λProlog($\mathcal{QS}$): Functional Spatial Reasoning in Higher Order Logic Programming

Beidi Li
Department of Engineering, Aarhus University, DK
beidi.li@eng.au.dk

Mehul Bhatt
School of Science and Technology, Örebro University, Sweden
http://www.mehulbhatt.org/
Mehul.Bhatt@oru.se

Carl Schultz
DIGIT, Aarhus University, DK
http://think-spatial.org
cschultz@eng.au.dk

Abstract

We present a framework and proof-of-concept implementation for functional spatial reasoning within *high-order logic programming*. The developed approach extends λProlog to support reasoning over spatial variables via Constraint Handling Rules. We implement our approach within Embeddable λProlog Interpreter (ELPI) and demonstrate key features from combined reasoning over spatial functions and relations. The reported research is an ongoing development of the *declarative spatial reasoning* paradigm.

2012 ACM Subject Classification Computing methodologies → Spatial and physical reasoning

Keywords and phrases Spatial reasoning, Functional logic programming, Lambda-Prolog

Digital Object Identifier 10.4230/LIPIcs.COSIT.2019.26

Category Short Paper

Funding This research was funded in part by the Danish Independent Research Fund within the project Intelligent Software Healing Environments.

1 Introduction

Declarative spatial reasoning denotes the ability to (declaratively) specify and solve real-world problems related to mixed geometric (i.e., quantitative) and qualitative visual and spatial representation and reasoning [3]; the paradigm emphasises diverse forms of reasoning capabilities (e.g., question-answering, learning, abduction) with a rich spatio-temporal ontology where aspects pertaining to space, time, events, actions, change, interaction, conceptual knowledge may be handled as first-class objects within a systematic formal artificial intelligence / *knowledge representation and reasoning* (KR) framework [2]. From the practical viewpoint of practical KR methods, this encompasses spatial reasoning with *answer set programming* [12, 14, 15], *constraint logic programming* [3, 10], and *inductive logic programming* [11]. This paper continues this line of work by developing a KR framework for reasoning in a seamless, integrated way over spatial *functions*, spatial relations, and KR-based domain-specific conceptual knowledge.

In many application areas where *space* plays a central role, such as architectural design or Constructive Solid Geometry, it is necessary to not only represent and reason about *relations* between spatial entities, but to also express and evaluate *functions* over spatial entities. For example, we may want to query the incidence relation between a point $(5, 5)$ and the

14th International Conference on Spatial Information Theory (COSIT 2019).
Editors: Sabine Timpf, Christoph Schlieder, Markus Kattenbeck, Bernd Ludwig, and Kathleen Stewart; Article No. 26; pp. 26:1–26:8
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

intersection of two polygons A, B. In the context of architectural design, polygons A and B may be used to represent the *visibility space* from which a sign and a landmark L_A, L_B are visible, and the point may represent an important *threshold* position where a person is expected to need to orient themselves as they enter a large open room:

```
?- A = polygon [(vertex 0 0), ...],
   B = polygon [(vertex 10 0), ...],
   incidence Relation (point 5 5) (intersect A B).
```

The query result is:

```
Relation = exterior.
```

That is, the point is *exterior* to the intersection region meaning that, in the context of architectural design, the sign and the landmark are not mutually visible from the threshold position, suggesting that an occupant may lose orientation at that critical location. For a second example in the context of Constructive Solid Geometry, suppose we have cube *Cube* that has side length 7 and whose centroid is located at point $(5, 5, 5)$, and sphere *Sphere* with radius 4 and centroid $(10, \_, 5)$, such that the Y coordinate of the centroid is unknown (i.e. the Y coordinate is an unbound real valued variable). These spatial entities may be defined by transforming (translating, scaling) primitive unit-sized entities e.g. a unit cube with side length 1 centred at point $(0, 0, 0)$, and a unit sphere with radius 1 centred at $(0, 0, 0)$. We then assert that *Cube* is topologically *part of* the *Sphere*:

```
?- Cube   = translate (vector  5 5 5) (scale 7 unit_cube),
   Sphere = translate (vector 10 _ 5) (scale 4 unit_sphere),
   topology part_of Cube Sphere.
```

The query result is:

```
false.
```

This means that no translation can satisfy the required topological relation, due to the cube being too large to be part of the sphere.

