Ontology-Based Processing of Dynamic Maps in Automated Driving

Haonan Qiu[1,2], Adel Ayara[1] and Birte Glimm[2]
[1]*BMW Car IT GmbH, Ulm, Germany,*
[2]*Institute of Artificial Intelligence, University of Ulm, Germany,*
{haonan.qiu,adel.ayara}@bmw.de, birte.glimm@uni-ulm.de

Keywords: Autonomous driving, Digital maps, Ontologies, Rules, Knowledge Representation

Abstract: Autonomous cars act in a highly dynamic environment and consistently have to provide safety and comfort to the passengers. For a car to understand its surroundings, a detailed, high-definition digital map is needed, which acts as a powerful virtual "sensor". Compared to traditional digital maps, high-definition maps require significantly more storage space, which makes it largely impossible to store a complete map in a navigation system. Furthermore, map data is provided in numerous heterogeneous formats. Consequently, interoperability and scalability have become the main challenges of existing map processing solutions. We address these challenges by providing an interoperable knowledge-spatial architecture layer based on ontologies and confirm the scalability in an empirical evaluation.

1 INTRODUCTION

In 1990, a digital map was used for the first time in a built-in GPS-based navigation system offered by Mazda Eunos Cosmos cars (Leite, 2018). In the 2000s, great effort has been made to develop enhanced digital maps for supporting Advanced Driver Assistance Systems (ADAS). Enhanced digital maps are low-resolution maps containing road geometry and road attributes, such as road width and speed limit for ADAS at road level. We refer to enhanced digital maps as *standard definition (SD) maps*. Autonomous vehicles need, however, maps with a higher level of detail than what SD maps typically offer. In 2010, the concept of a "high-definition map" was introduced during a research workshop and such maps were then used in 2013 in the Bertha Drive Project (Herrtwich, 2018).

In contrast to SD maps, *high-definition (HD) maps* provide detailed information at *lane level* to support vehicle perception and localisation (Gruyer et al., 2016). Based on the functional system architecture of an automated driving system by Ulbrich et al. (2017), HD maps with localisation functionality provide inputs to the perception module for world modelling, and eventually support the decision making of the planning and control module. Almost all players in the area of highly automated driving, e.g., Google, HERE, TomTom, Baidu, BMW, or Toyota, rely on HD maps to further improve the driving capabilities of their autonomous vehicles.

Recently, collective efforts have been made towards standardising HD maps, such as Geographic Data File (GDF) 5.1 at international level (ISO/TC 204, 2018) or Navigation Data Standard (NDS) and the ADASIS protocol V3 at industrial level (NDS Association, 2018). There are also national efforts for developing HD map standards, such as the China Industry Innovation Alliance for the Intelligent and Connected Vehicles (CAICV). However, as of now, there is no single, authoritative format or standard for HD maps. As a result, map model development, maintenance, and integration, as well as map data exchange and sharing pose major challenges in practise.

HD maps are extremely detailed and, therefore, require significantly more processing power and computation resources compared to SD maps. This makes it largely impossible to store a complete detailed map within a vehicle's navigation system. Hence, the navigation system needs to constantly request map data streams while the car is progressing along a route and care has to be taken to provide any relevant information in time.

Summing up, the characteristics of HD maps require a novel approach that allows for a generic representation of the road environment and a dynamic update mechanism. Ontology-based road environment modeling has gained growing interest for this purpose in the Intelligent Transportation Systems community (Katsumi and Fox, 2018; Chen and Kloul, 2018; Bagschik et al., 2018), e.g., for road models that provide different features for specific application requirements (Zhao et al., 2017; Armand et al., 2017) and for spatial knowledge representation and reasoning for navigation and path planning (Belouaer et al., 2010; Hudelot et al., 2008).

In this paper, we address the challenge of dynamically updating the map data used within the navigation system of autonomous vehicles. The main contributions are as follows:

– We propose a knowledge-spatial architecture, based on OWL 2 RL ontologies (Hitzler et al., 2009), Datalog rules (Abiteboul et al., 1995), and SPARQL queries (W3C SPARQL Working Group, 2013), to deal with both knowledge abstraction and spatial reasoning on map data streams.

– We present a categorisation of the types of rules that are needed for knowledge abstraction and spatial reasoning.

– We propose the use of spatial window queries for continuous knowledge updates. Spatial windows contain an object, if the object satisfies a certain spatial query.

– We evaluate the approach for dynamic map updates in a high way scenario within our prototype called SmartMapApp.

The remainder of this article is structured as follows: First, we present some background knowledge about map data concepts and ontological modelling in Section 2. In Section 3, we introduce the knowledge-spatial architecture in detail. In Section 4, we evaluate the approach. We then briefly discuss the designing choices between SPARQL and Datalog in Section 5. Finally, we give conclusion in Section 6.

2 Preliminaries

In this section, we first briefly describe the map data model. Then we provide an introduction to ontologies, rules and SPARQL.

2.1 Map Data

Conceptually, HD maps mainly contain three layers: the road model, the lane model, and the localisation model, while SD maps only cover the road model. The road and the lane model layers are very important since they contain the semantics of the road and lane segments. The road model contains parts of the road that are not part of the lanes, such as edges of the road. It uses an ordered sequence of shape points that describe the geometry of a polyline in the direction of the road. Each road segment has a start and an end node, which belong to the intersection at the start and end of the road.

However, since autonomous vehicles are designed to center themselves in the lane, in reality, autonomous vehicles only need to deal with the lane model except on rare occasions when they need to travel outside the lane boundaries. The lane model includes information on lane geometries (boundaries, width, curvature, etc.), lane connectivity, lane type (vehicle lane, exit lane, etc.), travel direction, lane marking types (solid/dashed, single/double. etc.), and speed limits. The lane geometry determines the accuracy, storage efficiency and usability of the HD maps (Gwon et al., 2017). We refer the interested reader to the literature for further details about HD maps (Liu et al., 2020; Gran, 2019).

