# Representing Computational Relations in Knowledge Graphs Using Functional Languages

**Yanmin Qi**
School of Computer Science, University of Nottingham Ningbo, China
State Key Laboratory of Resources and Environmental Information System, Institute of Geographic Sciences and Natural Resources Research, Chinese Academy of Sciences, Beijing, China

**Heshan Du**
School of Computer Science, University of Nottingham Ningbo, China

**Amin Farjudian**
School of Computer Science, University of Nottingham Ningbo, China

**Yunqiang Zhu**
State Key Laboratory of Resources and Environmental Information System, Institute of Geographic Sciences and Natural Resources Research, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

## Abstract

Knowledge representation is the cornerstone of constructing a geoscience knowledge graph (GKG). The existing representations of spatial and computational relations in GKGs, however, are inadequate. In this paper, we use Dimensionally Extended Nine-Intersection Model (DE-9IM) to represent spatial topological relations. To represent computational relations, we use typed lambda calculus via its implementation in the functional language Haskell, in which functions are first-class primitives. We exemplify our ideas through some basic examples in Haskell.

**2012 ACM Subject Classification** Theory of computation → Semantics and reasoning

**Keywords and phrases** spatial relation, computational relation, functional programming, Haskell, geo-knowledge graph

**Digital Object Identifier** 10.4230/LIPIcs.COSIT.2022.29

**Category** Short Paper

**Funding** *Yunqiang Zhu*: National Natural Science Foundation of China [grant number 42050101]

## 1 Introduction

Knowledge graphs provide a paradigm for representing interconnected information derived from heterogeneous sources [4]. A geoscience knowledge graph (GKG) organizes geoscience knowledge with a structured graph, which is suitable for algorithmic processing [21]. The fundamental idea of establishing a GKG is inspired by the spatial and temporal features contained in geoscience phenomena and processes, and by knowledge representation models illustrated in various disciplines of geoscience [21]. Overall, multi-scale spatial and temporal features are the most significant characteristics of GKGs when compared with knowledge graphs in other disciplines [21].

The cornerstone of constructing a GKG is knowledge representation. Knowledge representation is closely connected with formal ontology, which deals with entities from the given world, together with their properties and relations between them [5]. Normally, two types of representation models are implemented: one being {entity, relation, entity} (e. g., {Ningbo City, within, Zhejiang Province}), and another being {entity, attribute, attribute value} (e. g., {Ningbo City, has population, 8.202 million}) [19]. Reasoning – the process of inferring conclusions from existing knowledge [3] – provides a salient purpose for knowledge representation. Geoscience knowledge reasoning is carried out over GKGs [21], and is used to shed light on the evolutionary features of geoscience knowledge systems.

It is, however, challenging to represent complex interdisciplinary geoscience knowledge by the existing representation models [21]. Another challenge is posed by the uniform integration of spatial, temporal, and computational relations in GKGs [21]. Three kinds of spatial relations are primarily used in geoscience, which are direction, distance, and topology [20]. Temporal relations signify how events relate to one another in time. Computational relations describe the process of using mathematical or logical methods to generate entities from other entities, and they normally include mathematical formulas and modelling procedures [21]. In the existing models, the representation of a computational relation only expresses the existence of the relation. The formula or logical rule underlying the computational relation, however, is not represented, because the models are not expressive enough.

In this paper, we use Dimensionally Extended Nine-Intersection Model (DE-9IM) to represent spatial topological relations. We use the functional programming language Haskell [8] to represent computational relations and perform reasoning about spatial topological relations. The paper is organized as follows: Section 2 describes the representation model of spatial relations. The Haskell implementations of spatial reasoning and representation of computational relations are exemplified in Section 3. Conclusions and directions for future work are presented in Section 4.

