Formalizing and Evaluating Requirements of Perception Systems for Automated Vehicles using Spatio-Temporal Perception Logic

Mohammad Hekmatnejad, Bardh Hoxha, Jyotirmoy V. Deshmukh, Yezhou Yang, and Georgios Fainekos

Abstract-Automated vehicles (AV) heavily depend on robust perception systems. Current methods for evaluating vision systems focus mainly on frame-by-frame performance. Such evaluation methods appear to be inadequate in assessing the performance of a perception subsystem when used within an AV. In this paper, we present a logic - referred to as Spatio-Temporal Perception Logic (STPL) - which utilizes both spatial and temporal modalities. STPL enables reasoning over perception data using spatial and temporal operators. One major advantage of STPL is that it facilitates basic sanity checks on the functional performance of the perception system, even without ground-truth data in some cases. We identify a fragment of STPL which is efficiently monitorable offline in polynomial time. Finally, we present a range of specifications for AV perception systems to highlight the types of requirements that can be expressed and analyzed through offline monitoring with STPL.

Index Terms—Formal Methods, Perception System, Temporal Logic, Autonomous Driving Systems.

I. INTRODUCTION

The safe operation of automated vehicles (AV), advanced driver assist systems (ADAS), and mobile robots in general, fundamentally depends on the correctness and robustness of the perception system (see discussion in Sec. 3 by Schwarting et al. (2018)). Faulty perception systems and, consequently, inaccurate situational awareness can lead to dangerous situations (Lee (2018); Templeton (2020)). For example, in the accident in Tempe in 2018 involving an Uber vehicle (Lee (2018)), different sensors had detected the pedestrian, but the perception system was not robust enough to assign a single consistent object class to the pedestrian. Hence, the AV was not able to predict the future path of the pedestrian. This accident highlights the importance of identifying the right questions to ask when evaluating perception systems.

In this paper, we argue that in order to improve AV/ADAS safety, we need to be able to formally express what should be the assumptions, performance, and guarantees provided by a perception system. For instance, the aforementioned accident highlights the need for predicting the right object class quickly and robustly. An informal requirement (in natural language)

expressing such a perception system performance expectation could be:

Req. 1: Whenever a new object is detected, then it is assigned a class within 1 sec, after which the object does not change class until it disappears.

Such requirements are not only needed for deployed safety critical systems, but also for existing training and evaluation data sets. For example, a sequence of six image frames from the KITTI dataset is shown in Figure 1. All detected objects with bounding boxes are labeled with class names such as *Car*, *Pedestrian*, and *Cyclist* in those frames. Nonetheless, in Fig. 1(a), a *cyclist* that was detected in frame f = 0 and existed in frame f = 1 changed its class to *pedestrian* in frame f = 2, which is a violation of Req. 1. If such a requirement is too strict for the perception system, then it could be relaxed by requiring that the object is properly classified in at least 4 out of 5 frames (or any other desired frequency). The first challenge for an automated testing & verification framework is to be able to formally (i.e., mathematically) represent such high-level requirements.

Formalizing the expectations on the performance of the perception system has several benefits. First and foremost, such formal requirements can help us evaluate AV/ADAS perception systems beyond what existing metrics used in vision and object tracking can achieve (Yurtsever et al. (2020); Richter et al. (2017)). In particular, evaluation metrics used for image classification algorithms are typically operating on a frame-by-frame basis ignoring cross-frame relations and dependencies. Although tracking algorithms can detect misclassifications, they cannot assess the frequency and severity of them, or more complex spatio-temporal failure conditions.

Second, formal specifications enable automated searchbased testing (e.g., see Abbas et al. (2017); Tuncali et al. (2020); Dreossi et al. (2019b,a); Corso et al. (2020); Gladisch et al. (2019); DeCastro et al. (2020)) and monitoring (e.g., see Rizaldi et al. (2017); Hekmatnejad et al. (2019)) for AV/ADAS. Requirements are necessary in testing in order to express the conditions (assumptions) under which system level violations are meaningful. Similarly, logical requirements are necessary in monitoring since without formal requirements, it is very hard to concisely capture the rules of the road. We envision that a formal specification language which specifically targets perception systems will enable similar innovations in testing and monitoring AV/ADAS.

Third, formal specifications on the perception system can also function as a requirements language between original

Mohammad Hekmatnejad was with the School of Computing and Augmented Intelligence (formerly CIDSE), Arizona State University, USA; email: mhekmatn@asu.edu.

Bardh Hoxha is with the Toyota Research Institute of North America, USA. Jyotirmoy V. Deshmukh is with the University of Southern California, USA. Yezhou Yang is with Arizona State University, USA.

Georgios Fainekos is with the Toyota Research Institute of North America, USA; the work was performed while he was with the Arizona State University, USA.



(f) Frame f = 5 and $\tau(5) = 0.2$

Fig. 1: A series of image frames taken from KITTI (Geiger et al. (2013a)) augmented with the data about classified dataobjects using SqueezeDet (Wu et al. (2017)). Here, τ is a function that maps each frame f to its captured time. equipment manufacturers (OEM) and suppliers. As a simple example for the need of a requirements language consider the most basic question: should the image classifier correctly classify all objects within the sensing range of the lidar, or all objects of at least x pixels in diameter? Again, without requirements, any system behavior is acceptable and, most importantly, not testable/verifiable!

Finally, using a formal specification language, we can search offline perception datasets (see Kim et al. (2022); Anderson et al. (2023); Yadav and Curry (2019)) to find scenarios that violate the requirements in order to reproduce them in simulation, e.g., Bashetty et al. (2020), or to assess the real-to-sim gap in testing, e.g., Fremont et al. (2020).

Even though there is a large body of literature on the application of formal languages and methods to AV/ADAS (see the survey by Mehdipour et al. (2023)), the prior works have certain limitations when perception systems are specifically targeted. Works like Tuncali et al. (2020) and Dreossi et al. (2019a) demonstrated the importance of testing system level requirements for an AV when the AV contains machine learning enabled perception components. Both works use Signal Temporal Logic (STL) (Maler and Nickovic (2004)) for expressing requirements on the performance of the AV/ADAS. However, STL cannot directly express requirements on the performance of the perception system itself.

This limitation was identified by Dokhanchi et al. (2018a) who developed a new logic – Timed Quality Temporal Logic (TQTL) – which can directly express requirements for image classification algorithms like SqueezeDet by Wu et al. (2017). Namely, TQTL was designed to enable reasoning over objects classified in video frames and their attributes. For example, TQTL can formalize Req. 1 that an object should not change classes. However, TQTL does not support spatial or topological reasoning over the objects or regions in a scene. This is a major limitation since TQTL cannot express requirements such as occlusion, overlap, and other spatial relations for basic sanity checks, e.g., that objects do not "teleport" from frame to frame, or that "every 3D bounding box contains at least 1 lidar point".

In this paper, we introduce the Spatio-Temporal Perception Logic (STPL) to address the aforementioned limitations. We combine TQTL with the spatial logic $S4_u^1$ to produce a more expressive logic specifically focused on perception systems. STPL supports quantification among objects or regions in an image, and time variables for referring to specific points in time. It also enables both 2D and 3D spatial reasoning by expressing relations between objects across time. For example with STPL, we can express requirements on the rate that bounding boxes should overlap:

Req. 2: The frames per second of the camera is high enough so that for all detected cars, their bounding boxes self-overlap for at least 3 frames and for at least 10% of the area. which is satisfiable for the image frames in Fig. 1.