In practise, different companies provide maps in their own format and structure. Table 1 shows examples of the differences between two HD map formats, namely the HERE map content format and the NDS format. The first two columns show the differences in naming conventions, while the later two columns show the differences in definitions.

2.2 Ontologies

Studer et al. (1998) define an ontology as a "a formal, explicit specification of a shared conceptualization". More concretely, an ontology describes concepts and relations that are relevant to the modeling of a domain of interest. The three main building blocks of an ontology are: 1) *Classes* (aka concepts), which are representations of the concepts of the domain that the ontology aims to describe (e.g., Lane); 2) *Individuals*, which represent concrete entities in the application domain (e.g., lane1); 3) *Properties*, which represent attributes of and relationships between individuals (e.g., hasLength or hasDirectLeft). Properties are divided into *object* and *data properties*. Object properties are binary relationships between two individuals, while data properties relate individuals to literals/concrete data values (e.g., an integer value). Finally, an *ontology* is a set of *axioms*, i.e., *general statements* (e.g., stating that LeftMostLane is a subclass of Lane) and concrete *facts* (e.g., hasDirectLeft(lane1, lane2) or Lane(lane1)) about the application domain described using classes, properties and individuals.

The Web Ontology Language (OWL) is a formal language designed to model complex knowledge. OWL 2 RL is a fragment (aka profile) of OWL designed for handling large amounts of data/facts, where

Table 1: Examples of differences in HD map formats

	Point	Road Part	Numbering of Lanes	Lane Forming/Ending
HERE	node	segment	left to right	#lanes changes
NDS	shape point	link	curbside to middle	#lanes changes & physically separated

implicit facts can be derived via rule-based reasoning. Ontologies can be specified using the Resource Description Framework (RDF), where the axioms are written as subject-predicate-object triples using IRIs, blank nodes (representing existentially quantified variables) and literal values. For example, the fact Lane(lane1) is written as :lane1 rdf:type :Lane in RDF assuming suitable prefix declarations for the IRIs.

2.3 Rules

Rules can be used to axiomatise the semantics of a particular Web ontology language, e.g., OWL 2 RL, via a static set of rules or, using datalog rules, users can create custom rule sets. In order to define *datalog rules*, we fix countable, disjoint sets of *constants* and *variables*. A *term* is a constant or a variable. An *atom* has the form $P(t_1,\ldots,t_k)$, where P is a k-ary predicate and each t_i, $1 \leq i \leq k$, is a term. We focus on unary and binary atoms only (i.e., $1 \leq k \leq 2$), which correspond to classes and properties of the ontology, respectively. An atom is *ground* if it does not contain variables. A *fact* is a ground atom and a *dataset* is a finite set of facts, e.g., as defined in an ontology. A datalog *rule* is a logical implication of the form $H_1,\ldots,H_j \leftarrow B_1,\ldots,B_k$, where each H_i, $1 \leq i \leq j$, is a *head* atom, and each B_ℓ, $1 \leq \ell \leq k$, is a *body* atom. A datalog *program* is a finite set of rules. The consequents (head) of the rule are true, if all body atoms of a rule are true. The main computational problem in datalog systems is answering queries over the facts that logically follow from the explicitly stated facts and a datalog program.

A *negative* body atom has the form, `NOT EXISTS` $v_1, \ldots, v_j$ `IN` B, where each v_i, $1 \leq i \leq j$, is a variable and B is an atom. A rule r is *safe* if variables that appear in the head or in a negative body atom also appear in a positive body atom. A safe datalog rule can be extended with *stratified negation* by extending the rule to have negative body atoms, where there is no cyclic dependency between any predicate and a negated predicate.

An *aggregate* is a function that takes a multiset of values as input and returns a single value as output. An aggregate atom has the form `Aggregate`$(B_1,\ldots,B_k$ `ON` $x_1,\ldots,x_j$ `BIND` $f_1(e_1)$ `AS` $r_1 \ldots$ `BIND` $f_n(e_n)$ `AS` $r_n)$, where each B_i, $1 \leq i \leq k$, is an atom, each x_u, $1 \leq u \leq j$, is a variable that appears in B_i, each f_v, $1 \leq v \leq n$, is an aggregate function, each e_w, $1 \leq w \leq n$, is an expression containing variables from B_i, and each r_z, $1 \leq z \leq n$, is a constant for a variable that does not appear in B_i.

2.4 SPARQL

SPARQL is a query language for RDF graphs based on pattern matching. For example, the query

```
SELECT ?l WHERE
{ ?l rdf:type :LeftMostLane }
```

selects all individuals that are instances of the class `:LeftMostLane`. Note that we omit prefix declarations and we assume the queries to be evaluated over a pre-defined dataset in an RDF datastore. SPARQL 1.1 is the latest version supporting subqueries, value assignment, path expressions, aggregates, negation via `FILTER NOT EXISTS` and derivation of RDF nodes via `BIND` (W3C SPARQL Working Group, 2013). It also provides operations that update, create and remove data.

2.5 SKOLEM

Skolem functions allow dynamically generate “fresh" IRIs for blank nodes, called *Skolem IRIs*. Systems use Skolem IRIs “should mint a new, globally unique IRI (a Skolem IRI) for each blank node so replaced."(Cyganiak et al., 2014) The Skolem function takes the form as `SKOLEM`$("s", v_1,\ldots,v_k)$, where s is a arbitrary string and each v_i, $1 \leq i \leq k$, is a variable appearing in the rule body. Skolem IRIs are implemented using hasing schemes of various lengths (Hogan, 2015). *BIND* function assgins the Skolem IRI to an new variable. It takes the form as `BIND(SKOLEM`$("s", v_1,\ldots,v_k)$ `AS` $n)$, where n is the new variable holds the Skolem IRI.