In this paper we show that reasoning over the combination of spatial functions and spatial relations overcomes numerical instability problems in certain well-defined cases (that would otherwise result in logical inconsistencies), and provides significantly more computationally efficient query answering. Returning to the architecture example above, suppose that the visibility spaces A, B are *disconnected*: the intersection of A and B will be the empty (void) region, to which every point is necessarily *exterior*. Therefore, the result that $Relation = exterior$ is arrived at based on purely qualitative spatial reasoning, thus avoiding the need for potentially expensive and unstable numerical computations of polygon intersections, and point-region incidence checks.

In this paper we develop the foundations for reasoning about spatial functions in logic programming, λProlog($\mathcal{QS}$), based on the λProlog framework extended with constraint programming. Based on previous work we target a specific class of *qualitative spatial constraints* that we formulate in the framework of polynomial constraint solving [3, 14]. Our key contributions, and the novel features provided by our integration of spatial reasoning in λProlog, are:

- integrated reasoning about both spatial functions and spatial relations (Section 4);
- by representing spatial functions as abstract syntax trees we can avoid logical inconsistencies that arise from numerical instabilities when computing intermediate functions (Section 5).
- a proof-of-concept implementation of λProlog($\mathcal{QS}$)with query examples is available at: `http://think-spatial.org/Resources/LamPrologQS.zip`

2 Preliminaries: Lambda Prolog

Our λProlog($\mathcal{QS}$) system builds on lambda logic programming theory originally developed by Nadathur and Miller [9], and extended with constraint programming [7].

Prolog. [13] We assume basic familiarity with first-order logic. A *term* is either a variable, constant, or a compound term (or *predicate*) $f(t_1, \ldots, t_n)$ with functor f applied to terms $t_1, \ldots, t_n$. A Prolog program L_P consists of a finite set of universally quantified *rules* of the form $h \leftarrow b_1, \ldots, b_n$ such that h is a predicate, and the expression $b_1, \ldots, b_n$ is a conjunction of predicates (i.e. rules are Horn clauses). Prolog *facts* are rules of the form $h \leftarrow \top$. A *query* is a conjunction of predicates $b_1, \ldots, b_n$. A ground term is a term with no variables. The Herbrand universe U of L_P is the set of ground terms that can be made from the constants and function symbols of L_P. Let q be a query, then $q\theta$ is a conjunction of ground predicates resulting from an assignment of all variables in q to values from U. A *query* is a logical consequence of L_P if $\exists\theta(L_P \models q\theta)$.

λProlog. λProlog [9] is an extension of Prolog that supports λ-terms as data structures, and higher-order programming beyond what can be expressed using Horn clauses.[1] λ-terms include variables (e.g. x, y, z), constants (e.g. alphanumerical strings), function application $(s\ t)$ and abstraction $(\lambda x.s)$, where s, t are λ-terms. λ-terms enable high-order unification by λ-conversion and facilitate the manipulation of variable names and substitution. λPrologalso incorporates a GENERIC search operation for unification so that type errors detected during parsing are used to 1identify goals that will never succeed.

ELPI [4] is an implementation of λProlog extended with a constraint system based on the Constraint Handling Rules (CHR) language [5]. We implement spatial relations as CHR constraints in ELPI. The constraint system extension consists of a *constraint store* and CHR *rules*. Whenever a λ-term is added to the store, all CHR rules are checked to see if a λ-term match occurs, causing the rule to *fire*. Rules have the form:

rule $t_{match} \setminus t_{remove} \mid t_{guard} \Leftrightarrow t_{add}$

where $t_{match}, t_{remove}, t_{add}$ are λ-terms and t_{guard} is a condition that is either true or false. A rule is fired if t_{match} and t_{remove} are in the store, and t_{guard} is true. This causes term t_{add} to be added to the store, and t_{remove} to be removed from the store.

3 Spatial Representation and Reasoning

The qualitative spatial domain ($\mathcal{QS}$) that we focus on in our formal framework consists of the following ontology.

