeString= do
  a<-liftIO $ G.geocodeText id geocodeString
  json a

toPointType :: T.Text -> IP.Point
toPointType a
  | a == "Start" = IP.Start
  | a == "End" = IP.End

saveGeocodingString ::Int -> String -> IP.Point -> ActionM()
saveGeocodingString id stringToSave ptype = do
  text <-liftIO $ IP.savePointString id ptype stringToSave
  json  text

-- * additional fields to fill in
addContext :: Int-> String-> ActionM()
addContext id ctx = do
  newctx <- liftIO $ IP.addNewContext id ctx
  json $ T.pack newctx

toJSONAllContext :: ActionM()
toJSONAllContext = do
  allContexts <-liftIO $IP.retrieveAllContexts
  let toJSON = map (\c -> T.concat [T.pack "{\"value\":\"",c , T.pack "\"}"]) allContexts
      most = T.concat (L.intersperse "," toJSON)
      all = T.concat [T.pack "{\"query\":\"Unit\",\"suggestions\":[", most ,T.pack "]}" ]
  json  all

addBeschreibung :: Int -> String -> ActionM()
addBeschreibung id besch= do
  newBeschreibung <- liftIO $ IP.addNewBeschreibung id besch
  json $ T.pack newBeschreibung
```

Algorithm B.11: Module StatusProcesssing

```
module Database.StatusProcessing (ProcessStatus(..)
                                  ,selectTodo
                                  ,selectInWork
                                  ,selectFinished
                                  ,updateProcess)
       where

import Database.HDBC
import Control.Monad
import qualified  Data.Text as T hiding (map, concat)

import DataTransferObject.Status
import Database.DBUtils

data ProcessStatus = Todo | Work | Finished

instance Show ProcessStatus where
  show (Todo) = "todo"
  show (Work) = "inWork"
  show (Finished) = "finished"

-- * Getting overview about entries
selectTodo :: IO [Status]
selectTodo  = selectS (convertToStatus  fromSql) "select * from todo limit 10"

selectInWork  :: IO [Status]
selectInWork  =selectS (convertToStatus  fromSql) "select * from inprogress"

selectFinished  :: IO [Status]
selectFinished  = selectS (convertToStatus  fromSql) "select * from finished"

updateProcess :: Int -> ProcessStatus -> IO ()
updateProcess id progress =  void $ updateFieldFrom id (show progress)
                                   "update near SET status=? where id=?"

convertToStatus :: (SqlValue -> T.Text) -> [SqlValue] -> Status
convertToStatus f [idSs, beschreibungS, contextsS, statusS, startstatusS, endstatusS
                 , usernameS]=
  S {idE = f idSs,
         beschreibung = f beschreibungS,
         contexts = f contextsS,
         status = f statusS,
         startStatus = if (SqlNull == startstatusS || toSql "-.-" == startstatusS)
                        then T.empty
                        else  f  startstatusS, --f startstatusS,
         endStatus = if (SqlNull == endstatusS || toSql "-.-" == endstatusS)
                      then T.empty
                      else  f endstatusS,
         username =if (SqlNull == usernameS)
                      then T.empty
                      else  f usernameS }-- f endstatusS}
convertSqlToStatus _ _ = error "not able to retrieve status"
```

Algorithm B.12: Module SessionHandling

```
{-# LANGUAGE ScopedTypeVariables #-}
module Database.SessionHandling (UserSession(..), isUserPassValid, createSession
                    ,isSessionActive, deleteSession ,getUserId) where
import Database.HDBC
import Database.HDBC.PostgreSQL
import Data.Text.Read
import Data.Maybe
import qualified Data.Text as T hiding (map, concat)
import qualified DataTransferObject.Status as S
import Database.DBUtils

type Username = String
type UserSession = (Username,String)

isUserPassValid :: Username -> String -> IO (Maybe String)
isUserPassValid username pass = do
  conn <- createConnection
  selectStatement<- prepare conn "select count(*) from users where name=? and pass=?"
  execute selectStatement [toSql username ,toSql pass]
  result <- fetchAllRows' selectStatement
  disconnect conn
  case fromSql . head . head $  result of
   1 -> return $ Just username
   _ -> return Nothing

getUserId :: Username -> IO (Maybe Int)
getUserId username = do
  conn <- createConnection
  field <- quickQuery' conn "select id from users where name=?" [toSql username]
  disconnect conn
  case decimal (fromSql . head . head $ field) of
   Left _ -> return Nothing
   Right a-> return $ Just $ fst a

createSession :: Username -> IO UserSession
createSession username = do
  conn <- createConnection
  userId <- getUserId username
  let statem = "update users SET active=?, \"activeTill\"=now()+ interval '1 hour'  ↩
where id=?"
  selectStatement<- prepare conn statem
  execute selectStatement [toSql $ T.pack "y",toSql userId] -- here the other id
  activeTill <- quickQuery' conn "select \"activeTill\" from users where id=?"
                                 [toSql userId]
  commit conn
  disconnect conn
  return  (username,fromSql . head . head $ activeTill)

deleteSession :: UserSession -> IO ()
deleteSession (username, _)= do
  conn <- createConnection
  userId <- getUserId username
  selectStatement<- prepare conn  "update users SET active=?,\"activeTill\"=? where id=?"
  execute selectStatement [toSql $ T.pack "n",SqlNull,toSql userId]
  commit conn
  disconnect conn
  return ()

isSessionActive :: UserSession -> IO (Maybe UserSession)
isSessionActive (username,_) = do
  conn <- createConnection
  userId <- getUserId username
  let ui = fromJust userId
      statem =  "select \"activeTill\">=now()+ interval '10 minutes',\"activeTill\"
                 from users where id=? and active=?"
  active <- quickQuery' conn statem [toSql ui, toSql "y"]
  disconnect conn
  if fromSql . head . head $ active
    then (return $ Just (username,fromSql . head. tail . head $ active))
    else (do a <- createSession username
             return $ Just a)
```

Algorithm B.13: Module ImmobilieProcessing, part 1

```
{-# LANGUAGE ScopedTypeVariables, OverloadedStrings, MultiParamTypeClasses #-}
module Database.ImmobilieProcessing (Point(..), retrieveImmobilie, retrieveContext
   ,addNewContext, retrieveBeschreibung, addNewBeschreibung, retrievePoint, updatePoint
   ,processedByUser, savePointString, retrieveAllContexts) where
import Database.HDBC
import Data.Text  hiding (map, concat,head)
import Data.Maybe
import Database.DBUtils
import Database.SessionHandling (getUserId)
import DataTransferObject.Immobilie

data Point  = Start  | End
isStartPoint:: Point  -> Bool
isStartPoint Start = True
isStartPoint End  = False

retrieveImmobilie :: Int ->IO Immobilie
retrieveImmobilie  id = do
   allrecs<-selectS (convertSqlToImmobilie fromSql) statement
  return . head $ allrecs
 where statement = "select id,plz,ort,strasse,land,bundesland,etage
,userdefinedsimplified,objekttitel,objektbeschreibung from immobilie where id="++ show id

retrieveContext :: Int -> IO (String)
retrieveContext id = retrieveFieldFrom id selectStatement
  where selectStatement= "select contexts from near where id=?"

addNewContext :: Int-> String -> IO (String)
addNewContext ctxId ctx=updateFieldFrom ctxId ctx "update near SET contexts=? where id=?"

retrieveAllContexts :: IO [Text]
retrieveAllContexts = selectS convertSqlToContext statement
 where statement= "select distinct contexts from near order by contexts"

convertSqlToContext ::  [SqlValue] -> Text
convertSqlToContext  =  fromSql. head

retrieveBeschreibung :: Int -> IO (String)
retrieveBeschreibung id =  retrieveFieldFrom id selectStatement
  where selectStatement= "select beschreibung from near where id=?"

addNewBeschreibung :: Int-> String -> IO (String)
addNewBeschreibung id beschreibung = updateFieldFrom id beschreibung statement
 where statement  =  "update near SET beschreibung=? where id=?"

retrievePoint :: Int -> Point-> IO String
retrievePoint id point= retrieveFieldFrom id statement
 where statement= if isStartPoint point
       then "select ST_AsGeoJSON(start) from near where id=?"
       else "select ST_AsGeoJSON(\"end\") from near where id=?"

updatePoint :: Int -> String -> String-> Point -> IO()
updatePoint id  x y point = do
  conn <- createConnection
  updateStatement <- prepare conn statement
  execute updateStatement [toSql id]
  commit conn
  disconnect conn
   where statement=  if isStartPoint point
    then "update near SET start=
          ST_PointFromText('POINT("++x++" "++y++")', 4326) where id=?"
    else "update near SET \"end\"=
          ST_PointFromText('POINT("++x++" "++y++")', 4326) where id=?"
```

Algorithm B.14: Module ImmobilieProcessing, part 2