3 A Knowledge-Spatial Architecture

In this section, we present the knowledge-spatial architecture that enables autonomous vehicles to perceive their environment with a dynamic map. We then explain the two-level ontologies and related rules patterns for knowledge abstraction and discuss how spatial reasoning interacts with knowledge abstraction.

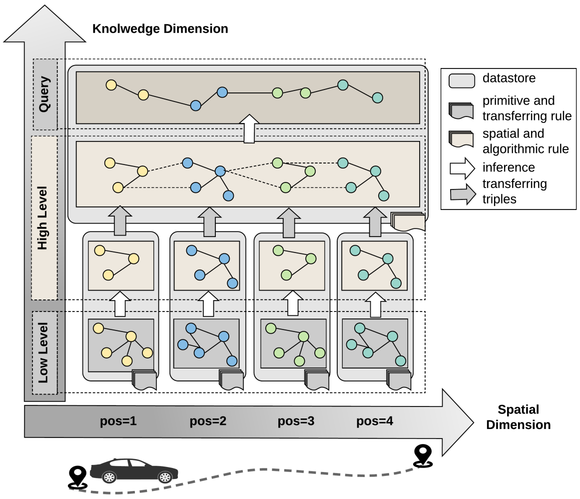


Figure 1: The overview of knowledge-spatial architecture

3.1 Architecture Overview

Figure 1 depicts the overall knowledge architecture. Inspired by the `Global Ontology Approach` (Wache et al., 2001), we perform a knowledge abstraction process across the *knowledge dimension* (vertical axis) from the format-specific and detailed low-level ontologies to the generic high-level ontology. The horizontal (time) axis represents road knowledge acquisition events, which trigger the knowledge abstraction process via spatial reasoning. Spatial reasoning considers updated vehicle motion events determined in the knowledge abstraction process and searches for spatial patterns to derive the relevant consequences of what is happening on the road. Different low-level ontologies for the different map formats can be used to feed the high-level ontology, which makes the proposed architecture very flexible as application-oriented queries, such as ADAS functions, are posed over the generic high-level ontology (Suryawanshi et al., 2019; Qiu et al., 2020).

The *spatial dimension* is orthogonal to the knowledge dimension and correlates facts that are true within a certain space. It describes the continuous spatial reasoning process with respect to the updated vehicle position and dynamic road environmental knowledge. We adopt the notion of a *spatial window* with a fixed width or region in terms of a geographic element shift (slide) over a path line. Inspired by Mokbel et al. (2005), we use the notion of *spatial expiration* depending on the spatial location of a moving object, e.g., a vehicle, and stored data expires only when the object leaves the spatial window.

Location-aware environments are, by nature, highly dynamic and the size of spatial streams is potentially infinite. To deal with this challenge, certain spacial events (e.g., the available map foresight of the vehicle reaches a threshold) trigger the initialisation of a new datastore for a low-level ontology. The next map data region is loaded (as defined by the specific map data format) and used to generate more abstract, high-level knowledge. While the high-level datastore uses spatial expiration for deletions, a low-level datastore is discarded once the high-level datastore is populated.

3.2 Rule Classification

Rules play a key role for knowledge abstraction and spatial reasoning. The main challenges for rule modelling are to (1) enrich instances at the primitive level; (2) transfer low-level ontologies with different concept definitions to a unified high-level ontology; (3) derive spatial relations for spatial reasoning; (4) capture decision-making algorithms in rules.

Based on the challenges mentioned above, we classify rules into (1) *primitive*, (2) *transfer*, (3) *spacial* and (4) *algorithmic rules* and we describe the four categories with sub-categories, examples and more formal rule patterns below. In the formal patterns, we use (possibly with subscripts) C for classes, op for object properties, and dp for data properties of the ontology over which the rules are to be executed.

3.2.1 Primitive Rules

These rules enrich instances with one-step inferences and their results serve as input for all other rules. Primitive rules use the identifiers to infer the relationships or attributes and are divided into *primitive relationship rules* and *primitive attribute rules*.

(a) Primitive relationship rules infer relationships between individuals. For a concrete example consider:

$$\begin{aligned}\mathsf{hasLane}(x,y) \leftarrow\ & \mathsf{RoadPart}(x),\ \mathsf{hasIdx}(x,i),\\ & \mathsf{Lane}(y),\ \mathsf{hasIdx}(y,i).\end{aligned}$$

More generally, such rules have the form

$$\mathsf{op}(x,y) \leftarrow \mathsf{C}_1(x),\ \mathsf{dp}(x,z),\ \mathsf{C}_2(y),\ \mathsf{dp}(y,z)$$

(b) Primitive attribute rules infer an attribute of an individual. For a concrete example consider:

$$\begin{aligned}\mathsf{speed}(x,v) \leftarrow\ & \mathsf{Node}(x),\ \mathsf{hasIdx}(x,i),\\ & \mathsf{speedVal}(y,v),\ \mathsf{hasIdx}(y,i).\end{aligned}$$

More generally, such rules have the form

$$\mathsf{dp}(x,v) \leftarrow \mathsf{C}(x),\ \mathsf{dp}_1(x,z),\ \mathsf{dp}_2(y,v),\ \mathsf{dp}_1(y,z).$$

3.2.2 Transfer Rules

These rules create a new individual in one ontology (typically the high-level ontology) based on one or more individuals in another ontology (typically a low-level ontology). SKOLEM functions and string manipulations are used to dynamically generate named (not anonymous) individuals (Shaw et al., 2011). The rules are further divided into *simple* (1:1) and *complex* (n:1) transfer rules according to instance correspondence.