Spatial Domains. Domain entities in $\mathcal{QS}$ are as follows. A 2D *point* is a pair of reals x, y. A *3D point* is a triple of reals x, y, z. A *simple polygon* is a 2D spatial region (single piece, no holes) defined by a list of n vertices (points) $p_1, \ldots, p_n$ (spatially ordered counter-clockwise) such that the boundary is non-self-intersecting, i.e., there does not exist a polygon boundary

[1] In summary, Horn clauses in Prolog are replaced by Hereditary Harrop formulas in λProlog. The role of *resolution refutation* as the logical foundation for sound querying in Prolog is replaced by *sequent calculus* in λProlog.

edge between vertices p_i, p_{i+1} that intersects some other edge p_j, p_{j+1} for all $1 \leq i < j < n$ and $i + 1 < j$. A *simple polyhedron* is a 3D spatial region (single piece, no holes) defined by a set of 3D vertices (points) $V = p_1, \ldots, p_n$ and a set of faces $f_1, \ldots, f_m$ where each face is a triple of vertices $v_1, v_2, v_3 \in V$. A (general) *polygon* is a set of *boundaries* and a set of *holes* (each set of which are simple polygons) such that every hole is a non-tangential part of one boundary. A (general) *poyhedron* is a set of *boundaries* and a set of *holes* (each set of which are simple polyhedra) such that every hole is a non-tangential part of one boundary.

A spatial *object* $o \in O$ is a variable associated with a spatial domain D (e.g. the domain of $2D$ points). An *instance* of an object $i \in D$ is an element from the domain. Given $O = \{o_1, \ldots, o_n\}$, and domains $D_1, \ldots, D_n$ such that o_i is associated with domain D_i, then a *configuration* of objects ψ is a one-to-one mapping between object variables and instances from the domain, $\psi(o_i) \in D_i$.

For example, a variable o_1 is associated with the domain D_1 of 2D points. The point $(0, 1)$ is an instance of D_1. A configuration is defined that maps o_1 to $(0, 1)$ i.e. $\psi(o_1) = (0, 1)$.

Spatial Relations and Spatial Functions. Let $D_1, \ldots, D_n$ be spatial domains. A spatial relation r of arity n $(0 < n)$ is defined as:

$$r \subseteq D_1 \times \cdots \times D_n$$

Given a set of objects O, a relation r of arity n can be asserted as a constraint that must hold between objects $o_1, \ldots, o_n \in O$, denoted $r(o_1, \ldots, o_n)$. The constraint $r(o_1, \ldots, o_n)$ is satisfied by configuration ψ if $\big(\psi(o_1), \ldots, \psi(o_n)\big) \in r$. For example, if *dc* is a topological relation *disconnected*, and O is a set of polygon objects, then $dc(o_4, o_9)$ is the constraint that polygons $o_4, o_9 \in O$ are disconnected. We define topological, size, and incidence spatial relations, as presented in Table 1.[2]

A spatial function f of arity $n - 1$ $(1 < n)$ is defined as:

$$f : D_1 \times \cdots \times D_{n-1} \rightarrow D_n$$

That is, each function maps $(n - 1)$ spatial entities to a (single) spatial entity. For example, if *translate* is a spatial transformation function, v is a vector $(5, 5)$ and T is a polygon with vertices $((0, 0), (10, 0), (5, 5))$ then $(\textit{translate}\ v\ T)$ evaluates to the polygon with vertices $((5, 5), (15, 5), (10, 10))$. We introduce the unique *void* spatial entity to ensure that spatial functions are closed over the spatial domains. For example, the intersection of two disconnected polygons is not itself a polygon, but rather the *void* spatial entity. Spatial functions defined in λProlog($\mathcal{QS}$) are presented in Table 1.

4 Spatial Functions in λProlog:

Using the λProlog type system we define fundamental spatial types *point*, *region*, and define vertices, simple polygons, and (general) polygons as functions:

```
%% defining instances of topological relationships (extract only)
type contact, disconnect, partial_overlap, part_of relation_topology.
%% defining specialised spatial domains        %% polymorphic typing through functions (extract only)
kind point  type.                              type vertex        real -> real -> point.
kind region type.                              type polygon       list point -> list point -> region.
kind relation type.                            type spatial_void  region.
```

[2] *Discrete from* means that two regions do not share any interior point, *overlaps* means they share at least one interior point, and *disconnected* means they do not share any point including on the boundary.

Table 1 λProlog($\mathcal{QS}$) relation predicates and functions.

$\mathcal{QS}$ **Relations**	**Description**
size: Relation × Region × Region	Size relations between regions: smaller, equisize, larger.
topology: Relation × Region × Region	Contact relations between regions: contact, disconnected, discrete_from, overlaps. partial_overlap, part_of, proper_part_of.
incidence: Relation × Point × Region	Incidence relations between points and regions: interior, on_boundary, exterior.