Beyond expressing requirements on the functional performance of perception systems, STPL can be used to compare

¹For a historical introduction to $S4_u$ see van Benthem and Bezhanishvili (2007) and Kontchakov et al. (2007).

different machine learning algorithms on the same perception data. For example, STPL could assess how rain removal algorithms (see the work by Sun et al. (2019)) improve the performance of the object detection algorithms over video streams. Along these lines, Mallick et al. (2023) proposed to use TQTL for differential testing between two different models for pedestrian detection in extreme weather conditions. As another example, STPL could evaluate the impact of a point cloud clustering algorithm on a prediction algorithm (see the works by Campbell et al. (2016) and Qin et al. (2016)).

Since STPL is a very expressive formal language $(TQTL+S4_u)$, it is not a surprise that both the offline and online monitoring problems can be computationally expensive. The main challenge in designing a requirements language for perception systems is to find the right balance between expressivity and computability.

STPL borrows time variables and freeze time quantifiers from the Timed Propositional Temporal Logic (TPTL) by Alur and Henzinger (1994). Unfortunately, the time complexity of monitoring TPTL requirements is PSPACE-hard (see Markey and Raskin (2006)). Even though the syntax and semantics of STPL that we introduce in Section IV allow arbitrary use of time variables in the formulas, in practice, in our implementation, we focus on TPTL fragments which are computable in polynomial time (see Dokhanchi et al. (2016); Elgyutt et al. (2018); Ghorbel and Prabhu (2022)).

Another source of computational complexity in STPL is the use of object/region quantification and set operations, i.e., intersection, complementation, and union. Even our limited use of quantifiers (\exists, \forall) over finite sets (i.e., objects in a frame) introduces a combinatorial component to our monitoring algorithms. This is a well known problem in first-order temporal logics that deal with data (e.g., see Basin et al. (2015) and Havelund et al. (2020)). In addition, the computational complexity of checking whether sets, which correspond to objects/regions, satisfy an arbitrary $S4_u$ formula depends on the set representation and the dimensionality of the space (e.g., polytopes Baotic (2009), orthogonal polyhedra Bournez et al. (1999), quadtrees or octrees Aluru (2018), etc).

In this paper, we introduce an offline monitoring algorithm for STPL in Section VI, and We study its computational complexity in Section VI-E. In Section VI-F, we identify a fragment which can remain efficiently computable. Irrespective of the computational complexity of the full logic, in practice, our experimental results in Section VII indicate that we can monitor many complex specifications within practically relevant times.

All in all, to the best of our knowledge, this is the first time that a formal language is specifically designed to address the problem of functional correctness of perception systems within the context of autonomous/automated mobile systems. Our goal is to design a logic that can express functional correctness requirements on perception systems while understanding what is the computational complexity of monitoring such requirements in practice. As a bi-product, we develop a logical framework that can be used to compare different perception algorithms beyond the standard metrics.

In summary, the contributions of this paper are:

- We introduce a new logic Spatio-Temporal Perception Logic (STPL) that combines quantification and functions over perception data, reasoning over timing of events, and set based operations for spatial reasoning (Section IV).
- We present examples of STPL specifications of increasing complexity as a tutorial (Section V), and we demonstrate that STPL can find label inconsistencies in publicly available AV datasets (see Figure 7).
- We introduce an offline monitoring algorithm (Section VI), and we propose to use different set representations for efficient computations (Section VI-E).
- We study the time complexity of the offline monitoring problem for STPL formulas, and we propose fragments for which the monitoring problem is computable in polynomial time (Section VI-E).
- 5) We experimentally study offline monitoring runtimes for a range of specifications and present the trade offs in using the different STPL operators (Section VII).
- 6) We have released a publicly available open-source toolbox (STPL (2023)) for offline monitoring of STPL specifications over perception datasets. The toolbox allows for easy integration of new user-defined functions over time and object variables, and new data structures for set representation.

One common challenge with any formal language is its accessibility to users with limited experience in formal requirements. In order to remove such barriers, Anderson et al. (2022) developed PyFoReL which is a domain specific language (DSL) for STPL inspired by Python and integrated with Visual Studio Code. In addition, our publicly available toolbox contains all the examples and case studies presented in this paper which can be used as a tutorial by users without prior experience in formal languages. One can further envision integration with Large Language Models for direct use of natural language for requirement elicitation similar to the work by Pan et al. (2023); Fuggitti and Chakraborti (2023); Cosler et al. (2023) for signal or linear temporal logic.

Finally, even though the examples in the paper focus on perception datasets from AV/ADAS applications (i.e., Motional (2019); Geiger et al. (2013b)), STPL can be useful in other applications as well. One such example could be in robot manipulation problems where mission requirements are already described in temporal logic (e.g., see He et al. (2018) and He et al. (2015)).

II. PRELIMINARIES

We start by introducing definitions that are used throughout the paper for representing classified data generated by perception subsystems. In the following, an ego car (or the Vehicle Under Test – VUT) is a vehicle equipped with perception systems and other automated decision making software components, but not necessarily a fully automated vehicle (e.g., SAE automation level 4 or 5).

A. Data-object Stream

A *data-object stream* D is a sequence of data-objects representing objects in the ego vehicle's environment. Such

data objects could be the output of the perception system, or ground truth data, or even a combination of these. The ground truth can be useful to analyze the precision of the output of a perception system. That is, given both the output of the perception system and the ground truth, it is possible to write requirements on how much, how frequently, and under what conditions the perception system is allowed to deviate from the ground truth. In all the examples and case studies that follow, we assume that we are either given the output of the perception system, or the ground truth – but not both. In fact, we treat ground truth data as the output of a perception system in order to demonstrate our framework. Interestingly, in this way, we also discover inconsistencies in the labels of training data sets for AV/ADAS (see Fig. 7).

We refer to each dataset in the sequence as a frame. In each frame, objects are classified and stored in a data-object. Data-objects are abstract because in the absence of a standard classification format, the output of different perception systems are not necessarily compatible with each other. To simplify the presentation, we will also refer to the order of each frame in the data stream as a frame number. For each frame i, $\mathcal{D}(i)$ is the set of data-objects for frame i, where i is the order of the frame in the data-object stream \mathcal{D} . We assume that a data stream is provided by a perception system, in which a function \mathcal{R} retrieves data-attributes for an identified object. A *dataattribute* is a property of an object such as class, probability, geometric characteristics, etc. Some examples could be:

- by R(D(i), id).Class, we refer to the class of an object identified by id in the i'th frame,
- by $\mathcal{R}(\mathcal{D}(i), id)$. *Prob*, we refer to the probability that an object identified by *id* in the *i*'th frame is correctly classified,
- by $\mathcal{R}(\mathcal{D}(i), id).PC$, we refer to the point clouds associated with an object identified by id in the *i*'th frame.