(a) Simple rules create, for each individual of a certain type in one ontology, a new individual of a certain type in another ontology. For a concrete example, consider:

$$\begin{aligned}&\mathsf{RoadPart}(r), \mathsf{hasPoint}(r,s), \mathsf{length}(r,l) \leftarrow\\ &\quad \mathsf{Link}(k), \mathsf{hasShapePoint}(k,s),\\ &\quad \mathsf{length}(k,l), \mathtt{BIND}(\mathtt{SKOLEM}(\texttt{"}rp\texttt{"},k)\ \mathtt{AS}\ r).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}&\mathsf{C}_2(n),\ \mathsf{op}_2(n,u),\ \mathsf{dp}_2(n,v) \leftarrow\\ &\quad \mathsf{C}_1(x),\ \mathsf{op}_1(x,u), \mathsf{dp}_1(x,v),\\ &\quad \mathtt{BIND}(\mathtt{SKOLEM}(\texttt{"}c\texttt{"},\ x)\ \mathtt{AS}\ n),\end{aligned}$$

where C_1, op_1, and dp_1 are from one ontology and C_2, op_2, and dp_2 are from another ontology.

(b) Complex rules create a new individual in one ontology based on several individuals in another ontology and, hence, perform an aggregation process. As a concrete example, consider the rule

$$\begin{aligned}\mathsf{Lane}(l) \leftarrow\quad &\mathsf{Lane}(l_1),\ \mathsf{Lane}(l_2),\\ &\mathsf{laneNum}(l_1,n),\ \mathsf{laneNum}(l_2,n),\\ &\mathsf{hasConn}(l_1,c),\ \mathsf{hasConn}(l_2,c),\\ &\mathtt{BIND}(\mathtt{SKOLEM}(\texttt{"}l\texttt{"},\ c)\ \mathtt{AS}\ l).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}\mathsf{C}_2(w) \leftarrow\quad &\mathsf{C}_1(x),\ \mathsf{C}_1(y),\\ &\mathsf{dp}_1(x,v),\ \mathsf{dp}_1(y,v),\\ &\mathsf{op}_1(y,z),\ \mathsf{op}_1(y,z),\\ &\mathtt{BIND}(\mathtt{SKOLEM}(\texttt{"}c\texttt{"},\ z)\ \mathtt{AS}\ w).\end{aligned}$$

3.2.3 Spatial Rules

These rules infer important road environmental knowledge for vehicle navigation (Gayathri and V., 2018). We further subdivide them into *bounding rules*, *topological rules* and *distance rules*.

(a) Bounding rules infer the boundaries of an area or the range of a line, such as a start/end point or the left or right-most lane. Aggregation functions (e.g., MIN or MAX) can be used to identify an individual with a minimal or maximal bounding value. As a concrete example, consider:

$$\begin{aligned}\mathsf{LeftMostLane}(z) \leftarrow\ &\mathsf{Lane}(p),\\ &\mathtt{AGGREGATE}(\mathsf{hasIdx}(p,\ idx)\\ &\mathtt{ON}\ p\ \mathtt{BIND}\ \mathtt{MAX}(idx)\ AS\ m),\\ &\mathsf{Lane}(z),\ \mathsf{hasIdx}(z,\ m).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}&\mathsf{C}_2(z) \leftarrow \mathsf{C}_1(x),\ \mathtt{AGGREGATE}(\mathsf{dp}_1(x,v)\ \mathtt{ON}\ x\\ &\quad \mathtt{BIND}\ \mathtt{MAX}(v)\ \mathtt{AS}\ m),\ \mathsf{C}_1(z),\ \mathsf{dp}_1(z,m).\end{aligned}$$

Such rules might also use (stratified) negation to identify individuals without some properties:

$$\begin{aligned}&\mathsf{EndLane}(x) \leftarrow \mathsf{Lane}(x),\ \mathtt{NOT\ EXISTS}\ y\ \mathtt{IN}\\ &\quad (\mathsf{Lane}(y),\ \mathsf{hasNext}(x,\ y))\end{aligned}$$

More generally, such rules have the form

$$\mathsf{C}_2(x) \leftarrow \mathsf{C}_1(x),\ \mathtt{NOT\ EXISTS}\ y\ \mathtt{IN}\ (\mathsf{C}_1(y),\ \mathsf{op}_1(x,\ y)).$$

(b) Topological rules refer to topological relations, more specifically, lateral (left/right) and longitudinal (predecessor/successor) relations. Reachability can naturally be expressed using recursive rules.

$$\begin{aligned}&\mathsf{hasLeft}(x,y) \leftarrow \mathsf{hasDirectLeft}(x,y)\\ &\mathsf{hasLeft}(x,z) \leftarrow \mathsf{hasDirectLeft}(x,y),\ \mathsf{hasLeft}(y,z).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}&\mathsf{op}_1(x,y) \leftarrow \mathsf{op}_2(x,y)\\ &\mathsf{op}_1(x,z) \leftarrow \mathsf{op}_2(x,y),\ \mathsf{op}_1(y,z)\end{aligned}$$

(c) Distance rules refer to the spatial arrangement of objects, such as the distance to a point of interest. There are two types of distance relations: *coordinate distance* and *length distance*.

Coordinate distance rules indicate the distance between two points using coordinates. An auxiliary concept (CoordinateDistance) represents the ternary relation that connects the source point to the target point via two object properties hasSource and hasTarget and the calculated distance value via the data property distance:

$$\begin{aligned}&\mathsf{CoordinateDistance}(d),\ \mathsf{hasSource}(d,s),\\ &\quad \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\quad \mathsf{Point}(s),\ \mathsf{x}(s,x_s),\ \mathsf{y}(s,y_s),\\ &\quad \mathsf{Point}(t),\ \mathsf{x}(t,x_t),\ \mathsf{y}(t,y_t),\\ &\quad \mathtt{BIND}(\mathtt{sqrt}((x_s-x_t)^2+(y_s-y_t)^2)\ \mathtt{AS}\ z),\\ &\quad \mathtt{BIND}(\mathtt{SKOLEM}(\texttt{"}d\texttt{"},\ s,\ t)\ \mathtt{AS}\ d).\end{aligned}$$