```
-- * User input
processedByUser :: Int -> String -> IO ()
processedByUser id username = do
  conn <- createConnection
  userId <- getUserId username
  let uid = fromJust userId
  updateStatement <- prepare conn "update near Set userid=? where id=?"
  execute updateStatement [toSql uid ,toSql id]
  commit conn
  disconnect conn

savePointString :: Int -> Point -> String -> IO String
savePointString id Start text= updateFieldFrom id text
                               "update near SET \"startText\"=? where id=?"
savePointString id End text = updateFieldFrom id text
                               "update near SET \"endText\"=? where id=?"

selectStandardNahe :: String ->IO [Immobilie]
selectStandardNahe  = selectS (convertSqlToImmobilie fromSql)

convertSqlToImmobilie :: (b -> Text) ->  [b] -> Immobilie
convertSqlToImmobilie f [idS , plzS, ortS , strasseS, landS, bundeslandS,etageS
                        ,userDefinedSimplifiedS, objekttitelS, objektbeschreibungS
                         ] =
  Immobilie { idI = f idS
            , geo =toGeo f [plzS, ortS, strasseS, landS, bundeslandS, etageS
                           , userDefinedSimplifiedS]
            , freitexte= toFreitexte f [ objekttitelS, objektbeschreibungS]
}
convertSqlToImmobilie _ _ = error "not able to retrieve Immobilie"

toGeo ::(a->Text)-> [a]-> Geo
toGeo f [plzS, ortS, strasseS, landS, bundeslandS, etageS, userDefinedSimplifiedS]=
  Geo {plz = (f plzS)
      , ort =(f ortS)
      ,strasse=(f strasseS)
      ,land = (f landS)
      , bundesland = (f bundeslandS)
      , etage = (f etageS)
      , user_defined_simplified = (f userDefinedSimplifiedS)}
toGeo f _ = error "cannot convert SQLValues to Geo datatype"

toFreitexte ::(a->Text)-> [a]-> Freitext
toFreitexte f [objekttitelS, objektbeschreibungS]= Freitext
 {objekttitel= f objekttitelS, objektbeschreibung= f objektbeschreibungS}
```

Algorithm B.15: Module DBUtils

```
module Database.DBUtils (createConnection, retrieveFieldFrom, updateFieldFrom, selectS) where

import Database.HDBC
import Database.HDBC.PostgreSQL

createConnection =  connectPostgreSQL con
con = "host=127.0.0.1 dbname=spatialConceptNear user=postgres password=t"

-- * generics
retrieveFieldFrom :: Int -> String -> IO String
retrieveFieldFrom id selectStatement = do
  conn <- createConnection
  field <- quickQuery' conn selectStatement [toSql id]
  disconnect conn
  return .fromSql . head . head $ field

updateFieldFrom :: Int -> String -> String -> IO String
updateFieldFrom id updateValue updateStatement = do
  conn <- createConnection
  updateStatement <- prepare conn updateStatement
  execute updateStatement [toSql updateValue ,toSql id]
  commit conn
  disconnect conn
  return updateValue

selectS :: ([SqlValue] -> a) -> String -> IO [a]
selectS converter statement = do
  conn <- createConnection
  selectStatement <- prepare conn statement
  execute selectStatement []
  result <- fetchAllRows' selectStatement
--  print result
  disconnect conn
  return . map converter $ result
```

Algorithm B.16: Module Immobilie

```
module DataTransferObject.Immobilie (Immobilie(..), Geo(..), Freitext(..)) where

import  Data.Text hiding (map, concat)

data Immobilie = Immobilie {idI :: !Text
                           , geo :: Geo
                           , freitexte :: Freitext
                           } deriving Show;

applyfImmobilie ::(Text -> b) -> Immobilie -> [b]
applyfImmobilie f i = concat [applyfGeo f $ geo i
                             ,applyfFreitexte f $ freitexte i
                             ]

data Geo = Geo {plz :: !Text,ort :: !Text, strasse :: !Text, land :: !Text
 , bundesland:: !Text , etage :: !Text, user_defined_simplified :: !Text } deriving Show;

applyfGeo  ::(Text -> b) -> Geo -> [b]
applyfGeo f g = map f [plz g, ort g, strasse g, land g, bundesland g, etage g
                      , user_defined_simplified g]

data Freitext = Freitext {objekttitel :: !Text, objektbeschreibung :: !Text} deriving Show;

applyfFreitexte :: (Text -> b) -> Freitext -> [b]
applyfFreitexte f text = map f [objekttitel text, objektbeschreibung text]
```

Algorithm B.17: Module Status

```
module DataTransferObject.Status (Status(..)) where

import qualified  Data.Text as T hiding (map, concat)

data Status = S {idE :: !T.Text,
         beschreibung  :: !T.Text,
         contexts  :: !T.Text,
         status  :: !T.Text,
         startStatus  :: !T.Text,
         endStatus :: !T.Text,
         username :: !T.Text} deriving Show;
```

Algorithm B.18: Module Geocode

```
{-# LANGUAGE OverloadedStrings #-}
module Service.Geocode (geocodeText) where

import qualified Geocoder
import qualified Data.Text as T
import qualified Data.List as L

myDeveloperkey = T.pack "retrieve your key from opencage"

geocodeText :: Int -> String -> IO T.Text
geocodeText id geocodeString = do
 geocodingResult <-  Geocoder.geocode (T.pack geocodeString) myDeveloperkey
 let resultsInGeoJson = map (\a -> (resultToJson  a id)) geocodingResult
     geoJson = [
       T.pack "{\"type\": \"FeatureCollection\", \"crs\": {\"type\": \"name\",
               \"properties\": {\"name\": \"EPSG:4326\" } },\"features\": [" ]
               ++ (L.intersperse (T.pack ",") resultsInGeoJson)
               ++ [T.pack "]}" ]
 return $ T.concat geoJson

resultToJson  :: Geocoder.Result -> Int -> T.Text
resultToJson r id= T.concat [
   T.pack "{\"type\":\"", T.pack "Feature\""
   , T.pack ",\"geometry\":{\"type\":\"Point\""
   , T.pack ",\"coordinates\":["
      , T.pack $ show $ Geocoder.lng $ Geocoder.geometry r
      , T.pack ","
      , T.pack $ show $ Geocoder.lat $ Geocoder.geometry r
   , T.pack "]}"
   , T.pack ",\"properties\":{"
    , T.pack " \"Format\":\"",Geocoder.formatted r  ,"\""
    , T.pack ",\"Id\":\"", T.pack (show id),"\""
    , T.pack "}"
   , T.pack "}"]
```

Algorithm B.19: Module LogOnView

```
{-# LANGUAGE OverloadedStrings #-}
module View.LogOnView where

import Text.Blaze.Html5
import qualified Text.Blaze.Html5.Attributes as A

import Prelude hiding (head,div,span)

loadHeader :: Html
loadHeader = do
 head $ do
  title "Process the concept near, geocoding, context influence"
  link ! A.rel "stylesheet" ! A.type_ "text/css" ! A.href  "/css/mycss.css"
  link ! A.rel "stylesheet" !
       A.href "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css"
  link ! A.rel "stylesheet" ! A.href "/css/jquery-ui.css"
  script "" ! A.src "https://code.jquery.com/jquery-1.11.2.min.js"
  script "" ! A.src "//code.jquery.com/ui/1.11.4/jquery-ui.js"
  script "" ! A.src "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"
  script "" ! A.src "http://openlayers.org/en/v3.9.0/build/ol-debug.js"
  script "" ! A.src "/js/myquery.js"
  style $ ".modal-header, h4, .close {background-color: #5cb85c; color:white !important;
        text-align: center;font-size: 30px;}.modal-footer { background-color: #f9f9f9;}"

renderLoginPage :: Html
renderLoginPage = do
 loadHeader
 body ! A.onload "$(\"#myModal\").modal();"$ do
  div ! A.class_ "modal fade" ! A.id "myModal"$ do
    div ! A.class_ "modal-dialog" $ do
     div ! A.class_ "modal-content" $ do
      div ! A.class_ "modal-header"! A.style "style=padding:35px 50px;" $ do
        button ! A.type_ "button" ! A.class_ "close" $ "X"
        h4 $ span ! A.class_ "glyphicon glyphicon-lock" $ "Login"
      div ! A.class_ "modal-body" $ do
       form ! A.class_ "form col-md-12 center-clock"  ! A.action "/login"
            ! A.method "post"$ do
        div ! A.class_ "form-group" $ do
         label ! A.for "username" $ span ! A.class_ "glyphicon glyphicon-user"
               $ "Username"
         input ! A.type_ "text" ! A.class_ "form-control input-lg"
               ! A.placeholder "Username" ! A.name "username" ! A.id "username"
          div ! A.class_ "form-group" $ do
           label ! A.for "password" $ span
                 ! A.class_ "glyphicon glyphicon-eye-open" $ "Password"
           input ! A.type_ "password" ! A.class_ "form-control input-lg"
                 ! A.placeholder "Password" ! A.name "password" ! A.id "password"
          div ! A.class_ "form-group" $ do
            button ! A.class_ "btn btn-success btn-block" ! A.type_ "submit" $
                     span ! A.class_ "glyphicon glyphicon-off" $ "Login"
      div ! A.class_ "modal-footer" $ do
        div ! A.class_ "col-md-12" $ button ! A.class_ "btn" $ "Cancel"
```

Algorithm B.20: Module ProcessView, part 1