The function OI returns the set of identifiers for a given set of data-objects. By OI(D(i)), we refer to all the identifiers that are assigned to the data-objects in the *i*'th frame. Another important attribute of each frame is its time stamp, i.e., when this frame was captured in physical time. We represent this attribute for the frame *i* by $\tau(i)$.

Example 2.1: Assume that data-object stream \mathcal{D} is represented in Figure 1, then the below equalities hold:

- $\mathcal{OI}(\mathcal{D}(1)) = \{1, 2, 3\}$
- $\mathcal{R}(\mathcal{D}(1), 2)$.Class = cyclist
- $\mathcal{R}(\mathcal{D}(1), 2).Prob = 0.57$
- $\tau(1) = 0.04$

Finally, we remark that in offline monitoring, which is the application that we consider in this paper, the data stream \mathcal{D} is finite. We use the notation $|\mathcal{D}|$ to indicate the total number of frames in the data stream.

B. Topological Spaces

Throughout the paper, we will be using \mathbb{N} to refer to the set of natural numbers and \mathbb{R} to real numbers. In the following, for completeness, we present some definitions and notation related to topological spaces and the $S4_u$ logic which are adopted



Fig. 2: Let $X = X_1 \cup X_2$ with $\mathbf{C}X = X$. Left: the whole rectangle represents the universe (colored gray) \mathbb{U} , the regions X_1 and X_2 without the dashed-lines (colored light blue) represent the interior $\mathbf{I}X$ of the set X, the dashed lines (colored darker blue) represent the boundary $\mathbf{C}X - \mathbf{I}X$ of the set X, and the remaining region represents $\mathbb{U} - X$. Right: the tightest bounding box around set X that includes the four unique elements of the set as in Def. 2.4.

from Gabelaia et al. (2005) and Kontchakov et al. (2007). Later, we build our $MTL \times S4_u$ definition on this notation.

Definition 2.1 (Topological Space): A topological space is a pair $\mathbf{T} = \langle \mathbb{U}, \mathbf{I} \rangle$ in which \mathbb{U} is the universe of the space, and \mathbf{I} is the interior operator on \mathbb{U} . The universe is a nonempty set, and \mathbf{I} satisfies the standard Kuratowski axioms where $X, Y \subseteq \mathbb{U}$:

$$\mathbf{I}(X \cap Y) = \mathbf{I}X \cap \mathbf{I}Y, \ \mathbf{I}X = \mathbf{I}\mathbf{I}X, \ \mathbf{I}X \subseteq X, \ \text{and} \ \mathbf{I}\mathbb{U} = \mathbb{U}.$$

We denote C (closure) as the dual operator to I, such that CX = U - I(U - X) for every $X \subseteq U$. Below we list some remarks related to the above definitions:

- IX is the *interior* of a set X,
- CX is the *closure* of a set X,
- X is called open if X = IX,
- X is called *closed* if $X = \mathbf{C}X$,
- If X is an open set then its complement X
 = U − X is a closed set and vice versa,
- For any set X ⊆ U, its *boundary* is defined as CX IX (X and its complement have the same boundary),

C. Image Space Notation and Definitions

Even though our work is general and applies to arbitrary finite dimensional spaces, our main focus is on 2D and 3D spaces as encountered in perception systems in robotics. In the following, the discussion focuses on 2D image spaces with the usual pixel coordinate systems. In Figure 2 left, a closed set is illustrated as the light blue region, while its boundary is a dashed-line in darker blue.

Definition 2.2 (Totally-ordered Topological Space): A Totally-ordered (TO) topological space T is a topological space $\langle \mathbb{U}, \mathbf{I} \rangle$ that is equipped with a total ordering relation $\leq \subseteq \mathbb{U} \times \mathbb{U}$. That is, $\forall p_1, p_2 \in \mathbb{U}$ either $p_1 \leq p_2$ or $p_2 \leq p_1$.

Definition 2.3 (Total order in 2D spaces): In a twodimensional (2D) TO topological space \mathbf{T} , $p = (x, y) \in \mathbb{U}$ denotes its coordinates in the x-y Cartesian coordinate system, where $x, y \in \mathbb{R}_{\geq 0}$. For $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ the ordering relation is defined as:

$$(x_1, y_1) \leq_{2D} (x_2, y_2) \iff y_1 < y_2 \lor (y_1 = y_2 \land x_1 \leq x_2)$$

Note that even though Def. 2.3 considers the coordinates over the reals, i.e., $\mathbb{R}^2_{\geq 0}$, in practice, the image space is defined over the integers, i.e., \mathbb{N}^2 (pixel coordinates). The topological space in Figure 2 consists of the universe which is all the pixels that belong to the whole 2D gray-rectangle and the closure operator that for any set of regions includes their edges. Assume that the left-upper corner of the image is the origin of the x - y Cartesian coordinate system, and the x - axisand the y - axis are along the width and height of the image, respectively. This is the standard coordinate system in image spaces. Then, any pixel that belongs to the universe has an order with respect to the other pixels. For example, consider $X = X_1 \cup X_2$ in Fig. 2, then all the pixels that belong to X_2 have higher orders than those in X_1 .

Definition 2.4 (Spatial Ordering Functions): Given a closed set $U \subseteq \mathbb{U}$ from a 2D TO topological space $\mathbf{T} = \langle \mathbb{U}, \mathbf{I} \rangle$, we define the top-most, bottom-most, left-most, right-most, and center-point functions by TM, BM, LM, RM.CT : $2^{\mathbb{U}} \to \mathbb{U}$, respectively such that

$$\begin{aligned} \mathsf{TM}(U) &= (x_p, y_p) \in U \text{ s.t.} \\ &\forall s = (x_s, y_s) \in \mathbb{U}, y_p < y_s \lor (y_p = y_s \land x_p \ge x_s) \\ \mathsf{BM}(U) &= (x_p, y_p) \in U \text{ s.t.} \\ &\forall s = (x_s, y_s) \in \mathbb{U}, y_p > y_s \lor (y_p = y_s \land x_p \le x_s) \\ \mathsf{LM}(U) &= (x_p, y_p) \in U \text{ s.t.} \\ &\forall s = (x_s, y_s) \in \mathbb{U}, x_p < x_s \lor (x_p = x_s \land y_p \le y_s) \\ \mathsf{RM}(U) &= (x_p, y_p) \in U \text{ s.t.} \\ &\forall s = (x_s, y_s) \in \mathbb{U}, x_p > x_s \lor (x_p = x_s \land y_p \ge y_s) \end{aligned}$$

and, CT(U) is the center point of the rectangle defined by LM(U), RM(U), BM(U), and TM(U) points.

In Def. 2.4, the notation 2^A for a set A denotes the set of all subsets of A, i.e., the powerset of A. In Figure 2, we have identified the left-most, right-most, bottom-most, and right-most elements of the set X using the LM(X), TM(X), RM(X), and BM(X) function definitions. The right rectangle in Figure 2 is the tightest bounding box for the set X, and it can be derived using only those four points.