More generally, such rules have the following form, where A represents the auxiliary concept:

$$\begin{aligned}&\mathsf{A}(d),\ \mathsf{hasSource}(d,s),\\ &\quad \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\quad \mathsf{C}_1(s),\ \mathsf{x}(s,x_s),\ \mathsf{y}(s,y_s),\\ &\quad \mathsf{C}_1(t),\ \mathsf{x}(t,x_t),\ \mathsf{y}(t,y_t),\\ &\quad \mathtt{BIND}(\mathtt{sqrt}((x_s-x_t)^2+(y_s-y_t)^2)\ \mathtt{AS}\ z),\\ &\quad \mathtt{BIND}(\mathtt{SKOLEM}(\texttt{"}d\texttt{"},\ s,\ t)\ \mathtt{AS}\ d).\end{aligned}$$

Length distance rules are similar to coordinate distance rules, but the distance is calculated by aggregating the length of intermediate path elements between two points. As a concrete example consider:

$$\begin{aligned}&\mathsf{LengthDistance}(d),\ \mathsf{hasSource}(d,s),\\ &\quad \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\quad \mathsf{Lane}(s),\ \mathsf{length}(s,v),\\ &\quad \texttt{AGGREGATE}(\mathsf{hasNext}(s,p),\ \mathsf{hasNext}(p,t)),\\ &\quad \mathsf{length}(p,l)\ \texttt{ON}\ s\ \texttt{BIND}\ \texttt{SUM}(l)\ \texttt{AS}\ u),\\ &\quad \texttt{BIND}((v+u)\ \texttt{AS}\ z),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}f\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}&\mathsf{A}(d),\ \mathsf{hasSource}(d,s),\\ &\quad \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\quad \mathsf{C}_1(s),\ \mathsf{length}(s,v),\\ &\quad \texttt{AGGREGATE}(\mathsf{op}_1(s,p),\ \mathsf{op}_1(p,t)),\\ &\quad \mathsf{length}(p,l)\ \texttt{ON}\ s\ \texttt{BIND}\ \texttt{SUM}(l)\ \texttt{AS}\ u),\\ &\quad \texttt{BIND}((v+u)\ \texttt{AS}\ z),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}f\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

3.2.4 Algorithmic Rules

These rules are used for decision making and vehicle motion planning, e.g., for determining the most probable path that a vehicle will take, for providing lane-change notifications and for pre-fetching tiles. The rules are modelled as a combination of rules of different type, possibly requiring a specific rule order (Suryawanshi et al., 2019; Qiu et al., 2020).

3.3 Knowledge Dimension: Two Levels of Ontologies

To performance a knowledge abstraction process, there are three main components involved, namely low-level ontologies, a high-level ontology and rules. The *low-level ontologies* represent the different map data formats. The primitive rules produce the basic semantics of instances at the low level. These basic semantics are the fundamental ingredients for the transfer rules, which generate instances in the defined high-level ontology. The *high-level ontology* represents a generic and unified road environment model. Spatial and algorithmic rules are used to reason over the high-level ontology and provide the necessary knowledge for decision making and vehicle motion planing.

Figure 2 shows some of the concepts and relations of the high-level ontology and a low-level ontology representing HD map data based on the NDS specification. A LaneGroup is a set of one or more lanes on a part of a link. A Lane is connected to another lane via a LaneConnector. A LaneBoundary is a line that separates lanes with type and colour information. The corresponding high-level concepts of Lane and LaneBoundary are Lane and LaneMarking, respectively, and the instances of both concepts are generated through low-level instances. The corresponding high-level concept of Link is RoadPart and its instances are generated through the instances of Link. At the high level, the spatial relations among Lane and RoadPart are essential for maintaining dynamic high-level road environmental knowledge (see Figure 3). Figure 4 is a graph representation of the example shown in Figure 3.

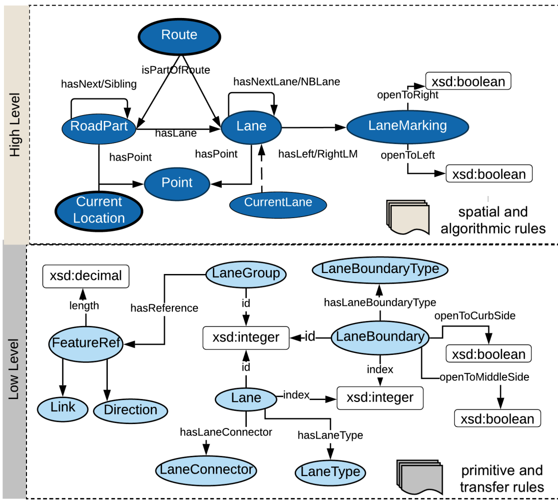


Figure 2: Overview of the two levels of ontologies for representing map data; hasNBLane stands for hasNeighbouringLane; hasLeftLM stands for hasLeftLaneMarking

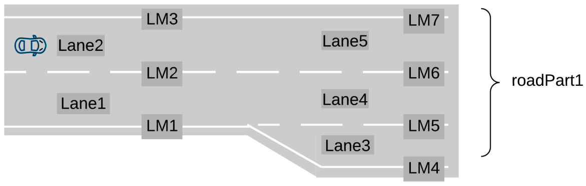


Figure 3: An example of a segment of a road and its lanes and lane markings (LM)

3.4 Spatial Dimension

With the volume and velocity of map data streams, it becomes infeasible to store and process all available information. Hence, only some portion of the available knowledge is stored in memory and only for a limited area. The choice of the stored knowledge is primarily dependent on continuous spatial queries. In addition, we adopt a mechanism to expire (delete) some of the stored knowledge and add information relevant to answer decision-making queries. In our