## 2 Representation of Spatial Topological Relations

According to the Open Geospatial Consortium (OGC), the DE-9IM [17] string code can be a standardized format for the data interchange of the typical spatial predicates [14]. DE-9IM represents the relationship between two spatial objects $A$ and $B$ based on the intersections of their interiors, boundaries, and exteriors. The spatial objects discussed here include points, lines, and 2D regions. For our discussion, the concepts of interior, boundary, and exterior are not the same as those defined in classical topology. For instance, for a given line, we consider the end-points as the boundary, and the (open) segment between the end-points as the interior. All the remaining points in the plane are regarded as the exterior. For a point, the interior is the point itself, the boundary is empty, and the exterior is the entire plane minus the point itself [17]. Dimension values are assigned to spatial objects as follows: 0 for points, 1 for lines, and 2 for 2D regions. All the above values (0, 1 and 2) are "TRUE" values in the spatial predicates as they signify non-empty sets. An empty set is represented by $-1$, which is the "FALSE" value in the spatial predicates. The eight spatial relationships involved in DE-9IM are "Crosses", "Within", "Contains", "Equals", "Disjoint", "Intersects", "Touches", and "Overlaps", as detailed in [17].

The spatial relations defined in DE-9IM have been implemented in function `ST_Relate`() of PostGIS [6]. For instance, the triple {Ningbo City, Touches, Shaoxing City} semantically describes that the relation between "Ningbo City" and "Shaoxing City" is "Touches". If we call `ST_Relate`("`NingboCity`", "`ShaoxingCity`") in PostGIS, the spatial relation "Touches" will be represented as $[FF2F11212]$, in which `F` represents "FALSE" $(-1)$.

For any particular spatial relation, the existing knowledge graphs use different semantic descriptions, which causes ambiguity. This problem is exacerbated by the ambiguity that is inherent in natural languages. For instance, describing entity $A$ as "nearTo" entity $B$ does not specify how far (say) in meters they are from each other. Therefore, the purpose of using relation matrixes represented by DE-9IM is to quantitatively represent spatial relations, which is more suitable for verified learning and inference of downstream models and reasoning [11].

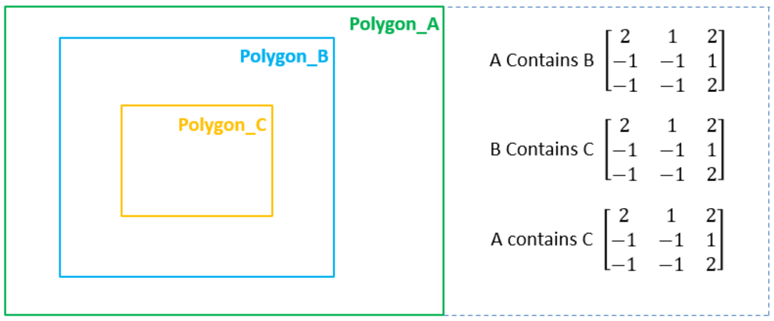
Polygon_A
Polygon_B
Polygon_C
A Contains B $\begin{bmatrix} 2 & 1 & 2 \\ -1 & -1 & 1 \\ -1 & -1 & 2 \end{bmatrix}$
B Contains C $\begin{bmatrix} 2 & 1 & 2 \\ -1 & -1 & 1 \\ -1 & -1 & 2 \end{bmatrix}$
A contains C $\begin{bmatrix} 2 & 1 & 2 \\ -1 & -1 & 1 \\ -1 & -1 & 2 \end{bmatrix}$

**Figure 1** Spatial topological relations among three objects.

## 3 Representation and Reasoning in Haskell

Haskell is a functional programming language, based on typed lambda calculus. As opposed to imperative languages, the syntax of functional languages resembles mathematical expressions more closely. As such, in functional programming, the focus is mainly on what is being computed, rather than low-level details of how it is computed [7]. Functional syntax has indeed been considered for representation and reasoning in the literature as well. For instance, Shahzad et al. [16] integrated Haskell with relational algebra to create a user model called Universe of Discourse (UoD). Leinberger et al. [9] defined and presented a functional language for handling semantic data. Haskell was also used for integrating conventional sensor information and volunteered geographic information [15].

For our purposes, Haskell provides many advantages over other languages, especially, imperative languages such as Java or Python. The most immediate advantage is that in Haskell, functions are first-class objects, a property shared with other pure functional languages. For the current work, this is the most important feature which enables us to integrate computational relations into GKGs seamlessly. There are also other strong features of Haskell which are vital wherever correctness must be guaranteed [2]. Haskell is a safe and strongly typed language. This enables faithful retention of ontological relations through transformations. In Haskell, variables are immutable, which enables verification of the code through algebraic manipulations [1]. Most importantly, imperative (impure) operations are syntactically separated from the pure ones through the use of monads [12, 18].