```
{-# LANGUAGE OverloadedStrings #-}
module View.ProcessView where

import Text.Blaze.Html5
import Text.Blaze.Html.Renderer.Text
import qualified Text.Blaze.Html5.Attributes as A
import qualified Data.Text as T
import Prelude hiding (head,div,span)
import DataTransferObject.Status
import qualified DataTransferObject.Immobilie as Immo

loadHeader :: Html
loadHeader = do
 head $ do
  title "Process the concept near, geocoding, context influence"
  link ! A.rel "stylesheet" ! A.type_ "text/css" ! A.href  "/css/mycss.css"
  link ! A.rel "stylesheet"
       ! A.href "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css"
  link ! A.rel "stylesheet" ! A.href "/css/jquery-ui.css"
  script "" ! A.src "https://code.jquery.com/jquery-1.11.2.min.js"
  script "" ! A.src "//code.jquery.com/ui/1.11.4/jquery-ui.js"
  script "" ! A.src "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"
  script "" ! A.src "http://openlayers.org/en/v3.9.0/build/ol-debug.js"
  script "" ! A.src "/js/myquery.js"

createOverview :: [Status] -> [Status] -> [Status] -> Html
createOverview todo progress finished = do
 loadHeader
 body $
  do nav ! A.class_ "navbar navbar-default navbar-fixed-top"$ createNavigationbar
     div ! A.class_ "panel panel-default"  $ do -- ! A.id ""
        div ! A.class_ "panel-heading" $ "Beschreibung"
        div ! A.class_ "panel-body"  $ renderBeschreibung
     div ! A.class_ "panel panel-default" ! A.id "todo" $ do
        div ! A.class_ "panel-heading" $ "todo"
        div ! A.class_ "panel-body"  $ statusToTable todo
     div ! A.class_ "panel panel-default" ! A.id "inwork" $ do
        div ! A.class_ "panel-heading" $ "inWork"
        div ! A.class_ "panel-body"  $  statusToTable progress
     div ! A.class_ "panel panel-default"  ! A.id "finished" $do
        div ! A.class_ "panel-heading" $ "finished"
        div ! A.class_ "panel-body"  $  statusToTable finished

renderBeschreibung :: Html
renderBeschreibung = do
  div ! A.class_ "text" $ do
    h1 "Fragestellung und Methodik"
    -- removed here described in the text

createNavigationbar :: Html
createNavigationbar = do
    div ! A.class_ "container-fluid" $ do
      div ! A.class_ "navbar-header" $ a ! A.class_ "navbar-brand" ! A.href "#"
          $ toHtml $ T.pack "Beschreibung"
      ul ! A.class_ "nav navbar-nav" $ do
        li $ a ! A.href "#todo" $ toHtml $ T.pack  "Todo"
        li $ a ! A.href "#inwork" $ toHtml $ T.pack  "inWork"
        li $ a ! A.href "#finished" $ toHtml $ T.pack  "Finished"
      ul ! A.class_ "nav navbar-nav navbar-right" $ do
        li $ a ! A.href "/logout" $ span ! A.class_ "glyphicon glyphicon-user"
          $ "Logout"
```

Algorithm B.21: Module ProcessView, part 2

```
statusToTable:: [Status]-> Html
statusToTable entries = do
 table ! A.class_ "table table-hover" $ do
  thead $ tr  $ mapM_ (th . toHtml. T.pack ) ["ID", "beschreibung","Start Punkt",
                                              "End Punkt", "Contexts","Bearbeiter"]
  tbody  $ tr $ mapM_  (\e -> tr $ statusEntryToTable e) entries

statusEntryToTable :: Status -> Html
statusEntryToTable s = do
  td . toHtml . idE $ s
  td $  addBewertungView$ beschreibung $ s
  td $ addPointView $ startStatus $s
  td $ addPointView $endStatus $s
  td . toHtml . contexts $s
  td . toHtml . username $ s
  td $
    form ! A.class_ "form-inline" ! A.action "/inprocess" ! A.method "get" $ toHtml $ do
        div ! A.class_ "form-group" $ do
         input ! A.type_ "hidden" ! A.name "id" ! A.value (toValue $ idE s )
         input  ! A.class_ "form-control" ! A.type_ "submit" ! A.value "bearbeiten"


addBewertungView  ::T.Text ->  Html
addBewertungView   bewer
  | (bewer == "not\_located" || bewer == "partly\_located" || bewer=="located") =do
  div ! A.class_ "btn-group" $ do
    button ! A.class_ (if bewer == "not\_located" then  "btn btn-success"
       else "btn btn-default") ! A.id "bewertung-not\_located" $ span $  "not\_located"
    button ! A.class_ (if bewer == "partly\_located" then  "btn btn-success"
       else "btn btn-default") ! A.id "bewertung-partly\_located" $ span $  ↩
"partly\_located"
    button ! A.class_ (if bewer == "located" then  "btn btn-success"
       else  "btn btn-default") ! A.id "bewertung-located" $ span $  "located"
   | otherwise =toHtml bewer

addPointView  ::T.Text -> Html
addPointView s =div ! A.class_ "btn-group" $ do
    button ! A.class_ (if s == T.empty then  "btn btn-default"
       else  "btn btn-success") ! A.id "point" $ span $
       (if s == T.empty then  "todo" else  "verortet")

immosNearToTable :: [Immo.Immobilie]-> Html
immosNearToTable entries = do
  table  ! A.class_ "table-hover" $ do
    thead . tr . mapM_ (th . toHtml. T.pack ) $ ["ID", "Objekttitel", "Status"]
    tbody . tr . mapM_  (tr . immoNearToTable ) $ entries

immoNearToTable :: Immo.Immobilie -> Html
immoNearToTable immo = do
  td . toHtml .  Immo.idI $ immo
  td . toHtml . Immo.objekttitel . Immo.freitexte $ immo
  td $
    form ! A.class_ "form-inline" ! A.action "/inprocess" ! A.method "get" $ toHtml $ do
        div ! A.class_ "form-group" $ do
         input ! A.type_ "hidden"  ! A.name "id"  ! A.value (toValue. Immo.idI $ immo )
         input  ! A.class_ "form-control" ! A.type_ "submit" ! A.value "bearbeiten"

idToHtml :: Immo.Immobilie -> Html
idToHtml e = do
  td  ( toHtml $ Immo.idI e)
  td  (geoToHtml $ Immo.geo e)
  td  ( toHtml . Immo.objekttitel .Immo.freitexte $ e)

geoToHtml :: Immo.Geo -> Html
geoToHtml e = do
 toHtml $ Immo.plz e
 td $ toHtml $ Immo.ort e
 td $ toHtml $ Immo.strasse e
```

Algorithm B.22: Module ImmobilieView, part 1

```
{-# LANGUAGE OverloadedStrings #-}
module View.ImmobilieView (renderImmobilie)where
import Text.Blaze.Html5
import qualified Text.Blaze.Html5.Attributes as A
import qualified Data.Text as T
import Text.Blaze.Html.Renderer.Text
import Data.List as L hiding (head,span)
import Prelude hiding (head,div,span)
import DataTransferObject.Immobilie

loadHeader :: Html
loadHeader = do
 head $ do
  title "Process the concept near, geocoding, context influence"
  script "" ! A.src "/js/jquery.autocomplete.js"
  script "" ! A.src "/js/jquery.highlight-5.js"
  -- same includes and scripts as in ProcessView

renderImmobilie :: Immobilie -> String-> String-> Html
renderImmobilie immobilie contexts bewertung= toHtml $ do
 loadHeader
 body ! A.onload "init();"$ do
  div ! A.class_ "container well" ! A.id "map" $ ""
  div ! A.class_ "panel-group" ! A.id "panelgroup" $ do
   div ! A.class_ "panel panel-default"! A.id "BeschreibungDiv"
                  $ divBeschreibung immobilie contexts
   div ! A.class_ "panel panel-default"! A.id "StartPointDiv" $ divStartPoint immobilie
   div ! A.class_ "panel panel-default"! A.id "EndPointDiv" $ divEndPoint immobilie
   div ! A.class_ "panel panel-default"! A.id "FinishedDiv"
                 $ divfinished immobilie contexts bewertung
 div ! A.id "dialog-confirm" !  A.style "float:left; margin:0 17px 20px 0;"
     ! A.title "Um welchen Punkt handelt es sich?" $ do
     p $ span ! A.class_  "ui-icon ui-icon-alert"  $ "the deleted?"