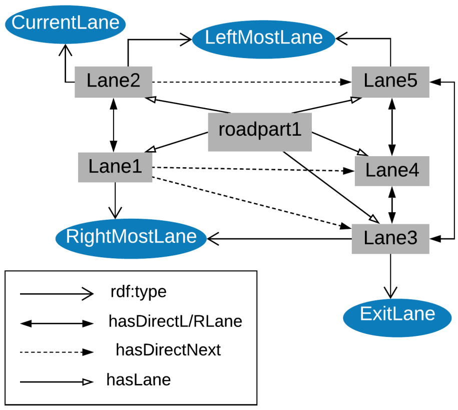
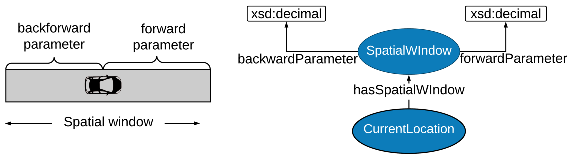


Figure 4: An example of Lane and RoadPart spatial relations; hasDirectL/RLane stands for hasDirectLeft/RightLane corresponding to Figure 3



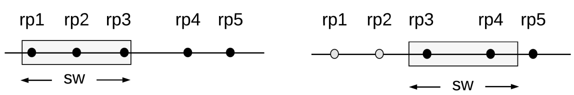
(a) An example of a spatial window (b) The ontological modelling of the spatial window concept

Figure 5: Modelling of the spatial window

approach, we employ a spatial window policy, where the spatial window is defined as a range of an area that satisfies some spatial properties (e.g., distance) with respect to a reference object. We use spatial rules, namely topological and distance rules, to performance spatial reasoning.

The spatial window is relevant to the vehicle location with the forward and backward parameters (see Figure 5 (a)). It is modelled as a class SpatialWindow with two data properties forwardParameter and backwardParameter (see Figure 5 (b)). We define the inside object property of a road part and a spatial window using the rule

$$\begin{aligned}&\mathsf{inside}(r,s) \leftarrow\\ &\quad \mathsf{RoadPart}(r), \mathsf{distanceToVehicle}(r,d),\\ &\quad \mathsf{currentLocation}(c), \mathsf{hasSpatialWindow}(c,s),\\ &\quad \mathsf{SpatialWindow}(s), \mathsf{forwardParameter}(s,f),\\ &\quad \mathsf{backwardParameter}(s,b),\\ &\quad \mathtt{FILTER}(d \leq f \,\&\&\, d \leq b)\end{aligned}$$



(a) A snapshot at location L1 (b) A snapshot at location L2

Figure 6: A example of the spatial window (sw) with respect to locations

The SPARQL query

```
SELECT ?r WHERE {
  ?r rdf:type :RoadPart .
  ?r :inside :SpatialWindow.
}
```

then retrieves road parts that are inside a specific SpatialWindow instance.

Figures 6 (a) and (b) give two snapshots of the vehicle environment at the locations L1 and L2. The objects labelled rp1 – rp5 represent road parts. At the location L1, continuous spatial queries are evaluated over rp1 – rp3 as they are inside the spatial window. At location L2, however, r1 – r2 move out of the spatial window, which means that they are expired. rp3 is a still valid object and rp4 is the new object inside the spatial window. Thus, the continuous spatial queries are evaluated over rp3 and rp4.

4 Evaluation

In this section, we evaluate the adequacy of the proposed knowledge-spatial architecture in our prototype called *SmartMapApp*. The prototype uses RDFox (Oxford Semantic Technologies, 2019) for storing, querying, and reasoning over map data. Then, we present the experimental results for the knowledge abstraction and for the spatial reasoning process.

4.1 Use case

In order for an autonomous vehicles to navigate safely and to ensure the comfort and safety of its passengers, the navigation system needs to constantly request map data streams while the vehicle is progressing along a route. To achieve this goal, the Advanced Driving (AD) system runs a continuous spatial query over the incrementally updated road environmental knowledge to check if the system needs to pre-load map data or delete expired road entities. The concrete scenario is described as following.

The vehicle initialises its world view with some low-level map data, which results in a high-level road view. As soon as the car starts to move, it triggers a continuous pre-fetching query with a spatial window whose forward parameter is set to 5 km. Afterwards, the system pre-loads data for a new map tile based on the answer of pre-fetching query. Consequently, the high-level road view is extended with the new spatial knowledge. While the system incrementally updates the road environmental knowledge, it also continuously checks if any road parts are “out of window" based on the backward parameter (e.g., 3 km) of the

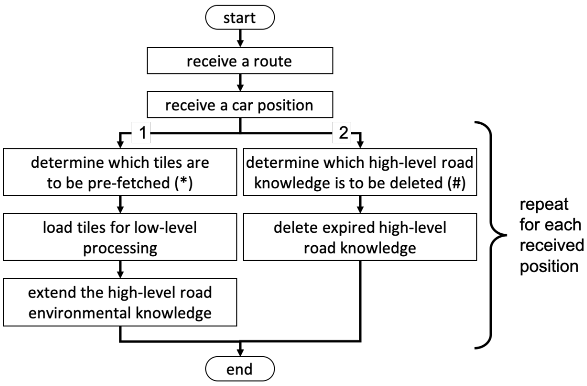


Figure 7: The method for updating the road environment along a route with two parallel processes

spatial window. The road parts which are no longer "inside" the spatial window are deleted.