Here, we present a simple example to illustrate the process of reasoning about spatial topological relations among three polygons. Then, a computational relation of a case study will be presented.

### 3.1 Reasoning about Spatial Topological Relation

Figure 1 depicts the spatial topological relations among three polygons. Polygon $A$ contains Polygon $B$, Polygon $B$ contains Polygon $C$. The DE-9IM matrices for the corresponding relations are also displayed on the right side of Figure 1. Based on the relation between Polygon $A$ and Polygon $B$, and the relation between Polygon $B$ and Polygon $C$, the relation between Polygon $A$ and Polygon $C$ can be inferred.

Although this is a simple deduction, the basic principles can be demonstrated via the corresponding Haskell implementation. Two functions are defined: one is the `compu_sr`() functions to compute the matrix of the spatial topological relation between two objects, and another one is the `reason_sr`(), which infers the matrix that represents the spatial topological relation based on two input matrices. For function `compu_sr`(), the two input parameters are two spatial objects, the output is the matrix that describes the topological relation between the two input objects. Keyword `data` is applied to define the data types of parameters, while `class` defines the types that share same behaviors, and can be computed by the function declared in the `class`.

```
data Matrix= Matrix [[Int]]

class COMPU_SPATIAL_RELATION object_1 object_2
  where compu_sr:: object_1-> object_2-> Matrix

class REASON_SPATIAL_RELATION matrix_1 matrix_2
  where reason_sr:: matrix_1-> matrix_2-> Matrix
```

Before implementing the defined functions, the data types of entities used as real instance data are defined. The matrices of relations between two pairs of entities ((Polygon_A, Polygon_B) and (Polygon_B, Polygon_C)) are generated.

```
data Entity_A = Polygon_A
data Entity_B = Polygon_B
data Entity_C = Polygon_C

instance COMPU_SPATIAL_RELATION Entity_A Entity_B where
compu_sr (Polygon_A)(Polygon_B) = Matrix [[2,1,2],[-1,-1,1],[-1,-1,2]]

instance COMPU_SPATIAL_RELATION Entity_B Entity_C where
compu_sr (Polygon_B)(Polygon_C) = Matrix [[2,1,2],[-1,-1,1],[-1,-1,2]]
```

Using the matrices computed above, the data types of input matrices are defined, and the matrix of the "contains" relation between Polygon $A$ and Polygon $C$ is generated.

```
data Matrix_M= Matrix_M [[Int]]

instance REASON_SPATIAL_RELATION Matrix_M Matrix_M
  where reason_sr (Matrix_M [[2,1,2],[-1,-1,1],[-1,-1,2]])
                  (Matrix_M [[2,1,2],[-1,-1,1],[-1,-1,2]])
                = Matrix [[2,1,2],[-1,-1,1],[-1,-1,2]]
```

## 3.2 Computational Relations

Soil erosion is a widespread and major environmental threat to terrestrial ecosystems. To study soil erosion in the Chinese Loess Plateau, we use the revised universal soil loss equation (RUSLE) [10]. RUSLE is a widely accepted method that can be used as the best-fitted model for monitoring soil erosion. The RUSLE equation is as follows:

$$A = R \times K \times LS \times C \times P,$$

in which $A$ is the average soil loss ($t \cdot hm^{-1} \cdot a^{-1}$). Here we take the calculation of factor $R$ as an example of a computational relation in models of GKGs, where $R$ is the rainfall erosivity factor ($MJ \cdot mm \cdot hm^{-1} \cdot h^{-1} \cdot a^{-1}$). The rainfall erosivity factor of a particular month can be obtained using:

$$R_i = 1.735 \times 10^{(1.5 \times lg \frac{p_i^2}{p_a} - 0.8188)} \tag{1}$$

where $R_i$ is the rainfall erosivity factor of a particular month, $p_i$ is the monthly rainfall in a particular year, and $p_a$ is the yearly rainfall. The first step is to construct formal ontology of rainfall. We integrate top-level ontologies in Semantic Web for Earth and Environmental Terminology (SWEET) Ontology [13] with type classes, which are sets of types sharing the same behavior.

```
class THING thing
```