createPointDialog :: Html
createPointDialog =
    div ! A.class_ "modal-dialog modal-lg" $
      div ! A.class_ "modal-content" $ do
        div ! A.class_ "modal-header" $ do
          button ! A.type_ "button" ! A.class_ "close" $ "&times;"
          h4 ! A.class_ "modal-title" $ "Um welchen Punkt handelt es sich?"
        div ! A.class_ "modal-body" $ p $ "body test"
        div ! A.class_ "modal-footer" $ do
          button ! A.id "StartpunktButton" ! A.type_ "submit"
                 ! A.class_ "btn btn-default" $ "Startpunkt"
          button ! A.id "EndpunktButton" ! A.type_ "submit"
                 ! A.class_ "btn btn-default" $ "Endtpunkt"

divBeschreibung :: Immobilie -> String -> Html
divBeschreibung immobilie contexts=do
  div ! A.class_ "panel-heading" ! A.id "BeschreibungHeader"
    $ preEscapedToHtml . objekttitel .freitexte $ immobilie
  div ! A.class_ "panel-body" ! A.id "BeschreibungContent" $ span
    $ preEscapedToHtml . objektbeschreibung .freitexte $ immobilie
  div ! A.class_ "panel-footer" $ addContextView immobilie contexts

addContextView :: Immobilie -> String -> Html
addContextView immobilie contexts=do
   label ! A.for ("context") $ "context:"
   input ! A.class_ "form-control" ! A.type_ "text" ! A.id "context"
         ! A.value (toValue contexts)
   let jsMethod = toValue ("return addContext("++(show $idI immobilie)++")")
   input ! A.id "contextButton" !A.class_ "form-control btn btn-default"
         ! A.type_ "button" ! A.onclick jsMethod ! A.value "add Context"
```

Algorithm B.23: Module ImmobilieView, part 2

```
divStartPoint :: Immobilie -> Html
divStartPoint immobilie = do
 div ! A.id "geocode start id"! A.class_ "panel-heading" $ "Startpunkt:"
 div ! A.class_ "panel-body" ! A.id "StartPointContent"
    $ geocodePoint immobilienId geocodeString "start"
 where geoEntry = geo immobilie
       geocodeString = (T.concat [plz geoEntry,T.pack ",",  ort geoEntry
                                 , T.pack ",", strasse geoEntry ])
       immobilienId =  idI immobilie

divEndPoint :: Immobilie -> Html
divEndPoint immobilie = do
 div  ! A.id "geocode end id" ! A.class_ "panel-heading" $ "Endpunkt:"
 div ! A.class_ "panel-body"!A.id "EndPointContent"$geocodePoint (idI immobilie) "" "end"

geocodePoint::T.Text-> T.Text -> String ->  Html
geocodePoint idE geocodeString method=
 form ! A.class_ "form-inline from-group" $ do
   div ! A.class_ "form-group" $ do
    let textId = if method == "start" then "startGeocodeId" else "endGeocodeId"
    input !A.class_ "form-control"!A.type_ "input"!A.id textId!A.value
          $toValue geocodeString
    let jsMethod = if method == "start"
                   then toValue ("return saveGeocodeStart("++(show idE)++")")
                   else toValue ("return saveGeocodeEnd("++(show idE)++")")
    div ! A.class_ "btn-group" $ do
      let textId = if method == "start" then "startGeocodeId" else "endGeocodeId"
          jsMethod2 = if method == "start"
                   then toValue ("return saveGeocodeStartString("++(show idE)++")")
                   else toValue ("return saveGeocodeEndString("++(show idE)++")")
      button !A.class_ "form-control btn btn-default" ! A.type_ "button"
 ! A.id (toValue $ T.pack (textId++"-button")) ! A.onclick jsMethod $ span $ "verorten"
      button !A.class_ "form-control btn btn-default" ! A.type_ "button"
 ! A.id (toValue $ T.pack (textId++"-save-button")) ! A.onclick jsMethod2 $ span $ ↩
"speichern"

divfinished :: Immobilie ->String->String-> Html
divfinished immobilie contexts bewertung=do
  div ! A.class_ "panel-heading" $ "Abschluss"
  div ! A.class_ "panel-body"  $ addBewertungView immobilie bewertung
  div ! A.class_ "panel-footer" $ do
        form  ! A.action "/finished" ! A.method "get" $ do -- ! A.class_ "form-inline"
          div ! A.class_ "form-group" $ do
           input !A.class_ "" ! A.type_ "hidden" ! A.name "id" ! A.value (toValue $ ↩
idI immobilie )
           input !A.class_ "btn btn-default" ! A.type_ "submit" ! A.value "finished"
        form  ! A.action "/overview" ! A.method "get" $do -- ! A.class_ "form-inline"
         div ! A.class_ "form-group" $
          input ! A.class_ "btn btn-default" ! A.type_ "submit" ! A.value "zurueck"

addBewertungView :: Immobilie->String ->  Html
addBewertungView immobilie bewer=
  div ! A.class_ "btn-group" $ do
    button ! A.class_ (if bewer == "not\_located" then  "btn btn-success"
     else  "btn btn-default") ! A.id "bewertung-not\_located"
     ! A.onclick (toValue ("return addBewertung("++
                 (show $idI immobilie)++",'not\_located')") ) $ span $  "not\_located"
    button ! A.class_ (if bewer == "partly\_located" then  "btn btn-success"
     else  "btn btn-default") ! A.id "bewertung-partly\_located"
     ! A.onclick (toValue ("return addBewertung("++
                 (show $idI immobilie)++",'partly\_located')") ) $ span $  ↩
"partly\_located"
    button ! A.class_ (if bewer == "located" then  "btn btn-success"
     else  "btn btn-default") ! A.id "bewertung-located"
     ! A.onclick (toValue ("return addBewertung("++
                 (show $idI immobilie)++",'located')") ) $ span $  "located"
```

Algorithm B.24: Cascading style sheet document mycss.css

```
#map {z-index:0;width:100%;}
#panelgroup {width:30%; position:fixed; position:absolute;top:0px; left:0px; z-index: 1; }
#BeschreibungContent {height:200px;overflow-y:scroll;z-index: 2;}
#todo {height:100%;overflow-y:scroll;}
#inwork {height:100%;overflow-y:scroll;}
#finished {height:100%;overflow-y:scroll;}
#startGeocodeId{width:100%;}
#endGeocodeId{width:100%;}

.ol-scale-line {position: absolute;right:10px;background: black;  padding: 5px;}
.autocomplete-suggestions { border: 1px solid #999; background: #FFF; overflow: auto; }
.autocomplete-suggestion { padding: 2px 5px; white-space: nowrap; overflow: hidden; }
.autocomplete-selected { background: #F0F0F0; }
.autocomplete-suggestions strong { font-weight: bolder; color: rgb(92, 184, 92); }
.highlight {background-color: rgb(92, 184, 92); color:rgb(255, 255, 255);}
```

Algorithm B.25: JavaScript document myquery.js, part 1

```
var localhost = 'http://localhost:3000';

var hostURL= localhost;

var map;
function init() {
  map = new ol.Map({
    layers: [
      new ol.layer.Tile({
        source: new ol.source.OSM()
      })
    ],
    target: 'map',
    controls: ol.control.defaults({
      attributionOptions: ({
        collapsible: false
      })
    }),
    view: new ol.View({
      projection: 'EPSG:3857',
      center: ol.proj.transform([16.3779201, 48.1784762], 'EPSG:4326', 'EPSG:3857'),
      zoom: 12
    })
  });
////////////////////// the autocomplete function
  var autocompleteContexts= autocompleteContexts = [
    { value: 'Mariahilferstrasse', data: 'MH' },
    { value: 'Zimbabwe', data: 'ZZ' }
    ];
  $.ajax({
    type: 'GET',
    url: hostURL +'/autocomplete/contexts',
    success: function(data) {
      var getContexts = eval('(' + data + ')');
      if (getContexts.suggestions == 0) {
      } else {
        $('#context').autocomplete({
          lookup: getContexts.suggestions,
        });
      }
    }
  });

////////////////////////// highlight the words:
  $('#BeschreibungContent').highlight('nahe');
  $('#BeschreibungHeader').highlight('nahe');
  $('#BeschreibungContent').highlight('naehe');
  $('#BeschreibungHeader').highlight('naehe');
/////
  map.on('singleclick', function(e) {
    var pixel = map.getEventPixel(e.originalEvent);
    var feature = map.forEachFeatureAtPixel(e.pixel,
                                             function(feature, layer) {
                                               return feature;
                                             });
    var geometry;
    if (feature) {
      geometry = feature.getGeometry().getCoordinates();
    } else {
      geometry = map.getEventCoordinate(e.originalEvent);
    }
    whichPoint(geometry);
  });
  var startpoint= getPoint("start");
  var endpoint= getPoint("end");
  map.addControl(new ol.control.ZoomSlider());
}
```

Algorithm B.26: JavaScript document myquery.js, part 2