The method for providing a dynamic road environment along a route is shown in Figure 7. The workflow starts with a received route and vehicle position. Next, process 1 and process 2, are started in parallel. Process 1 ($*$) determines which tiles are required to be pre-fetched and executes the knowledge abstraction process for each tile in parallel. Process 2 (#) determines which high-level road knowledge needs to be deleted. The step marked with ($*$) is modelled as SPARQL query (see Listing 1). The step marked with (#) in process 2 is modeled as a Datalog rule (see Listing 2). The two parallel processes repeat for each received vehicle position.

```
SELECT ?tileId2 ?tileId3
WHERE {
  ?l a :CurrentLane. ?rp :hasLane ?l.
  ?rp :hasNext ?rp12.
  ?rp12 :hasNext ?rp22.
  ?rp22 :length ?length2.
  ?rp12 :isLocatedIn ?tileId2.
  ?rp22 :isLocatedIn ?tileId3.
  ?curreLoct :remDisOfRP ?remainingDis.
  { SELECT ?rp22 (SUM(?len1) AS ?tLen1)
    WHERE {
      ?lane a :CurrentLane.
      ?rp :hasLane ?lane.
      ?rp :hasNext ?rp1.
      ?rp1 :hasNext ?rp22.
      ?rp1 :length ?len1.
    }
  GROUP BY ?rp22
  }
  BIND((?length2+?tLen1) AS ?tLen2)
  BIND((?remainingDis+?tLen2) AS ?dis2)
  :spatialWindow :forwardParameter ?val.
  FILTER(?dis2 < ?val)
}
```

Listing 1: A SPARQL query for pre-fetching map data tiles

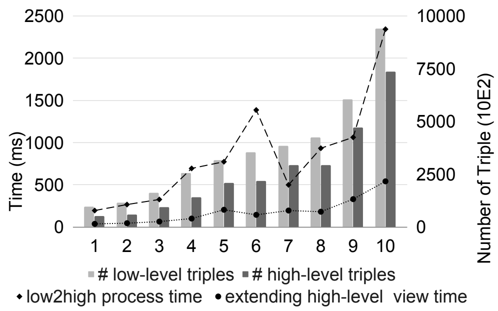


Figure 8: Performance evaluation of the knowledge abstraction process

ExpiredRoadPart$(r) \leftarrow$
  SpatialWindow(s),
  NOT EXISTS r IN (RoadPart(r), inside(r,s))

Listing 2: A Datalog rule for inferring expired road parts

4.2 Performance

We have implemented the proposed knowledge-spatial architecture into an application called SmartMapApp using RDFox 3.0.1 with the provided Java APIs. The used map data covers 63.75 km^2 and is split into 10 data sets for testing. The evaluation was performed on a 64-bit Ubuntu virtual machine with 8 Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz running at 33MHz with 15 GB memory. We recorded the computation time after doing a warm-up run by executing the tasks 3 times sequentially. The results are shown in Figures 8 and 9.

Figure 8 illustrates the knowledge abstraction process. The figure shows, for each data set, the number of low-level triples (grey bar), the resulting number of high-level triples (black bar), the time for transferring triples from the low-level to the high-level ontology (low2high process time, diamonds), and the time for extending the high-level road environmental view (dot).

Figure 9 shows an excerpt of a simulation result along a trace. The left vertical axis shows the execution and deletion time. The execution time (line) is under 50 ms on average. The deletion time (grey area) is almost as low as the execution time. The right vertical axis shows the time for the pre-loading process (orange bar). At position 18, there is a peak of the execution time, which is 104 ms. Additionally, the deletion time (111 ms) and the subsequent pre-loading time (1653 ms) are both long. This indicates that the amount of loaded data has an impact on the execution time.

Overall, the results demonstrate the feasibility of our ontology-based approach to process maps dynam-

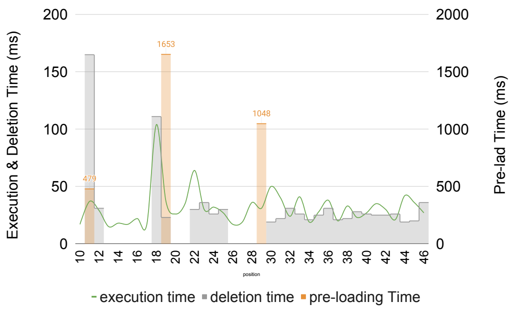


Figure 9: Performance evaluation of the spatial reasoning process

ically for autonomous vehicles. The cost of reasoning generally depends not only on the number of rules, but also on the complexity of the combination of certain rules and the input data. For details of how RDFox performs reasoning, we refer interested readers to the description of the RDFox materialization algorithm (Motik et al., 2015).

5 Discussion

Existing work has shown that Datalog can be translated to SPARQL, and vice versa (Polleres, 2007; Maier et al., 2018) and that it is feasible to use either SPARQL or Datalog for reasoning over streams (Rinne and Nuutila, 2017; Margara et al., 2018). The choice of modeling rules as Datalog or SPARQL is a trade-off between semantic expressibility and computation performance. In particular, it is beneficial to study the evaluation mechanisms of Datalog (materialization) and SPARQL (graph pattern matching). Having the advantages of each language in mind provides a practical guideline to modelling the knowledge.

In our work, the datastores for the low-level ontologies need to perform the knowledge abstraction process by means of primitive, transfer, distance and bounding rules (see Section 3.2) without the necessity to store the facts and inferred knowledge. Based on our experiments, SPARQL queries outperform Datalog rules considering the same desired ADAS functions. Hence, all the rules in the low-level datastores are modelled using SPARQL. However, the high-level datastore needs to store the knowledge and perform incremental updates by means of spatial reasoning. This naturally leads to the modelling of the knowledge as Datalog rules, taking advantage of incremental reasoning with Negation as Failure as supported by RDFox.

6 Conclusions

In this paper, we present a a novel knowledge-spatial architecture considering the knowledge and spatial dimension of map data streams. We discuss the process of knowledge abstraction using two-level ontologies. We present the mechanism of spatial reasoning for building a dynamic map using a spatial-window concept. In particular, we describe how to utilize different types of rules to achieve two dimensional reasoning. We evaluated our SmartMapApp prototype through a running example of providing a dynamic map along a route using RDFox. We chose RDFox as it is the only triple store that can deal with incremental reasoning with Negation as Failure. The results of this paper demonstrate the feasibility of adopting an ontology-based approach for processing map data in autonomous vehicles by using a sophisticated reasoner with the desired scalability, such as RDFox.