D. Spatio-Temporal Logic $MTL \times S4_u$

Next, we introduce the logic $MTL \times S4_u$ over discrete-time semantics. $MTL \times S4_u$ is a combination of Metric Temporal Logic (MTL) (Koymans (1990)) with the $S4_u$ logic of topological spaces. From another perspective, it is an extension to the logic $PTL \times S4_u$ (see Gabelaia et al. (2005)) by adding time/frame intervals into the spatio-temporal operators. Even though $MTL \times S4_u$ is a new logic introduced in this paper, we present it in the preliminaries section in order to gradually introduce notation and concepts needed for STPL. In this paper, we use the Backus-Naur form (BNF) (see Perugini (2021)) which is standard for providing the grammar when we define the syntax of formal and programming languages.

In the following, we provide the formal definitions for $MTL \times S4_u$. Henceforth, the symbol p refers to a spatial proposition, i.e., it represents a set in the topological space.

In addition, the symbol \mathcal{T} represents spatial expressions that evaluate to subsets of the topological space. We refer to \mathcal{T} as a *spatial term*.

Definition 2.5 (Syntax of $MTL \times S4_u$ as a temporal logic of topological spaces): Let Π be a finite set of spatial propositions over $\mathbf{T} = \langle \mathbb{U}, \mathbf{I} \rangle$, then a formula φ of $MTL \times S4_u$ can be defined as follows:

$$\mathcal{T} \coloneqq p \mid \overline{\mathcal{T}} \mid \mathcal{T} \sqcap \mathcal{T} \mid \mathbf{I} \mathcal{T} \mid \mathcal{T} \mathcal{U}_{\mathcal{I}}^{s} \mathcal{T} \mid \bigcirc_{\mathcal{I}}^{s} \mathcal{T}$$
$$\varphi \coloneqq \exists \mathcal{T} \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \mathcal{U}_{\mathcal{I}} \varphi \mid \bigcirc_{\mathcal{I}} \varphi$$

where $p \in \Pi$ is a spatial proposition.

In the above definition, the grammar for $MTL \times S4_u$ consists of two sets of production rules \mathcal{T} and φ . The spatial production rule \mathcal{T} in Def. 2.5 contains a mix of spatial $(\bar{,} \sqcap, \mathbf{I})$ and spatio-temporal $(\mathcal{U}_{\mathcal{I}}^s, \bigcirc_{\mathcal{I}}^s)$ operators. Here, $\overline{\mathcal{T}}$ is the spatial complement of the spatial term \mathcal{T} , \sqcap is the spatial intersection operator, and I is the spatial interior operator. The $\mathcal{U}_{\mathcal{I}}^s$ and $\bigcirc_{\mathcal{T}}^s$ operators are the spatio-temporal until and the spatiotemporal next, respectively. Here, the subscript \mathcal{I} denotes a non-empty interval of $\mathbb{R}_{\geq 0}$ and captures any timing constraints for the operators. When timing constraints are not needed, then we can set $\mathcal{I} = [0, +\infty)$, or remove the subscript \mathcal{I} from the notation. Intuitively, the spatio-temporal until and next operators introduce spatial operations over time. For instance, the expression $p_1 \mathcal{U}_{[1,3]}^s p_2$ computes the union over all sets resulting by the intersection of each occurrence of set p_2 at some time in the interval [1,3] with all the sets p_1 up to that time (see Def. 2.6 for more details). We refer to the spatiotemporal operators $\mathcal{U}^s_{\mathcal{T}}$ and $\bigcirc^s_{\mathcal{T}}$, as Spatio-Temporal Evolving (STE) operators. Also, we call a formula STE if it has STE operators in it. Similarly, we call a spatio-temporal formula Spatial Purely Evolving (SPE) if there is no STE operator in the formula. For more information refer to the (OC) and (PC) definitions by Gabelaia et al. (2005).

In the production rule φ in Def. 2.5, the *spatially exists* \exists checks if the spatial term following it evaluates to a non-empty set. That is, $\exists \mathcal{T}$ checks if there exist some points in the set represented by \mathcal{T} . In φ , except for the *spatially exists* \exists , the other operators are the same as in MTL. That is \neg , \land , $\mathcal{U}_{\mathcal{I}}$ and $\bigcirc_{\mathcal{I}}$ are the negation, conjunction, timed until, and next time operators, respectively (see the review by Bartocci et al. (2010)). As an example of a simple formula in MTL×S4_u, the expression $\exists p_1 \mathcal{U}_{[0.5,2.5]} \exists p_2$ is true if the set represented by p_2 is nonempty at some point in time between [0.5, 2.5] and until then the set represented by p_1 is non-empty.

In the following, we define the bounded discrete-time semantics for MTL× $S4_u$. The definition uses the spatial semantics of $S4_u$ while extending the temporal fragment (PTL) with time constraints over finite traces as in MTL. The semantics are defined over a data-object stream \mathcal{D} . However, for consistency with PTL× $S4_u$, we will assume the existence of a spatio-temporal valuation function $\mathfrak{U} : \Pi \times \mathbb{N} \to 2^{\mathbb{U}}$ that associates with every proposition p and time frame i a subset of the topological space. In the definition of STPL in Section IV, the sets $\mathfrak{U}(p, i)$ will correspond to identified objects in the environment, i.e., bounding boxes, bounding rectangles, or even regions in semantic segmentation. In this section, the the sets $\mathfrak{U}(p, i)$ are just arbitrary subsets of the universe. Definition 2.6 (Quantified Topological Temporal Model and Valuation): A Quantified Topological Temporal Model (QTTM) is a tuple of the form $\mathbf{Q} = \langle \mathbf{T}, \mathfrak{U}, \mathcal{D}, \tau \rangle$, where $\mathbf{T} = \langle \mathbb{U}, \mathbf{I} \rangle$ is a TO topological space, \mathfrak{U} is a spatio-temporal valuation function, \mathcal{D} is a data-object stream of size $|\mathcal{D}|$, $\tau : \mathbb{N} \to \mathbb{R}^+$ maps frame numbers to their physical times, and \mathcal{I} is any non-empty interval of $\mathbb{R}_{\geq 0}$ over time.

Given a model \mathbf{Q} , the valuation function $\mathbf{V}(p, \mathcal{D}, i, \tau)$ represents a subset of the topological space \mathbf{T} that is occupied by spatial proposition $p \in \Pi$ in the *i*'th frame (e.g., $\tau(i)$ is the captured time). The valuation \mathbf{V} is inductively extended to any formulas that can be produced by the production rule \mathcal{T} in the Def. 2.5 as follows:

$$\begin{split} \mathbf{V}(p,\mathcal{D},i,\tau) &\coloneqq \mathfrak{U}(p,i) \\ \mathbf{V}(\mathcal{T}_{1} \ \mathcal{U}_{\mathcal{I}}^{s} \ \mathcal{T}_{2},\mathcal{D},i,\tau) &\coloneqq \bigcup_{i' \in \{j \in \mathbb{N} \ | \ \tau(j) \in (\tau(i) + \mathcal{I})\}} \\ & \left(\mathbf{V}(\mathcal{T}_{2},\mathcal{D},i',\tau) \cap \bigcap_{i \leq i'' < i'} \mathbf{V}(\mathcal{T}_{1},\mathcal{D},i'',\tau) \right) \\ \mathbf{V}(\bigcirc_{\mathcal{I}}^{s} \ \mathcal{T},\mathcal{D},i,\tau) &\coloneqq \\ & \left\{ \begin{array}{c} \mathbf{V}(\mathcal{T},\mathcal{D},i+1,\tau) & \text{if } i+1 < |\mathcal{D}|, \tau(i+1) \in (\tau(i) + \mathcal{I}) \\ \varnothing & \text{otherwise (i.e., an empty set)} \end{array} \right. \\ & \mathbf{V}(\mathcal{T}_{1} \sqcap \mathcal{T}_{2},\mathcal{D},i,\tau) \coloneqq \mathbf{V}(\mathcal{T}_{1},\mathcal{D},i,\tau) \cap \ \mathbf{V}(\mathcal{T}_{2},\mathcal{D},i,\tau) \\ & \mathbf{V}(\overline{\mathcal{T}},\mathcal{D},i,\tau) \coloneqq \overline{\mathbf{V}(\mathcal{T},\mathcal{D},i,\tau)} \\ & \mathbf{V}(\mathbf{I} \ \mathcal{T},\mathcal{D},i,\tau) \coloneqq \mathbf{I} \ \mathbf{V}(\mathcal{T},\mathcal{D},i,\tau) \end{split}$$

where $t + \mathcal{I} = \{t'' \mid \exists t' \in \mathcal{I} . t'' = t + t'\}.$

The valuation function V definition is straightforward for the spatial operations, i.e., \neg, \neg, \mathbf{I} ; V is just applying the corresponding set theoretic operations, i.e., \neg, \neg, \mathbf{I} . The more interesting cases are the spatio-temporal $(\mathcal{U}_{\mathcal{I}}^s \bigcirc_{\mathcal{I}}^s)$ operators. The spatial-next operator $\bigcirc_{\mathcal{I}}^s \mathcal{T}$ first checks if the next sample (i+1) satisfies the timing constraints, i.e., $\tau(i+1) \in (\tau(i)+\mathcal{I})$, and if so, it returns the set that \mathcal{T} represents at time (i+1). The spatial until is a little bit more involved and it is better explained through derived operators. In the following, we define some of the commonly used derived operators:

- The spatially for all operator \forall checks if the spatial expression \mathcal{T} represents a set which is the same as the universe: $\forall \mathcal{T} \equiv \neg \exists \overline{\mathcal{T}}$,
- The spatial union operator: $\mathcal{T}_1 \sqcup \mathcal{T}_2 \equiv \overline{\mathcal{T}_1} \sqcap \overline{\mathcal{T}_2}$,
- The spatial closure operator: $\mathbf{C} \mathcal{T} \equiv \mathbf{I} \overline{\mathcal{T}}$,
- The spatial eventually operator: $\Diamond_{\mathcal{I}}^s \mathcal{T} \equiv \mathbb{U} \mathcal{U}_{\mathcal{I}}^s \mathcal{T}$,
- The spatial always operator: $\Box_{\mathcal{I}}^s \mathcal{T} \equiv \Diamond_{\mathcal{I}}^s \overline{\mathcal{T}}$.

Notice that in the definition of the spatial eventually operator, we used the universe set \mathbb{U} as a terminal even though the syntax in Def. 2.5 does not explicitly allow for that. The universe (\mathbb{U}) and the empty set (\emptyset) can be defined as derived spatial expressions, i.e., $\mathbb{U} = p \sqcup \overline{p}$ and $\emptyset = p \sqcap \overline{p}$. Therefore, if we replace \mathbb{U} in the definition of \mathbf{V} for $\mathcal{U}_{\mathcal{I}}^s$, we can observe that $\diamond_{\mathcal{I}}^s$ corresponds to the spatial union of the expression \mathcal{T} over the time interval \mathcal{I} .

Example 2.2: A simple example is presented in Fig. 3 for a data-stream with 4 frames. The spatial expression $\diamond^s p$ corresponds to the union of all the sets represented by p over time (gray set in Fig. 3) since there are no timing constraints.



Fig. 3: Left: the evolution of a spatial proposition p over four frames with $\tau(0) = 0$, $\tau(1) = 0.4$, $\tau(0) = 0.8$, $\tau(0) = 1.2$; **Right**: *Gray*: the set resulting from $\diamondsuit^s p$, and *Black*: the set resulting from $\Box_{[0,1]}^s p$.

On the other hand, the spatial expression $\Box_{[0,1]}^s p$ corresponds to the intersection of the sets of p at frames i = 0, 1, 2 since the last frame (i = 3) with $\tau(3) = 1.2$ does not satisfy the timing constraints [0,1]. The set $\Box_{[0,1]}^s p$ is represented as a black box in Fig. 3.

Given the definition of the valuation function V for QTTM, we can proceed to define the semantics of MTL× $S4_u$. Recall that the valuation of spatial expressions returns sets from some topological space. On the other hand, MTL× $S4_u$ formulas are interpreted over Boolean values True (\top) / False (\bot), i.e., the formulas are satisfied or are not satisfied. To evaluate MTL× $S4_u$ formulas, we use a valuation function [[φ]] which takes as an input an MTL× $S4_u$ formula φ , a data stream \mathcal{D} , a sample *i*, and a timestamp function τ , and returns True (\top) or False (\bot).

Definition 2.7 (MTL×S4_u semantics): Given an MTL×S4_u formula φ , a QTTM $\mathbf{Q} = \langle \mathbf{T}, \mathfrak{U}, \mathcal{D}, \tau \rangle$, and a frame $i \in \mathbb{N}$, the semantics of φ are defined recursively as follows:

$$\begin{split} & \llbracket \exists \mathcal{T} \rrbracket (\mathcal{D}, i, \tau) \coloneqq \begin{cases} \top & \text{if } \mathbf{V}(\mathcal{T}, \mathcal{D}, i, \tau) \neq \varnothing \\ \bot & \text{otherwise (i.e., an empty set)} \end{cases} \\ & \llbracket \neg \phi \rrbracket (\mathcal{D}, i, \tau) \coloneqq \neg \llbracket \phi \rrbracket (\mathcal{D}, i, \tau) \\ & \llbracket \phi_1 \land \phi_2 \rrbracket (\mathcal{D}, i, \tau) \coloneqq \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \tau) \land \llbracket \phi_2 \rrbracket (\mathcal{D}, i, \tau) \\ & \llbracket \phi_1 \ \mathcal{U}_{\mathcal{I}} \ \phi_2 \rrbracket (\mathcal{D}, i, \tau) \coloneqq \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \tau) \land \llbracket \phi_2 \rrbracket (\mathcal{D}, i, \tau) \\ & \llbracket \phi_1 \ \mathcal{U}_{\mathcal{I}} \ \phi_2 \rrbracket (\mathcal{D}, i, \tau) \coloneqq \end{split} \\ & \bigvee_{j \in \{k \in \mathbb{N} \mid \tau(k) \in (\tau(i) + \mathcal{I})\}} \left(\llbracket \phi_2 \rrbracket (\mathcal{D}, j, \tau) \land \bigwedge_{i \leq k < j} \llbracket \phi_1 \rrbracket (\mathcal{D}, k, \tau) \right) \\ & \llbracket \bigcirc_{\mathcal{I}} \phi \rrbracket (\mathcal{D}, i, \tau) \coloneqq \\ & \int \llbracket \phi \rrbracket (\mathcal{D}, i + 1, \tau) & \text{if } i + 1 < |\mathcal{D}|, \tau(i+1) \in (\tau(i) + \mathcal{I}) \\ & \downarrow & \text{otherwise} \end{cases} \end{split}$$