```
function whichPoint(geometry){
  $(function() {
    $("#dialog-confirm").dialog({
      resizable: false,
      height: 140,
      modal: true,
      buttons: {
        "is Startpoint": function() {
          ajaxPointSave(geometry,"saveStartPoint");
          $(this).dialog("close");
        },
        "is Endpoint": function() {
          ajaxPointSave(geometry,"saveEndPoint");
          $(this).dialog("close");
        }
      }
    });
  });
}

function getPoint(kind){
  var ctxid =  $('input[name=id]:hidden').val();
  var url;
  if (kind=="start") {
    url= hostURL+'/getStartPoint';
  }else{
    url=hostURL+'/getEndPoint';
  }
  $.ajax({
    type: 'GET',
    data: {
      id: ctxid
    },
    url: url,
    success: function(data) {
      if (data.length == 0) {
      } else {
        var iconFeatures=[];
        var point = new ol.format.GeoJSON().readGeometry(data, {
          featureProjection: 'EPSG:3857'
        });
        var iconFeature = new ol.Feature({
          geometry: new ol.geom.Point(point.getCoordinates())
        });
        iconFeatures.push(iconFeature);
        var iconsrc;
        if (kind == "start" ){
          iconsrc =hostURL+'/images/letter_a1.png';
        }else{
          iconsrc= hostURL+'/images/letter_e1.png';
        }
        addMarker(iconFeatures, iconsrc);
      }
    }
  });
}
```

Algorithm B.27: JavaScript document myquery.js, part 3

```
function ajaxPointSave(coordinates,startOrEnd){
  var ctxid =  $('input[name=id]:hidden').val();
  var saveUrl = hostURL+'/'+startOrEnd;
  var reprojected = ol.proj.transform(coordinates, 'EPSG:3857', 'EPSG:4326' );
  $.ajax({
    type: 'GET',
    data: {
      X:reprojected[0],
      Y:reprojected[1],
      id: ctxid
    },
    url: saveUrl,
    success: function(data) {
      var iconFeatures = []
      var iconFeature = new ol.Feature({
        geometry: new ol.geom.Point(coordinates)
      });
      iconFeatures.push(iconFeature);
      var iconsrc;
      if (startOrEnd == "saveStartPoint" ){
        iconsrc =hostURL+'/images/letter_a1.png';
      }else{
        iconsrc= hostURL+'/images/letter_e1.png';
      }
      addMarker(iconFeatures, iconsrc);
    }
  });
}

function addContext(id) {
  var ctx = $('input#context').val();
  $.ajax({
    type: 'GET',
    data: {
      context: ctx,
      id: id
    },
    url: hostURL+'/addContext',
    beforeSend: function() {
      $('input#contextButton').addClass("btn-info").removeClass("btn-default");
    },
    success: function(data) {
      $('input#' + id).val(data);
      $('input#contextButton').addClass("btn-success").
        removeClass("btn-default btn-info");
    },
    error: function(error) {
      $('input#contextButton').removeClass("btn-default btn-success btn-info").
       addClass("btn-danger");
    }
  });
}
```

Algorithm B.28: JavaScript document myquery.js, part 4

```
function addBewertung(id,bewertung) {
  $.ajax({
    type: 'GET',
    data: {
      bewertung: bewertung,
      id: id
    },
    url: hostURL+'/addBewertung',
    success: function(data) {
    if (bewertung == "partly\_located" ){
    $('button#bewertung-partly\_located').removeClass("btn-success").addClass("btn-default");
    $('button#bewertung-located').removeClass("btn-success").addClass("btn-default");
    $('button#bewertung-partly\_located').removeClass("btn-default").addClass("btn-success");
    } else if (bewertung == "not_located" ){
    $('button#bewertung-partly\_located').removeClass("btn-success").addClass("btn-default");
    $('button#bewertung-located').removeClass("btn-success").addClass("btn-default");
    $('button#bewertung-not\_located').removeClass("btn-default").addClass("btn-success");
    }else{
    $('button#bewertung-not\_located').removeClass("btn-success").addClass("btn-default");
    $('button#bewertung-partly\_located').removeClass("btn-success").addClass("btn-default");
    $('button#bewertung-located').removeClass("btn-default").addClass("btn-success");
    }
    }
  });
}

function saveGeocodeStart(id) {geocode(id, 'startGeocodeId');}

function saveGeocodeEnd(id) { geocode(id, 'endGeocodeId');}

function geocode(idToGeocode, field) {
  var text = $('input#' + field).val();
  var result;
  $.ajax({
    type: 'GET',
    data: {
      toGeocode: text,
      id: idToGeocode
    },
    url: hostURL+'/geocode',
    dataType: 'json',
    beforeSend: function() {
      if (field == 'endGeocodeId') {
        $('button#endGeocodeId-button').removeClass("btn-default btn-success")
        .addClass("btn-info");
      }else{
        $('button#startGeocodeId-button').removeClass("btn-default btn-success")
        .addClass("btn-info");
      }
    },
    success: function(data) {
      if (data.length == 0) {
        alert("Geocodierung lieferte kein Ergebnis");
      } else {
        var iconsrc = hostURL+'/images/letter_a.png ';
        if (field == 'endGeocodeId') {
          iconsrc = hostURL+'/images/letter_e.png ';
          $('button#endGeocodeId-button').removeClass("btn-default btn-info")
          .addClass("btn-success");
        }else{
          $('button#startGeocodeId-button').removeClass("btn-default btn-info")
          .addClass("btn-success");
        }
        var iconFeatures = (new ol.format.GeoJSON()).readFeatures(data, {
          featureProjection: 'EPSG:3857'
        });
        addMarker(iconFeatures, iconsrc);
      }
    }
  });
}
```

Algorithm B.29: JavaScript document myquery.js, part 5

```
function saveGeocodeStartString(id){ var text = $('input#startGeocodeId').val();
  saveGeocodeString(id,text,"Start");
}

function saveGeocodeEndString(id){ var text = $('input#endGeocodeId').val();
  saveGeocodeString(id,text,"End");
}

function saveGeocodeString(idToSave, text,pointtype) {
  var result;
  $.ajax({
    type: 'GET',
    data: {
      id: idToSave,
      elementString: text,
      pointtype : pointtype
    },
    url: hostURL+'/saveGeocodeString',
    dataType: 'json',
    beforeSend: function() {
      if (pointtype == 'End') {
        $('button#endGeocodeId-save-button').removeClass("btn-default")
        .addClass("btn-info");
      }else{
        $('button#startGeocodeId-save-button').removeClass("btn-default")
        .addClass("btn-info");
      }
    },
    success: function(data) {
      if (data.length == 0) {
        alert("kein Ergebnis von der geocodierung");
      } else {
        if (pointtype == 'End') {
          $('button#endGeocodeId-save-button').removeClass("btn-default btn-info")
          .addClass("btn-success");
        }else{
          $('button#startGeocodeId-save-button').removeClass("btn-default btn-info")
          .addClass("btn-success");
        }
      }
    },
    error : function(data){
      if (pointtype == 'End') {
        $('button#endGeocodeId-save-button').removeClass("btn-default btn-info")
        .addClass("btn-danger");
      }else{
        $('button#startGeocodeId-save-button').removeClass("btn-default btn-info")
        .addClass("btn-danger");
      }
    }
  });
}

function addMarker(iconFeatures, markerurl){
  var vectorSource = new ol.source.Vector({ features: iconFeatures });
  var iconStyle = new ol.style.Style({
    image: new ol.style.Icon(({
      anchor: [0.5, 46],
      anchorXUnits: 'fraction',
      anchorYUnits: 'pixels',
      opacity: 0.75,
      src: markerurl
    }))
  });
  var vectorLayer = new ol.layer.Vector({
    source: vectorSource,
    style: iconStyle
  });
  map.addLayer(vectorLayer);
  var point = vectorSource.getFeatures()[0].getGeometry().getCoordinates();
  map.getView().setCenter(point);
}
```

B.4 Data analysis

The processed data are analyzed in order to remove false positives. The contexts are lemmatized and classified accorting to implied meaning. If entries are categorized as "located" but do not include necessary data *e.g.* the location of the locatum is missing, the entries are processed again. The distances between the relatum and locatum is calculated and checked for plausibility by looking at the resulting distances. If the distances are too high *e.g.* 20 kilometers the entries are checked if it the distances matches the description in the real estate entry. The results of the data analysis is given in Tables B.1, B.2, B.3 that include all identified contexts given in German language, the meaning and the number of occurrences classified in "not located", "partly located" and "located" entries.

B.5 Contextual typicality for "near"

Algorithm B.30 shows the algorithm used to calculate contextual typicality and prototypes for exemplars on a rational measurement scale using kernel density estimation (KDE).

Algorithm B.30: Instance ContextualizedConcept for exemplars on a rational measurement scale