The upper-case `THING` is the type class name, which is the top ontology in SWEET. The parameter `thing` is the type that belongs to the class. To represent subclass relations, the symbol => is used to illustrate the hierarchy of ontology levels. `PHENOMENA` is the subclass of `THING` class in SWEET, we then define the class `RAINFALL`, `AVERAGE_ANNUAL_RAINFALL` and `AVERAGE_MONTHLY_RAINFALL`.

```
class THING phenomena => PHENOMENA phenomena
class PHENOMENA rainfall => RAINFALL rainfall

class RAINFALL average_annual_rainfall =>
  AVERAGE_ANNUAL_RAINFALL average_annual_rainfall

class RAINFALL average_monthly_rainfall =>
  AVERAGE_MONTHLY_RAINFALL average_monthly_rainfall
```

To represent the calculation of factor $R$, the keyword `data` is applied to define the data type of parameters in (1). The name of the data type is `Value`. There are four value constructors in data type `Value`, for example, constructor `Measure` has the data type `Float`.

```
data Value=Measure Float | Count Int | Boolean Bool | Category String
```

To represent the computational relation described by (1), we first define the type class `RAINFALL_EROSIVITY_FACTOR` (REF), which corresponds to the dependent variable $R_i$ in (1). Two parameters must be included in type class REF: `average_monthly_rainfall` (representing monthly average rainfall $p_i$), and `average_annual_rainfall` (denoting yearly average rainfall $p_a$). REF is declared as a multi-parameter type class for the computational relation, defining which parameters are required to calculate the rainfall erosivity factor. The input parameters of the function `rFactor` are average monthly rainfall and average annual rainfall, the data type of the result is one of the data type constructors in `Value`.

```
class RAINFALL_EROSIVITY_FACTOR average_monthly_rainfall
average_annual_rainfall
  where rFactor ::  average_monthly_rainfall->
                    average_annual_rainfall -> Value
```

Next, we use the keyword `instance` to implement the computational relation defined above. Firstly, we define the data type of average monthly rainfall and average annual rainfall as float type. `Avg_monthly_rainfall` and `Avg_annual_rainfall` are the real instance data types that correspond to `average_monthly_rainfall` and `average_annual_rainfall`. The parameters `monthlyrain_avg` and `annualrain_avg` are the parameters with data type `Float`, which are the input parameters for function `rFactor`. Therefore, the constructor `Measure` in `Value` is used to illustrate the data type of rainfall erosivity factor. The computational relation exists among rainfall erosivity factor, average monthly rainfall and average annual rainfall.

```
data Avg_monthly_rainfall = Avg_monthly_rainfall Float
data Avg_annual_rainfall = Avg_annual_rainfall Float

instance RAINFALL_EROSIVITY_FACTOR Avg_monthly_rainfall
Avg_annual_rainfall
  where rFactor (Avg_monthly_rainfall monthlyrain_avg)
                (Avg_annual_rainfall annualrain_avg) =
                Measure (1.735*(10**(1.5* (logBase 10
                ((monthlyrain_avg))**2/(annualrain_avg)))-0.8188))
```

## 4 Conclusion

Functional languages provide many advantages over their imperative counterparts for representation and reasoning tasks in geoscience knowledge graphs (GKGs). The first step in this direction is to represent computational relations as first-class objects, which enables further reasoning and algorithmic processing in a uniform framework. As such, languages such as Haskell which treat functions as first-class primitives are ideal for augmenting GKGs with computational relations as first-class entities. Higher-order functions can then be used for uniform processing of ground entities and computational relations alike. Furthermore, Haskell is ideal for computational processes that must retain ontological relations faithfully. We have demonstrated through some simple examples how these tasks can be handled in Haskell.

In our future research, further computational relations will be studied. Besides mathematical expressions, logic rules and modelling procedures will also be considered. Moreover, the representation of spatial direction relations and temporal relations will be included. Based on the case study of representing computational relations in this paper, we will investigate efficient reasoning with spatial, temporal, and computational relations in GKGs. This includes seamless integration of functional programming into qualitative spatial and temporal reasoning.

## References

1 John Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641, 1978. doi:10.1145/359576.359579.

2 Thomas Bittner, Jonathan P. Bona, and Werner Ceusters. Ontologies of dynamical systems and verifiable ontology-based computation: Towards a Haskell-based implementation of Referent Tracking. In Roberta Ferrario and Werner Kuhn, editors, *Formal Ontology in Information Systems – Proceedings of the 9th International Conference, FOIS 2016, Annecy, France, July 6-9, 2016*, volume 283 of *Frontiers in Artificial Intelligence and Applications*, pages 313–327. IOS Press, 2016. doi:10.3233/978-1-61499-660-6-313.