Notice that the definitions of the propositional and temporal operators closely match the definitions of the spatial and spatio-temporal operators where set operations (union and intersection) have been replaced by Boolean operations (disjunction and conjunction). Therefore, similar to the spatial expressions, we can define *disjunction* as $\varphi_1 \lor \varphi_2 \equiv \neg(\neg \varphi_1 \land \neg \varphi_2)$, eventually as $\Diamond_{\mathcal{I}} \varphi \equiv \top \mathcal{U}_{\mathcal{I}} \varphi$, and always as $\Box_{\mathcal{I}} \varphi \equiv \neg \Diamond_{\mathcal{I}} \neg \varphi$. Finally, the constant *true* is defined as $\top \equiv [\forall] \mathbb{U}$.

Remark 2.8: In some specifications, it is easier to formalize a requirement using frame intervals and reason over frames rather than time intervals. To highlight this option, we add a tilde on top of the spatio-temporal operators with frame intervals, i.e., in $\tilde{\mathcal{U}}_{\mathcal{I}}^s$, $\tilde{\bigcirc}_{\mathcal{I}}^s$, $\tilde{\mathcal{U}}_{\mathcal{I}}$, and $\tilde{\bigcirc}_{\mathcal{I}}$ the interval \mathcal{I} is over frame interval. When reasoning over frames, $\tau(i)$ equals *i*, i.e., τ is the identity function.

Example 2.3: Revisiting Example 2.2, we can now introduce and explain some MTL× $S4_u$ formulas. The formulas $\exists \diamondsuit^s p$ and $\exists \square_{[0,1]}^s p$ evaluate to true since the sets $\diamondsuit^s p$ and $\square_{[0,1]}^s p$ are not empty. On the other hand, the formula $\exists \square^s p$ is false because the set that corresponds to $\square^s p$ is empty (in Fig. 3 there is no common subset for p across all frames). Notice that the MTL× $S4_u$ formula $\square^s \exists p$ is true since the set p is not empty at every frame. Finally, considering time stamps versus frames, the set $\square_{[0,1]}^s p$, which considers timing constraints, is the same as the set $\square_{[0,2]}^s p$, which considers frame constraints.

III. PROBLEM DEFINITION

Given a data stream \mathcal{D} as defined in II-A, the goal of this paper is to:

- 1) formulate object properties in such a data stream, and
- 2) monitor satisfiability of formulas over the stream.

A. Assumptions

Given a data stream \mathcal{D} , we assume that a tracking perception algorithm uniquely assigns identifiers to classified objects for all the frames (see the work by Gordon et al. (2018)).

In order to relax this assumption, we need to enable probabilistic reasoning within the spatio-temporal framework, which is out of the scope of this work. However, our framework could do some basic sanity checks about the relative positioning of the objects during a series of frames to detect misidentified objects.

This assumption is only needed for some certain types of requirements and for the rest it can be lifted.

B. Overall Solution

We define syntax and semantics of Spatio-Temporal Perception Logic (STPL) over sets of points in topological space, and quantifiers over objects. We build a monitoring algorithm over TPTL, MTL, and $S4_u$ and show the practicality, expressivity and efficiency of the algorithm by presenting examples. This is a powerful language and monitoring algorithm with many applications for verification and testing of complex perception systems. Our proposed language and its monitoring algorithm are different than prior works as we discussed in the introduction (see also related works in Section VIII), and briefly summarized below. STPL

- reasons over spatial and temporal properties of objects through functions and relations;
- extends the reasoning of prior set-based logics by equipping them with efficient offline monitoring algorithm tools;
- focuses on expressing functional requirements for perception systems; and
- is supported by open source monitoring tools.

IV. SPATIO-TEMPORAL PERCEPTION LOGIC

In this section, we present the syntax and semantics of the Spatio-Temporal Perception Logic (STPL). Our proposed logic evaluates the geometric relations of objects in a data stream and their evolution over time. Theoretically, any setbased topological space can be used in our logic. However, we focus on topological spaces and geometric operations relevant to applications related to perception.

Next, we are going to define and interpret STPL formulas over data-object streams. We define our topological space to be axis aligned rectangles in 2D images, or arbitrary polyhedral sets (potentially boxes) in 3D environments. An axis aligned rectangle can be represented by a set of two points that correspond to two non-adjacent corners of the rectangle with four sides each parallel to an axis. A polyhedral is a set formed by the intersection of a finite number of closed half spaces. We assume that this information is contained in the perception datastream as annotations or attributes for each object or region identified by the perception system. For instance, $\mathcal{R}(\mathcal{D}(i), id)$. \mathcal{CS} may store a rectangular axis-aligned convex-polygon that is associated with an object *id* in the *i*'th frame. Alternatively, $\mathcal{R}(\mathcal{D}(i), id)$. \mathcal{CS} may store a convexpolyhedron. Without loss of generality, we will typically use definitions for 2D spaces (image spaces), and later in the case study, we will use bounding volumes (3D spaces).

A. STPL Syntax

The syntax of STPL is based on the syntax of TQTL (Dokhanchi et al. (2018a)) and $S4_u$ (Gabelaia et al. (2005)).

In the context of STPL, the spatial propositions are the symbols that correspond to objects or regions in the perception dataset. We use the function symbol σ to map these objects to their corresponding sets. Hence, the syntax of spatial terms \mathcal{T} in STPL is the same as in MTL× $S4_u$, but the function symbols σ replace the spatial propositions p. In addition, we add grammar support under production rule \mathcal{A} for area computation for spatial terms. Area computation is needed in standard performance metrics for 2D vision algorithms, for example to compute union over intersection (UoI). In the same spirit, we can also support volume computation for spatial terms in 3D spaces.

For many requirements, we also need to access other attributes of the objects in the datastream, e.g., classes, probabilities, estimated velocities, or even compute some basic functions on object attributes, e.g., distances between bounding boxes. For usability, we define some basic functions to retrieve data and compute the desired properties. The functions which are currently supported are: object class (C), class membership probability (P), latitude (Lat) and longitude (Lon) coordinates of a point (CRT), area of bounding box (Area), and distance (Dist) between two points (CRT). We provide the syntax for using such functions under the production rule Θ .

This set of functions is sufficient to demonstrate the generality of our logic. However, a more general approach would be to define arithmetic expressions over user definable functions which can retrieve any desired data from the datastream. Such functionality support is going to be among the goals of future software releases.