```haskell
{-# LANGUAGE MultiParamTypeClasses, FlexibleInstances #-}

module ExemplarScales.RatioExemplars where

import Data.Function (on)
import qualified Data.Vector.Unboxed as U
import Statistics.Sample.KernelDensity (kde)
import qualified Data.List as List
import ContextualizedConcept
import Concept

instance ContextualizedConcept c Double where
  calculatePrototype =calculatePrototype' . createKDE . extractData

calculatePrototype' :: [(Double,Double)] -> (Double, Double)
calculatePrototype' = List.maximumBy (compare `on` snd)

createKDE :: [Double] -> [(Double,Double)]
createKDE rawdata =U.toList . uncurry U.zip . kde 64 $ dataVector
  where dataVector= U.fromList rawdata

extractData :: Concept c Double -> [Double]
extractData = map getExemplar . toObservationList
```

Using Algorithm B.30 the contextual typicalities and prototypes for **near** with more than five "located" categorized real estate entries are created. The contextualized concepts are illustrated in Figures where those contextualized concepts are grouped according to the distance where the prototype is included. The contexts are given in German language where a translation for each context is included in Table B.4. In summary eighteen contextual typicalities Figures including six distributions are given in Figure B.3, B.4, B.5, B.6, B.7, B.8.

context	meaning	number of occurrences		
		not located	partly located	located
		2618	201	952
Alm	space	1	0	0
Altstadt	space	0	1	1
Angehörige	relationship	3	0	0
Augarten	space	2	0	1
Auhof	space	0	1	0
Auskünfte	information	73	2	1
Autobahn	space	13	1	7
Badeteich	space	0	0	1
Bahn	space	3	0	0
Bahnhof	space	15	2	3
Bodensee	space	0	0	1
City	space	8	0	9
Details	information	8	0	0
Donau	space	3	0	1
Flughafen	space	2	0	1
Fußgängerzone	space	1	0	0
Getreidemarkt	space	0	0	1
Goldenen Stiege	space	1	0	0
Golfplatz	space	0	0	2
Graben	space	2	0	0
Grenze	space	0	1	1
Gürtel	space	0	0	1
Hauptbahnhof	space	0	0	1
Himmel	space	3	0	0
Hollabrunn	space	1	0	0
Horn	space	0	0	1
Informationen	information	79	10	0
Innenstadt	space	16	0	10
Interesse	information	12	1	0
Kagraner Platz	spcae	0	16	0
Kamp	space	1	0	0
Krankenhaus	space	4	0	2
Künstlerviertel	space	1	0	0
Küsten	space	0	0	1
Landeshauptstadt	space	0	0	1
Lobau	space	3	1	0
Luftlinien	spcae	0	0	1
Mariahilferstrasse	space	1	3	0

Table B.1: Number of context occurrences in the data sets from A-Ma

context	meaning	number of occurrences		
		not located	partly located	located
Meer	space	1	0	0
Mittelschule	space	1	0	0
Naherholung	space	2	0	1
Naheverhältnis	relationship	14	1	0
Naschmarkt	space	1	2	1
Naturschutzgebiet	space	1	0	0
Oper	space	3	0	0
Park	space	0	2	2
Post	space	0	1	0
S-Bahn	space	6	1	1
Schloss	space	0	0	1
See	space	18	3	19
Seilbahn	space	1	0	0
Siedlung	space	1	0	0
Sonnen	space	1	0	0
Stadt	space	20	6	25
Stadtpark	space	0	0	3
Stausee	space	0	0	1
Strand	space	3	0	1
Straßenbahn	space	1	0	0
Thermen	space	3	0	2
U-Bahn	space	18	3	12
Umfeld	space	2	0	1
Umgebung	space	11	5	4
Umkreis	space	1	0	0
Universität	space	6	1	5
Wald	space	5	1	2
Waldrand	space	1	0	0
Wien	space	6	0	1
Zentrum	space	113	12	78
Zentrum, City	space	0	0	2
Zentrum, U-Bahn	space	0	0	1
Zukunft	time	7	0	0
absolut, Zentrum	space	0	0	2
annähernd	space	3	1	0
annähernd eben	space	1	0	0
bei	space	11	6	4
boden	space	15	0	0

Table B.2: Number of context occurrences in the data sets from Me-b

context	meaning	number of occurrences		
		not located	partly located	located
direkter	space	0	0	2
endlos	space	1	0	2
ganz	space	5	1	35
gehen	space	62	7	67
gehen, liegen	space	0	0	4
greifen	space	71	0	2
grenz	space	0	1	1
großer	space	0	0	3
großstadt	space	0	0	1
ideal	space	0	0	1
liegen	space	131	66	447
liegen, Flughafen, Bahn	space	0	0	1
liegen, absolut, gehen	space	0	0	1
liegen, ganz	space	0	0	1
liegen, sehr	space	6	0	0
natur	space	10	7	12
nächster	space	0	2	7
ort	space	1	0	0
sehr	space	2	1	0
sehr, Zentrum	space	0	0	1
so	space	1	3	0
unmittelbar	space	133	34	271
unmittelbar, Altstadt	space	0	0	1
unmittelbar, Autobahn	space	4	0	0
unmittelbar, Luftlinie	space	0	0	5
unmittelbar, See	space	1	0	0
unmittelbar, Ufer	space	0	0	28
unmittelbar, Zentrum	space	0	0	1
unmittelbar, gehen	space	2	0	6
unmittelbar, liegen	space	0	0	2
urban	space	1	0	0
verlaufende	space	0	0	1
zu	space	28	7	4
zu, direkt	space	0	0	8

Table B.3: Number of context occurrences in the data sets from d-z

context in German language	context in English language
Autobahn	*highway*