$\mathcal{QS}$ **Functions**	**Description**
centroid: Region → Point	Centre point of region. Centroids of polygons and polyhedra are the average of their vertices.
extent: Region → $\mathbb{R}$	Area for 2D regions, and volume for 3D regions.
translate: Point × Region → Region	Translates region by a vector defined by the given point.
scale: $\mathbb{R}$ × Region → Region	Scales region by the given positive factor about the region's centroid point.
union: Region × Region → Region	Union of two regions.
intersect: Region × Region → Region	Intersection of two regions.
difference: Region × Region → Region	Difference of two regions.

We define spatial functions and spatial relations to range over these types (Table 1):

```
%% signatures of spatial relations
type incidence relation_incidence -> point  -> region -> o.
type topology  relation_topology  -> region -> region -> o.
type size      relation_size      -> region -> region -> o.

%% signatures of spatial functions
type centroid   region -> point.
type extent     region -> real.
type translate  point  -> region -> region.
type scale      real   -> region -> region.
type union      region -> region -> region.
type intersect  region -> region -> region.
type difference region -> region -> region.
```

We implement algebraic semantics of spatial relations in CHR. For example, no region is *disconnected* from itself (irreflexive), and if region A is *part of* region B, and region B is *part of* region C, then A must necessarily be a *part of* C:

```
rule(topology disconnected A B) | (A=B) <=> false.  %% disconnected is irreflexive
rule((topology part_of A B), (topology part_of B C)) | true <=> part_of A C.  %% part of is transitive
```

Combined reasoning about spatial functions and relations. We use λ-terms to capture the higher-order abstract syntax of spatial functions, and *reduce* this structure by rewriting it in a simplified form based on the algebraic properties of those spatial functions. For example, the union of a polygon A with itself, expressed as the λ-term ($union\ A\ A$), is necessarily equivalent to A, and thus we can reduce ($union\ A\ A$) simply to A without any further geometric calculations. More generally, given two polygons A, B, then ($union\ A\ B$) can reduce to B when A is a *part of* B. Even more generally still, the arguments A, B need not be polygons but can be arbitrarily complex spatial λ-terms: if A is part of B then the term ($union\ A\ B$) can be reduced to B.

On the other hand, we can deduce that certain spatial relations must hold between the arguments of a function and the result of the function. For example, two non-void regions A, B must each necessarily be *part of* the union of A and B. Similarly, if regions A and B topologically *overlap* then the intersection of A and B must necessarily be part of A and part of B. By recursively stepping through a spatial λ-term, deducing the relations between its parts and simplifying, we can potentially reduce the λ-term at a purely symbolic level. Once no further reductions can be made, λProlog($\mathcal{QS}$) evaluates the true numerical spatial functions (union etc.) using computational geometry libraries GPC[3] for polygons and PyMesh[4] for polyhedra. In the following section we demonstrate the power of this approach. The following code excerpt implements the above example cases, and the recursive *simplify* predicate for reducing spatial λ-terms:

```
%% Simplifying abstract syntax trees of spatial functions:
%%    (1) If A is part of B, then (union A B) reduces to B
simplify_ (union A B) B :- topology part_of A B.
%%    (2) If A is disconnected from B, then (intersect A B) reduces to the spatial void type
simplify_ (intersect A B) spatial_void :- topology discrete_from A B.

%% CHR rules for deducing spatial relations between function arguments and function evaluations:
%%    (1) A and B are each part of (union A B)
rule (deduce (union A B)) | true <=>
  topology part_of A (union A B), topology part_of B (union A B).
%%    (2) if A and B contact, then (intersect A B) is part of A and part of B
rule (deduce (intersect A B)) | (topology overlaps A B) <=>
  topology part_of (intersect A B) A, topology part_of (intersect A B) B.

%% Recursive definition of the simplify predicate
simplify (point X Y) (point X Y).  %% base case
simplify (polygon B H) (polygon B H).  %% base case
simplify (Op Left Right) Simp :-   %% recursive step
  simplify Left SLeft, simplify Right SRight,
  (deduce (Op SLeft SRight)), simplify_ (Op SLeft SRight) Simp.
```

5 Empirical Evaluation

In this section we demonstrate key features of our current implementation of λProlog($\mathcal{QS}$).