Definition 4.1 (STPL Syntax over Discrete-Time Signals): Let V_t and V_o be sets of time variables and object variables, respectively. Assume that $x \in V_t$ is a time variable, $id \in V_o$ is an object variable, \mathcal{I} is any non-empty interval of $\mathbb{R}_{\geq 0}$ over time. The syntax for Spatio-Temporal Perception Logic (STPL) formulas is provided by the following grammar starting from the production rule ϕ :

The syntax for spatial terms is:

 $\mathcal{T} \coloneqq \sigma(id) \mid \overline{\mathcal{T}} \mid \mathcal{T} \sqcap \mathcal{T} \mid \mathbf{I} \mathcal{T} \mid \mathcal{T} \mathcal{U}_{\mathcal{I}}^{s} \mathcal{T} \mid \bigcirc_{\mathcal{I}}^{s} \mathcal{T}$

The syntax for the functions that compute the area of a spatial term are:

$$\mathcal{A} \coloneqq Area(\mathcal{T}) \sim r \mid Area(\mathcal{T}) \sim r \times Area(\mathcal{T})$$

The atomic propositions that represent coordinates for a bounding box are:

$$CRT ::= LM | RM | TM | BM | CT$$

The syntax for spatio-temporal functions are:

$$\begin{split} \Theta &\coloneqq Dist(id, \text{CRT}, id, \text{CRT}) \sim r \mid \\ Lat(id, \text{CRT}) \sim r \mid Lon(id, \text{CRT}) \sim r \mid \\ Lat(id, \text{CRT}) \sim r \times Lat(id, \text{CRT}) \mid \\ Lon(id, \text{CRT}) \sim r \times Lon(id, \text{CRT}) \mid \\ Lat(id, \text{CRT}) \sim r \times Lon(id, \text{CRT}) \mid \\ Area(id) \sim r \mid Area(id) \sim r \times Area(id) \mid \\ C(id) = c \mid C(id) = C(id) \mid \\ P(id) \sim a \mid P(id) \sim r \times P(id) \end{split}$$

The syntax for the STPL formula is:

$$\begin{split} \phi &:= \mathsf{T} \mid x.\phi \mid \exists id@x.\phi \mid \exists id.\phi \mid id = id \mid \neg \phi \mid \phi \lor \phi \mid \\ \tau - x \sim t \mid \mathcal{F} - x \sim n \mid \mathcal{F} - x \% \ c \sim n \mid \\ \bigcirc \phi \mid \phi \ \mathcal{U} \phi \mid \odot \phi \mid \phi \ \mathcal{S} \phi \mid \\ \Theta \mid \exists \ \mathcal{T} \mid \mathcal{A} \end{split}$$

where \top is the symbol for *true*, $\sim \in \{<, >, \ge, \le, =\}$, and $r \in \mathbb{R}_{\geq 0}$, $c \in \mathbb{N}$, and $a \in [0, 1]$ are constants. Here, $\sigma : V_o \to \Pi$ is a function that maps object variables into spatial propositions.

The syntax of STPL is substantially different from the syntax of MTL×S4_u. In the STPL syntax, $x.\phi$ stands for the freeze time quantifier. When this expression is evaluated, the corresponding frame *i* is stored in the clock variable *x*. The prefix $\exists id$ in the rule $\exists id@x.\varphi$ or the rule $\exists id.\varphi$ is the *Existential* object quantifier. When the formula $\exists id@x.\varphi(id)$ is evaluated, then it is satisfied (true) when there is an object *id* at frame/time *x* that makes $\varphi(id)$ true. The formula $\exists id.\varphi(id)$ is also searching for an object that makes $\varphi(id)$ true, but in this case we do not need to refer to the time that the object was selected. Similarly, the *Universal* object quantifier is defined as $\forall id@x.\phi \equiv \neg(\exists id@x.\neg\phi)$ or $\forall id.\phi \equiv \neg(\exists id.\neg\phi)$. The universal quantifier requires that all the objects in a frame satisfy the subformula φ . Notice that the syntax of STPL cannot enforce that all uses of time or object variables

are bound, i.e., declared before use. In practice, such errors can be detected after the parse tree of the formula has been constructed through a simple tree traversal.

In contrast to $MTL \times S4_u$, the timing constraints in STPL are not annotating the temporal operators, but rather they are explicitly stated as arithmetic expressions in the formulas. For example, consider the specification "*There must exist a car* now which will have class membership probability greater than 90% within 1.5 sec". With timing constraints annotating the temporal operators, the requirement would be:

$$\exists id. (C(id) = \operatorname{car} \land \Diamond_{[0,1.5]} P(id) > 0.9)$$

Since STPL uses time variables and arithmetic expressions to express timing constraints, the same requirement may be written as:

$$\exists id@x.(C(id) = \operatorname{car} \land \diamondsuit(\tau - x \leq 1.5 \land P(id) > 0.9)).$$

The use of time variables enables the requirements engineer to define more complex timing requirements and to refer to objects at specific instances in time. The time, frame, and object constraints of STPL formulas are in the form of $\tau - x > r$, $\mathcal{F} - x > n$, and id = id, respectively. We denote $\tau - x$ and $\mathcal{F} - x$ to refer to the elapsed time and frames, respectively. Note that we use the same variable to refer to the freeze time and frame, but distinguish their type based on how they are used in the constraints (τ represents the current time, and \mathcal{F} represents the current frame number). The operator % in the expression $\mathcal{F} - x \% c \sim n$ is used to specify periodic constraints. For reasoning over the past time, we added \odot (*previous*) and \mathcal{S} (*since*) operators as duality for the \bigcirc and \mathcal{U} operators, respectively.

For STPL formulas ψ , ϕ , we define $\psi \land \phi \equiv \neg (\neg \psi \lor \neg \phi)$, $\bot \equiv \neg \top$ (False), $\psi \rightarrow \phi \equiv \neg \psi \lor \phi$ (ψ Implies ϕ), $\phi \mathcal{R} \psi \equiv \neg (\neg \phi \mathcal{U} \neg \psi)$ (ϕ releases ψ), $\phi \mathcal{R} \psi \equiv \phi \mathcal{R} (\phi \lor \psi)$ (ϕ nonstrictly releases ψ), $\diamond \psi \equiv \top \mathcal{U} \psi$ (Eventually ψ), $\Box \psi \equiv \neg \diamondsuit \neg \psi$ (Always ψ) using syntactic manipulation.

Remark 4.2: In principle, in STPL, it is easy to add multiple classes and classification probabilities for each object. We would simply need to replace in Def. 4.1 the rules

$$C(id) = c \mid P(id) \sim a \mid P(id) \sim r \times P(id)$$

in Θ with

$$c \in C(id) \mid P(id, c) \sim a \mid P(id, c) \sim a \times P(id, c).$$

where C(id) is now a function which returns a set of classes for the object. In order to write meaningful requirements over multiple classes, we would also need to introduce quantifiers over arbitrary sets. That is, we should be able to write a formula such as "there exists at least one class with probability greater than a": $\exists c \in C(Id) . P(Id, c) > a$. This is within the scope of First Order Temporal Logics (e.g., see Basin et al. (2015) and Havelund et al. (2020)) and we plan to consider such additions in the future.

In general, the monitoring problem for STPL formulas is PSPACE-hard (since STPL subsumes Timed Proposition Temporal Logic (TPTL); see Markey and Raskin (2006)). However, there exists a fragment of STPL which is efficiently monitorable (in polynomial time). Definition 4.3 (Almost Arbitrarily Nested Formula): An Almost Arbitrarily Nested (AAN) formula is an STPL formula in which no time or object variables are used in the scope of another freeze time operator.