Stadt	*city*
Innenstadt	*downtown, city center*
See	*lake*
Stadt	*Stadt (not translated)*
U-Bahn	*subway*
Universität	*university*
Zentrum	*center*
ganz	*realy*
gehen	*walk*
liegen	*located*
natur	*nature*
nächster	*next, close*
unmittelbar	*immediate*
unmittelbar, Luftlinie	*immediate, line of sight*
unmittelbar, gehen	*immediate walk*
unmittelbar, Ufer	*immediate, bank*
zu, direkt	*too, straight*

Table B.4: German- English translation of contexts presented in the resulting contextualized concepts for near

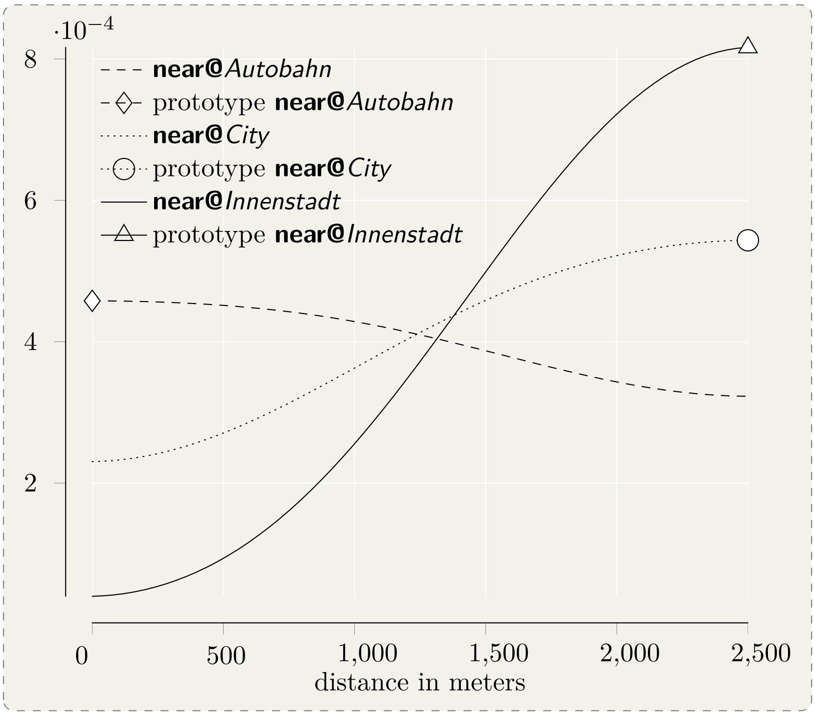


Figure B.3: Contextual typicality for **near@***highway*, **near@***city* and **near@***city center*

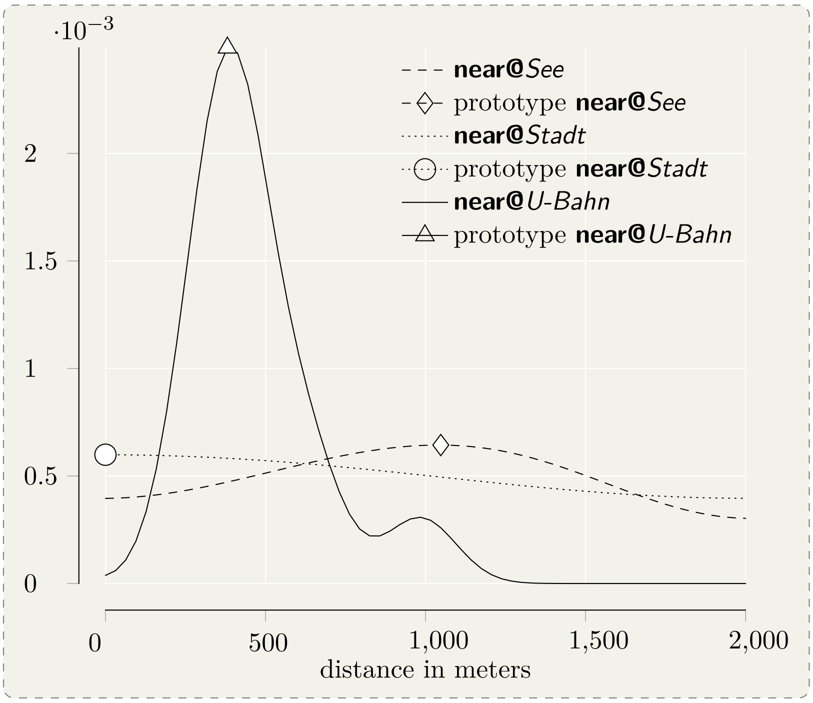


Figure B.4: Contextual typicality for **near@***lake*, **near@***city* and **near@***subway*

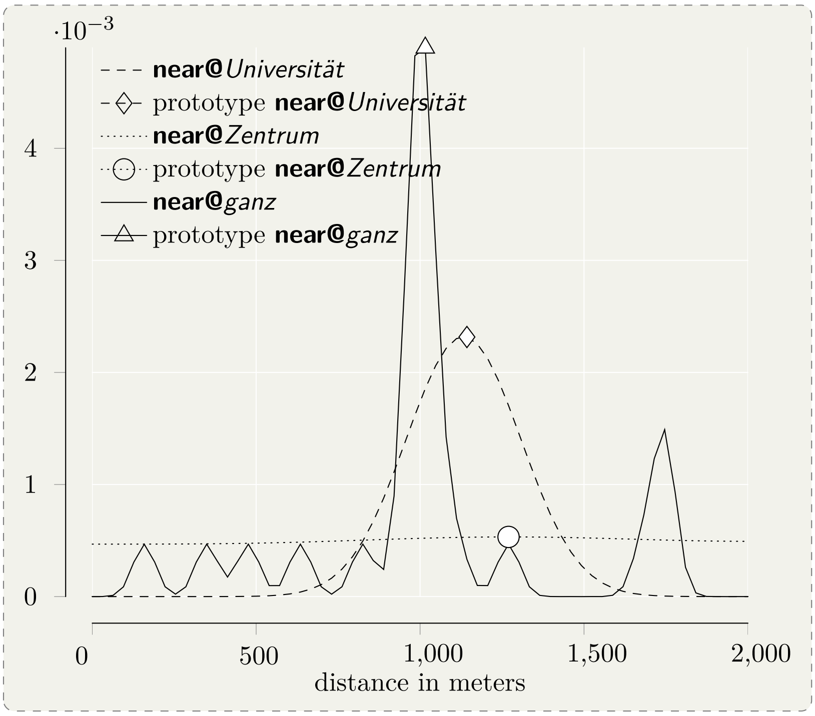


Figure B.5: Contextual typicality for **near@***university*, **near@***center* and **near@***really*

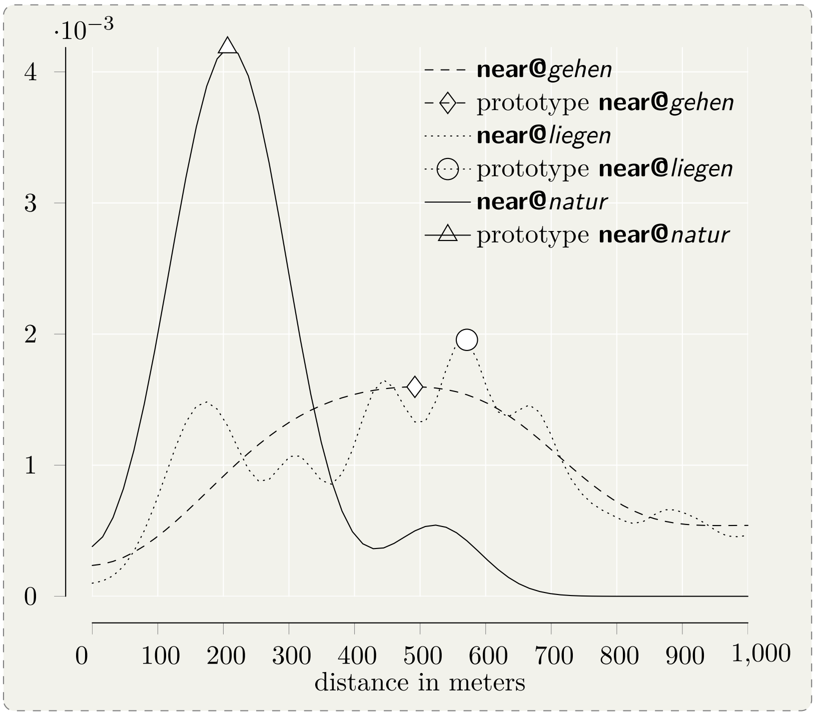


Figure B.6: Contextual typicality for **near@***walk*, **near@***located* and **near@***nature*

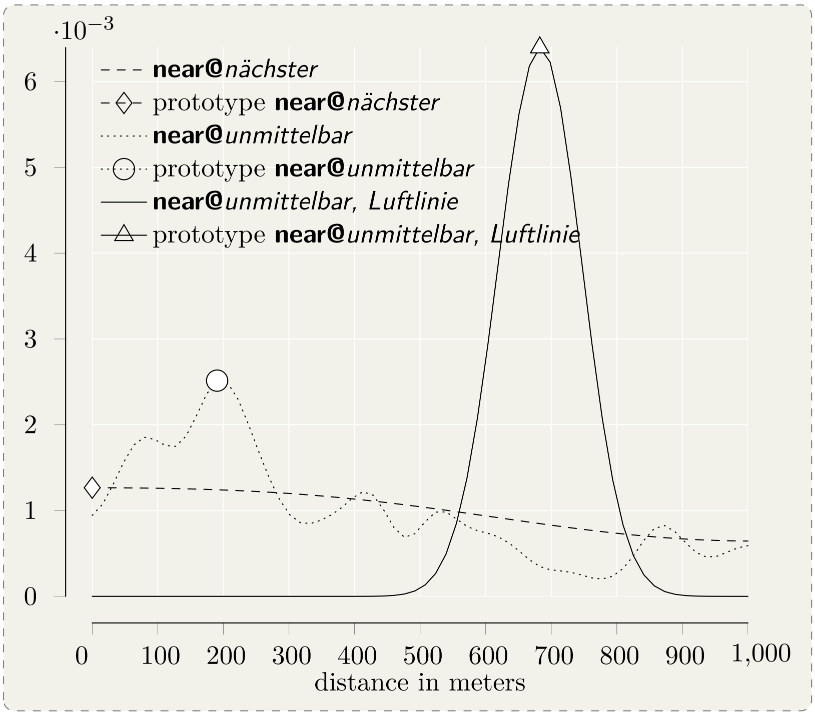


Figure B.7: Contextual typicality for **near@***next*, **near@***immediate* and **near@***immediate, line of sight*

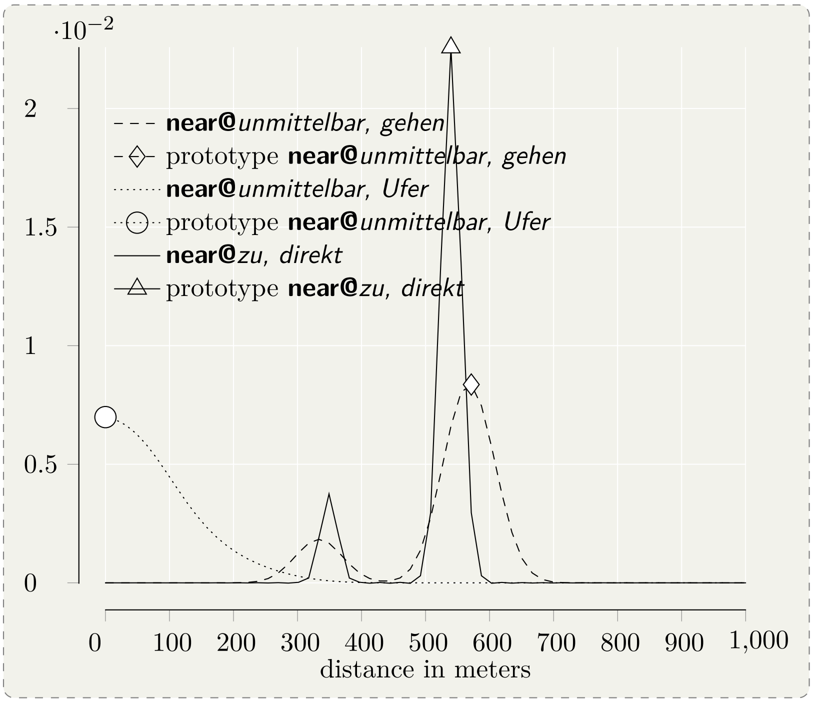
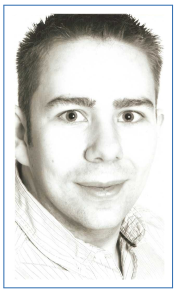


Figure B.8: Contextual typicality for **near@***immediate*, *walk*, **near@***immediate*, *bank* and **near@***too*, *straight*

Jürgen Hahn

Résumé

Kaiserstrasse 28/1/14
1070 Vienna
☎ *+43 664 55 31 682*
✉ *juergenhahn01@gmail.com*
hahnjuergen
juergenhah
in *juergenhahn01*
www.xing.com/profile/Juergen_Hahn29



I like challenges, but I love solutions!!

Working experience

08/2012-Present **Vienna University of Technology, Research Group Geoinformation**, *Academia*,
University Assistant (half time).

Research fields:

- Context modelling for GIS: Context algebra, Context model, extracting context from text descriptions.
- Quantum Interaction for GIS: Quantum theory used for context and communication coordination.
- Haskell for GIS: Developing a Haskell interface for a web geocoding service, Haskell for OpenLayers.
- Fuzzy vector mathematics for GIS: A model for spatial frame of references, Geodetic survey data.
- Game data extraction: Extraction of player trajectories and map data from a First Person Game.

Teaching and organization:

- GIS: Feasibility Study for a GIS (Lec. & Labs, Undergrad), Realisation of a GIS (Lec. & Labs, Undergrad), Geoinformation (Labs, Undergrad).
- Haskell: Haskell - Introduction for Geoinformatics (Lec. & Labs, Grad).
- Organization: Inventor of the 1st Workshop on Context-Awareness, Local organizor GIScience 2014.

02/2014-08/2015 **BROMsolutions Austria GmbH**, *Career Experts in SAP*,
Consultant - Freelance (on demand), Information and technology support, automatic backup of emails and files (cloud and local), administration of the network and computer hardware.

01/2011-07/2012 **IQSOFT GmbH**, *Geoinformation/ IT*,
Junior Consultant (full time).

- Including GIS in business processes of a railway company
  data import, object recognition, geocoding of rails, fusion of several data sources.
- Improving my knowledge in Java web libraries, extending insights about Java performance
  implementing web applications for a railway company, time and memory profiling of a 3rd party library.

01/2006-10/2006 **Stora Enso Timber AG**, *Wood Products*,
Electrician (full time), Maintenance of wood processing plants, fault management.

05/2005-12/2005 **Ing. Bruno Wildburger GmbH**, *Plumber and industrial clerk*,
Technical Illustrator (full time), Technical drawings of gas, water and heating systems.

Education

10/2011-Present **Vienna University of Technology**, *Geoinformation*,
PhD Candidate, Topic: "Context Algebra applied to Spatial Concepts.".

10/2006-12/2010 **University Center Rottenmann**, *GTEC-Geoinformation and Technology*,
Magister, Thesis title (translation): "Planning and Conducting interoperability tests and response time measurements of a Web Map Tile Service".

09/1999-06/2004 **HTBLuVA St. Pölten (Secondary Technical School)**, *Electrical engineering*,
A-level, Electrical/ Information Technology.

Fellowship

12/2014 **ETH Zurich, Geoinformation Engineering**, *Academia*,
Visiting Researcher, Research topics: order effects in spatial cognition, fuzzy vectors to model human frames of reference.