Ex1: Architectural Design. This example demonstrates how λ-term reduction based on combined reasoning over spatial functions and relations avoids potentially expensive geometric computations. A building consists of objects represented as facts in the knowledge base, including a landmark *statue* that is positioned in a central courtyard that is visible from many rooms, and a number of signs. Each object has a visibility space, i.e. a polygon describing the points on the floor plan from which an object can be seen (also referred to as the *isovist*). The building has numerous threshold positions from which building occupants are expected to need some orientation if they are unfamiliar with the building, such as the entrance to a large room. This is modelled as facts in λProlog($\mathcal{QS}$):

```
%% domain objects
landmark (id lm8263) (object_type statue).  sign (id sign73).
threshold_position (point 5.3 82.3).
%% 2D geometric representations of visibility spaces
visibility_space (id lm8263) (polygon [(vertex 52.3 56.0) ...]).
visibility_space (id sign73) (polygon[(vertex 32.3 281.0) ...]).
```

The architect wants to identify threshold positions from which the occupant does *not* have visible access to *both* the central statue and at least one sign.

```
?- threshold_position Position, landmark Statue (object_type statue),
   visibility_space Statue StatueVisibility,
   not((
      (sign Sign), (visibility_space Sign SignVisibility),
      incidence interior Position (intersect StatueVisibility SignVisibility)
   )).
```

Given such visibility constraints, a numerical program will need to compute the intersection of every pair of statue and sign visibility polygons to determine whether the threshold position

[3] http://www.cs.man.ac.uk/~toby/alan/software/
[4] https://pymesh.readthedocs.io/en/latest/

lies in their intersection. By contrast, λProlog($\mathcal{QS}$) directly reduces the intersection to the *void* spatial entity at a purely symbolic level when the visibility polygons are disconnected, thus avoiding potentially computationally expensive geometric calculations.

Ex2: Avoiding logical inconsistencies from numerical instability. This example demonstrates how λProlog($\mathcal{QS}$) guarantees logical soundness for λ-term reduction in cases where relying on numerically evaluating intermediate terms fails. The powerful polygon set operation library GPC cannot be used to conclude the trivial equality (see Figure 1):

```
?- A = simple_polygon([(vertex 0.0 0.0), (vertex 3.0 0.0), (vertex 3.0 4.0)]),
   B = simple_polygon([(vertex 1.0 0.0), (vertex 4.0 1.1), (vertex 0.0 3.2)]),
   equal A (union (intersect A B) (difference A B)).
```

λProlog($\mathcal{QS}$) gives the query result *true*, which is correct. In contrast, when the intermediate results of (*intersect* A B) and (*difference* A B) are evaluated using GPC, and then combined with a GPC union, the result has two extra vertices that are not precisely on the boundary of A due to rounding errors: $((3.0, 0.0), (0.0, 0.0), (0.71, 0.94), (1.7, 2.3), (3.0, 4.0))$, thus leading to a logical inconsistency that $A \neq A$. The problem becomes more evident in the 3D case where PyMesh generates erroneous mesh artefacts from computing $((S_1 \setminus S_2) \cup (S_1 \cap S_2)) \cup S_2$ where S_1 and S_2 are two meshes that approximate spheres (Figure 1). The result *should* be equal to $(S_1 \cup S_2)$ but due to the artefacts this equality does not hold. Again, λProlog($\mathcal{QS}$) gives the correct result through reduction, and only evaluates the actual numerical (geometric) results using GPC and PyMesh when no further λ-term reductions can be made.

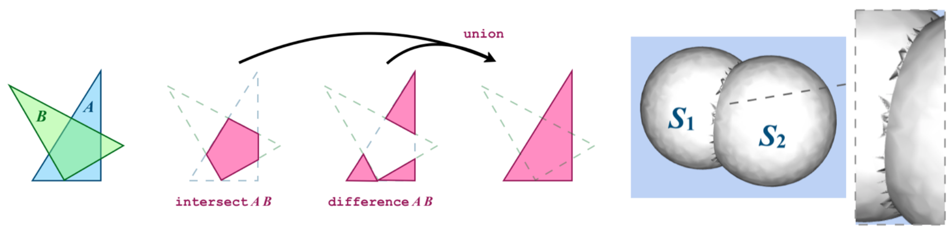

Figure 1 Cases where numerically evaluating intermediate functions using GPC and PyMesh results in logical inconsistencies. λProlog($\mathcal{QS}$) overcomes these limitations with λ-term reduction.