3 Xiaojun Chen, Shengbin Jia, and Yang Xiang. A review: Knowledge reasoning over knowledge graph. *Expert Syst. Appl.*, 141, 2020. doi:10.1016/j.eswa.2019.112948.

4 Jiaxin Du, Shaohua Wang, Xinyue Ye, Diana S Sinton, and Karen Kemp. GIS-KG: Building a large-scale hierarchical knowledge graph for geographic information science. *International Journal of Geographical Information Science*, pages 1–25, 2021.

5 Antony Galton. Spatial and temporal knowledge representation. *Earth Sci. Informatics*, 2(3):169–187, 2009. doi:10.1007/s12145-009-0027-6.

6 Leo S Hsu and Regina Obe. *PostGIS in action.* Simon and Schuster, 2021.

**7** Zhenjiang Hu, John Hughes, and Meng Wang. How functional programming mattered. *National Science Review*, 2(3):349–370, 2015.

**8** Graham Hutton. *Programming in Haskell*. Cambridge University Press, 2nd edition, 2016.

**9** Martin Leinberger, Ralf Lämmel, and Steffen Staab. The essence of functional programming on semantic data. In *Programming Languages and Systems – 26th European Symposium on Programming, ESOP 2017*, volume 10201 of *Lecture Notes in Computer Science*, pages 750–776. Springer, 2017.

**10** Xun-Gui Li and Xia Wei. Soil erosion analysis of human influence on the controlled basin system of check dams in small watersheds of the Loess Plateau, China. *Expert Syst. Appl.*, 38(4):4228–4233, 2011. `doi:10.1016/j.eswa.2010.09.088`.

**11** Gengchen Mai, Krzysztof Janowicz, Yingjie Hu, Song Gao, Bo Yan, Rui Zhu, Ling Cai, and Ni Lao. A review of location encoding for GeoAI: methods and applications. *International Journal of Geographical Information Science*, pages 1–35, 2022.

**12** Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. `doi:10.1016/0890-5401(91)90052-4`.

**13** Robert G Raskin and Michael J Pan. Knowledge representation in the Semantic Web for Earth and Environmental Terminology (SWEET). *Computers & Geosciences*, 31(9):1119–1125, 2005.

**14** Ahmet Sayar, Marlon Pierce, and Geoffrey Fox. OGC compatible geographical information systems web services. *Indiana Computer Science Report TR610*, 2005.

**15** Sven Schade, Frank O. Ostermann, and Laura Spinsanti. Functional integration for the observation web. In *KEOD 2011 – Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Paris, France, 26-29 October, 2011*, pages 498–504. SciTePress, 2011.

**16** Syed K Shahzad, Michael Granitzer, and Klause Tochterman. Designing user interfaces through ontological user model: functional programming approach. In *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pages 99–104, 2009.

**17** Christian Strobl. Dimensionally Extended Nine-Intersection Model (DE-9IM). In *Encyclopedia of GIS*, pages 470–476. Springer, 2017. `doi:10.1007/978-3-319-17885-1_298`.

**18** Philip Wadler. How to declare an imperative. *ACM Comput. Surv.*, 29(3):240–263, 1997. `doi:10.1145/262009.262011`.

**19** Shu Wang, Xueying Zhang, Peng Ye, Mi Du, Yanxu Lu, and Haonan Xue. Geographic Knowledge Graph (GeoKG): A formalized geographic knowledge representation. *ISPRS Int. J. Geo Inf.*, 8(4):184, 2019. `doi:10.3390/ijgi8040184`.

**20** Yi Zhang, Yong Gao, LuLu Xue, Si Shen, and KaiChen Chen. A common sense geographic knowledge base for GIR. *Science in China Series E: Technological Sciences*, 51(1):26–37, 2008.

**21** Chenghu Zhou, Hua Wang, Chengshan Wang, Zengqian Hou, Zhiming Zheng, Shuzhong Shen, Qiuming Cheng, Zhiqiang Feng, Xinbing Wang, Hairong Lv, et al. Geoscience knowledge graph in the big data era. *Science China Earth Sciences*, 64(7):1105–1114, 2021.