Supervised thesis

2016 **Study about advertisement revenues in respect of GIS**, *Bachelor thesis*,
Objective: How much would companies pay for GIS specific advertisement? (translation).

2016 **Automatic extraction of locations from Linked Data (translation)**, *Bachelor thesis*,
Objective: How can geographic locations encoded in linked data automatically be extracted and presented in a web map? (translation).

2015 **Haskell OpenLayers Wrapper**, *Master thesis*,
Objective: Can Haskell to Java Script cross compiled code be used to interact with OpenLayers? (translation).

2015 **Performance comparison of multiple JavaScript libraries for geodata visualisation (translation)**, *Bachelor thesis*,
Objective: Comparison of D^3, OpenLayers, Leaflet and Polymaps regarding memory consumption and response time for different amounts of geodata (translation).

2014 **Interfaces from Android & iOS to gather user information (translation)**, *Bachelor thesis*, Objective: Which smartphone interfaces can be used to extract context information from their users? (translation).

2014 **Is the Android-API usable for direct georeferencing of camera images? (translation)**, *Bachelor thesis*, Objective: Are smartphone sensors accurate to measure distances in multiple camera images? (translation).

2012 **Crime Prediction based on Spatial Analysis**, *Bachelor thesis*,
Objective: Find suitable algorithms to detact crime hotspots.

Internships

07/2008,09/2009 **Honauer Icon GmbH**, *Electric dispatcher assembling*,
Electrician (full time), Assembling electric dispatchers and installation on site.

05/2009-06/2009 **Vanilla Live Games GmbH**, *Game Publisher*,
Game Tester - Freelance (half time), Testing an online racing game, with focus on game mechanic and the bonus system.

08/2008,08/2009 **Freytag-Berndt und Artaria KG**, *Cartographic Publisher*,
Cartographer (full time), Establishing a new symbol set using ArcGIS based on MicroStation templates, cartographic work on hiking maps.

07/2007-08/2007 **Stora Enso Timber AG**, *Wood Products*,
Electrician (full time), Maintenance of wood processing plants, fault management.

Military service

09/2004-04/2005 **Panzerstabsbataillon 3**, Lower Austria, Spratzern.

Training courses and seminars

06/2014 BrainRead, Vienna University of Technology, Göran Askeljung.

01/2014 Successful negotiations (translation) - Erfolgreich verhandeln, Coverdale.

09/2013 Active Teaching at an University (translation) - Aktivierende Lehre and der Hochschule, Vienna University of Technology.

07/2013 Game ON: From Theory to Practice! BEST Vienna.

06/2013 Small Talk Conference Speak, Vienna University of Technology, Swantje Cooper.

06/2013 The art of free speech (translation) - Die Kunst des freien Sprechens, Vienna University of Technology, Beatrice I. Seum.

10/2012 Writing Scientific Texts, Vienna University of Technology, Mag. Elisabeth Hambrusch.

06/2012 Oracle Database 11g: Use XML DB Ed 2, Oracle University Training Centre - GNC Akademie GmbH.

07/2011 Oracle Spatial 11g: Essentials, Oracle University Training Centre - GNC Akademie GmbH.

07/2010 Summer school IT Outsourcing UZR.

Personality

Cuisine Cooking for my relatives and friends, italian and chinese cuisine.

Games Playing computer games, card games, jigsaw puzzles, inviting friends for dinner and crime or a pub quiz.

Technology Interested in Haskell, testing Linux distributions and Window managers.

Sports Cycling, running, hiking.

Languages

German Mother tongue.

English Fluent in reading, speaking and writing.

Technical skills

GIS QuantumGIS, Oracle Mapviewer, Open Street Map, GDAL, ESRI - ArcGIS, Safe FME, MicroStation.

Web GIS OpenLayers, GeoServer, OGC-Standards and Services, Google Maps API.

Database PostGIS, Oracle Spatial, Oracle, PostgreSQL.

Programming language Haskell, JavaScript, Java, PL/SQL, PHP, Python, C.

Haskell library Lattice, gnuplot, HTTP, parsec, HDBC, scotty, Aeson, blaze, snap, Fay, postgres-lite.

Java Hibernate, Spring, Appfuse, Servlets, JSP, JSF, Apache Tomcat, JBOSS.

IDE Emacs, Eclipse, Netbeans IDE, Oracle SQL Developer, Leksah.

Revision System Git, SVN.

Performance test JProfiler, Apache Jmeter.

Miscellaneous GnuPlot, AutoCAD.

Operating system Linux: Manjaro, Mint, Ubuntu; Windows.

Markup language GML, XML, HTML, CSS.

Office LaTeX, MS - Office.