6 Conclusions

We have presented a framework and proof-of-concept implementation of λProlog($\mathcal{QS}$) that integrates functional spatial reasoning within logic programming. Our method facilitates efficient high-level reasoning about both spatial functions, domain-specific knowledge and spatial constraints in a seamless manner. In the broader AI research field, diverse frameworks have been developed that formalise notions of *space*, including: (a) geometric reasoning and constructive solid geometry [6]; (b) relational algebraic semantics of "qualitative spatial calculi" [8] (e.g., the SparQ spatial reasoning tool [16]); and (c) axiomatic frameworks of mereotopology and mereogeometry [1]. However, the distinction with our research here, and what we argue is lacking within the KR community, is a systematic formal account and computational characterisation of such spatial theories as a KR language – e.g., *suited for declarative modelling, commonsense inference and query.* In this paper we emphasise the power of such a research agenda, as our approach leverages from the strengths of both extensive research in functional logic programming and (declarative) spatial reasoning.

References

1 Marco Aiello, Ian E. Pratt-Hartmann, and Johan F.A.K. van Benthem. *Handbook of Spatial Logics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

2 Mehul Bhatt. Reasoning about Space, Actions and Change: A Paradigm for Applications of Spatial Reasoning. In *Qualitative Spatial Representation and Reasoning: Trends and Future Directions*. IGI Global, USA, 2012.

3 Mehul Bhatt, Jae Hee Lee, and Carl Schultz. CLP(QS): a declarative spatial reasoning framework. In *International Conference on Spatial Information Theory*, pages 210–230. Springer, 2011.

4 Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. ELPI: Fast, Embeddable, lambda-Prolog Interpreter. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 460–468. Springer, 2015.

5 Thom Frühwirth. Constraint handling rules. In *French School on Theoretical Computer Science*, pages 90–107. Springer, 1994.

6 Deepak Kapur and Joseph L. Mundy, editors. *Geometric Reasoning*. MIT Press, Cambridge, MA, USA, 1988.

7 Javier Leach, Susana Nieva, and Mario Rodríguez-Artalejo. Constraint logic programming with hereditary Harrop formulas. *Theory and Practice of Logic Programming*, 1(4):409–445, 2001.

8 Gérard Ligozat. *Qualitative Spatial and Temporal Reasoning*. Wiley-ISTE, 2011.

9 Gopalan Nadathur and Dale Miller. An overview of Lambda-PROLOG. *University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-40*, 1988.

10 Jakob Suchan and Mehul Bhatt. Semantic Question-Answering with Video and Eye-Tracking Data: AI Foundations for Human Visual Perception Driven Cognitive Film Studies. In *IJCAI*, pages 2633–2639, 2016.

11 Jakob Suchan, Mehul Bhatt, and Carl P. L. Schultz. Deeply Semantic Inductive Spatio-Temporal Learning. In James Cussens and Alessandra Russo, editors, *Proceedings of the 26th International Conference on Inductive Logic Programming (Short papers), London, UK, 2016.*, volume 1865 of *CEUR Workshop Proceedings*, pages 73–80. CEUR-WS.org, 2016. URL: `http://ceur-ws.org/Vol-1865/paper-12.pdf`.

12 Jakob Suchan, Mehul Bhatt, Przemysław Wałega, and Carl Schultz. Visual explanation by high-level abduction: On answer-set programming driven reasoning about moving objects. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

13 Allen Van Gelder, Kenneth A Ross, and John S Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649, 1991.

14 Przemysław Andrzej Wałęga, Mehul Bhatt, and Carl Schultz. ASPMT(QS): non-monotonic spatial reasoning with answer set programming modulo theories. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 488–501. Springer, 2015.

15 Przemysław Andrzej Wałęga, Carl Schultz, and Mehul Bhatt. Non-monotonic spatial reasoning with answer set programming modulo theories. *Theory and Practice of Logic Programming*, 17(2):205–225, 2017.

16 Diedrich Wolter and Jan Oliver Wallgrün. Qualitative spatial reasoning for applications: New challenges and the SparQ toolbox. In *Geographic Information Systems: Concepts, Methodologies, Tools, and Applications*, pages 1639–1664. IGI Global, 2013.