In the future, we plan to extend the architecture to ensure map data quality as there might be errors in the raw map data. Expired road objects may also result in missing triples, where a future query might ask for deleted objects. There needs to be a mechanism to cache the expired road objects in order to provide answers to such queries.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of databases*, volume 8.

Armand, A., Ibanez-Guzman, J., and Zinoune, C. (2017). *Digital Maps for Driving Assistance Systems and Autonomous Driving*, pages 201–244.

Bagschik, G., Menzel, T., and Maurer, M. (2018). Ontology based scene creation for the development of automated vehicles. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820.

Belouaer, L., Bouzid, M., and Mouaddib, A. (2010). Ontology based spatial planning for human-robot interaction. In *17th Int. Symposium on Temporal Representation and Reasoning (TIME 2010)*, pages 103–110, Los Alamitos, CA, USA. IEEE Computer Society.

Chen, W. and Kloul, L. (2018). An ontology-based approach to generate the advanced driver assistance use cases of highway traffic. In *KEOD*, pages 73–81.

Cyganiak, R., Wood, D., and Lanthaler, M. (2014). RDF 1.1 Concepts and Abstract Syntax. (Accessed on 09/14/2020).

Gayathri, R. and V., U. (2018). Ontology based knowledge representation technique, domain modeling languages and planners for robotic path planning: A survey. *ICT Express*, 4(2):69 – 74.

Gran, C. W. (2019). Hd-maps in autonomous driving. Master's thesis, NTNU.

Gruyer, D., Belaroussi, R., and Revilloud, M. (2016). Accurate lateral positioning from map data and road marking detection. *Expert Systems with Applications*, 43:1–8.

Gwon, G., Hur, W., Kim, S., and Seo, S. (2017). Generation of a precise and efficient lane-level road map for intelligent vehicle systems. *IEEE Transactions on Vehicular Technology*, 66(6):4517–4533.

Herrtwich, R. (2018). The evolution of the HERE HD Live Map at Daimler. (Accessed on 04/29/2020).

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S. (27 Oct 2009). OWL 2 Web Ontology Language: Primer (2nd Edition).

Hogan, A. (2015). Skolemising blank nodes while preserving isomorphism. In *Proc. of the 24th Int. Conf. on World Wide Web*, WWW '15, page 430–440, Republic and Canton of Geneva, CHE.

Hudelot, C., Atif, J., and Bloch, I. (2008). Fuzzy spatial relation ontology for image interpretation. *Fuzzy Sets and Systems*, 159(15):1929 – 1951. From Knowledge Representation to Information Processing and Management.

ISO/TC 204 (2018). Intelligent transport systems-Local dynamic map. (Accessed on 05/27/2020).

Katsumi, M. and Fox, M. (2018). Ontologies for transportation research: A survey. *Transportation Research Part C: Emerging Technologies*, 89:53 – 82.

Leite, J. (2018). A brief history of gps in-car navigation.

Liu, R., Wang, J., and Zhang, B. (2020). High definition map for automated driving: Overview and analysis. *J. of Navigation*, 73(2):324–341.

Maier, D., Tekle, K. T., Kifer, M., and Warren, D. S. (2018). Datalog: concepts, history, and outlook. In *Declarative Logic Programming: Theory, Systems, and Applications*, pages 3–100. ACM Books.

Margara, A., Cugola, G., Collavini, D., and Dell'Aglio, D. (2018). Efficient temporal reasoning on streams of events with dotr. In *European Semantic Web Conf.*, pages 384–399.

Mokbel, M. F., Xiong, X., Hammad, M. A., and Aref, W. G. (2005). Continuous query processing of spatio-temporal data streams in place. *GeoInformatica*, 9(4):343–365.

Motik, B., Nenov, Y., Piro, R., and Horrocks, I. (2015). Combining rewriting and incremental materialisation maintenance for datalog programs with equality. In *Twenty-Fourth Int. Joint Conf. on Artificial Intelligence*.

NDS Association (2018). The standard for map data - NDS Association. (Accessed on 05/27/2020).

Oxford Semantic Technologies (2019). RDFox knowledge graph system.

Polleres, A. (2007). From sparql to rules (and back). In *Proc. of the 16th Int. Conf. on World Wide Web*, WWW '07, page 787–796, New York, NY, USA.

Qiu, H., Ayara, A., and Glimm, B. (2020). A knowledge architecture layer for map data in autonomous vehicles. In *2020 23rd Int. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE.

Rinne, M. and Nuutila, E. (2017). User-Configurable Semantic Data Stream Reasoning Using SPARQL Update. *J. on Data Semantics*, 6(3):125–138.

Shaw, M., Detwiler, L. T., Noy, N., Brinkley, J., and Suciu, D. (2011). vsparql: A view definition language for the semantic web. *J. of Biomedical Informatics*, 44(1):102–117.

Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197.

Suryawanshi, Y., Qiu, H., Ayara, A., and Glimm, B. (2019). An ontological model for map data in automotive systems. In *IEEE Int. Conf. on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 140–147.

Ulbrich, S., Reschka, A., Rieken, J., Ernst, S., Bagschik, G., Dierkes, F., Nolte, M., and Maurer, M. (2017). Towards a functional system architecture for automated vehicles. *arXiv preprint arXiv:1703.08557*.

W3C SPARQL Working Group (21 Mar 2013). SPARQL 1.1 Overview.

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. (2001). Ontology-based integration of information-a survey of existing approaches.

Zhao, L., Ichise, R., Liu, Z., Mita, S., and Sasaki, Y. (2017). Ontology-based driving decision making: A feasibility study at uncontrolled intersections. *IEICE Transactions on Information and Systems*, E100.D(7):1425–1439.