For example,

$$\varphi_1 = x_1. \Box \left(\mathcal{F} - x_1 > 2 \land x_2. \diamondsuit (\tau - x_2 < 0.01) \right)$$

$$\varphi_2 = \Box \exists Id_1 @x. \diamondsuit \left(\Box \forall Id_2. (Id_1 = Id_2) \land (\tau - x > 2) \right)$$

are AAN formulas. In formula φ_1 , the time variable x_1 is not used in the scope of the x_2 freeze time operator. In formula φ_2 , there is no nested quantifier/freeze time operators. Therefore, they are both ANN STPL formulas. On the other hand,

$$\varphi_{3} = x_{1} (\Box x_{2} \diamond (\mathcal{F} - x_{1} > 2 \land \tau - x_{2} < 0.01))$$

$$\varphi_{4} = \Box \forall Id_{1}@x_{1} \bigcirc \forall Id_{2}@x_{2} \bigcirc \Box \forall Id_{3}.$$

$$(P(Id_{3}) > P(Id_{1}) \land P(Id_{3}) < P(Id_{2}))$$

are not AAN formulas. That is because in φ_3 , the variable x_1 is used in the scope of the nested quantifier operator " x_2 .". In φ_4 , the Id_1 is used in the scope of the second nested quantifier operator " x_2 .".

The authors in Dokhanchi et al. (2016) presented an efficient monitoring algorithm for TPTL formulas with an arbitrary number of *independent* time variables. Our definition of AAN formulas is adopted from their definition of *encapsulated* TPTL formulas that are TPTL formulas with only independent time variables in them.

Remark 4.4: Our definition of AAN formulas is syntactical. However, there can be syntactically AAN formulas that can be rewritten as semantically equivalent non-AAN formulas. There are other ways to define formulas that are not syntactically ANN but semantically equivalent to AAN formulas. The precise mathematical definitions of these formulas is beyond the scope of this paper. In the following, we will show examples of AAN formulas and some other formulas that are equivalent to AAN formulas. For example,

$$\varphi_5 = \Box x. \Box y. ((\tau - x > 1) \implies \Box(\tau - y > 2))$$

which is not an AAN formula can be rewritten as AAN

$$\varphi_5' = \Box x. \Box ((\tau - x > 1) \implies y. \Box (\tau - y > 2)).$$

On the other hand, although

$$\varphi_6 = \Box \forall id_1@x. \Box \forall id_2@y.$$
$$((C(id_1) = C(id_2)) \implies (P(id_2) > 0.9)$$

is not an AAN formula, our monitoring tool supports it since the clock variables are not used. That is, it can be written as:

$$\varphi_6' = \Box \forall id_1 @x. \Box \forall id_2.$$

$$\left((C(id_1) = C(id_2)) \implies (P(id_2) > 0.9) \right)$$

B. STPL Semantics

Before getting into the semantics of the STPL, we represent spatio-temporal function definitions that are used in production rules \mathcal{A} , and Θ . The definitions of these functions are independent of the semantics of the STPL language, and therefore, we can extend them to increase the expressivity of the language. 1) Spatio-Temporal Functions: We define functions f as follows:

• Class of an object:

 $f_C(id, \mathcal{D}, i, \epsilon, \zeta)$ returns the \mathcal{R} (\mathcal{D} (k), $\epsilon(id)$). Class as the class of the $\epsilon(id)$ object in the k'th frame, where $k \leftarrow \zeta(id)$ if $\zeta(id)$ is specified, and $k \leftarrow i$ otherwise.

Probability of a classified object: *f_P(id, D, i, ε, ζ)* returns the *R(D(k), ε(id))*. Prob as the probability of the ε(id) object in the k'th frame, where k ← ζ(id) if ζ(id) is specified, and k ← i otherwise.

- Distance between two points from two different objects: $f_{Dist}(id_j, id_k, CRT_1, CRT_2, \mathcal{D}, i, \epsilon, \zeta)$ computes and returns the *Euclidean Distance* between the CRT_1 point of the $\epsilon(id_j)$ object and CRT_2 point of the $\epsilon(id_k)$ object in the k_1 'th and k_2 'th frames, respectively; and we have $k_1 \leftarrow \zeta(id_j)$ if $\zeta(id_j)$ is specified, and $k_1 \leftarrow i$ otherwise, and $k_2 \leftarrow \zeta(id_k)$ if $\zeta(id_k)$ is specified, and $k_2 \leftarrow i$ otherwise.
- Lateral distance of a point that belongs to an object in a coordinate system (for image space, see section II-C): *f_{LAT}(id*, CRT, *D*, *i*, *ε*, *ζ*) computes and returns the *Lateral Distance* of the CRT point of the *ε(id)* object in the k'th frame from the *Longitudinal axis*, where k ← ζ(*id*) if ζ(*id*) is specified, and k ← *i* otherwise.
- Longitudinal distance of a point that belongs to an object in a coordinate system (for image space, see section II-C): *f*_{LON}(*id*, CRT, *D*, *i*, *ε*, *ζ*) computes and returns the *Longitudinal Distance* of the CRT point of the *ε*(*id*) object in the *k*'th frame from the *Lateral axis*, where *k* ← *ζ*(*id*) if *ζ*(*id*) is specified, and *k* ← *i* otherwise.
- Area of a region:

 $f_{Area}(\mathcal{T})$ computes and returns the area of an spatial term \mathcal{T} if \mathcal{T} is specified, otherwise it returns unspecified.

In the above functions, we use identifier variables to refer to objects in a data stream. Some parameters are point specifiers to choose a single point from all the points that belong to an object. The only function with a spatial term as an argument is the *area* function that calculates the area of a 2D bounding box. Similar reasoning for a 3D geometric shape (polyhedra in general) using a volume function is possible. This is supported by our open-source software STPL (2023), but we don't provide formal definitions here for the brevity of the presentation.

All the functions have \mathcal{D} , i, ϵ and ζ as arguments. Functions need the data stream \mathcal{D} to access and retrieve perception data, the frame number i to access a specific frame in time ("current" frame), a data structure (typically referred to as "environment") ϵ to store and retrieve the values assigned to variables (both for time and objects), and, finally, a data structure ζ to acquire the time/frame at which an object was selected through quantification. The data structure ζ is used to distinguish whether the time that an object was selected is needed or not (see semantics for \exists in Section IV-B2). For example, if we would like to store in the object variable Id_1 , the value k as the identifier of an object in a frame, and store in the frame variable x, the value i as the frozen frame, then we would write

$$\epsilon[Id_1 \leftarrow k]$$
 and $\epsilon[x \leftarrow i]$

respectively. Similarly, we write

$$\zeta[Id_1 \leftarrow i]$$

to denote that we store the frame number *i* when the object variable Id_1 was set. The initial data structures (environments) ϵ_0 and ζ_0 are empty. That is, no information is stored about any object identifier of the clock variable. The environment ζ is needed for the comparison of objects, probabilities, etc over different points in time, and environment ϵ is needed for the comparison of time/frame constraints and quantification of the object variables.

Note that the above functions are application dependent, and we can add to this list based on future needs. In the next section, when we introduce an example of 3D space reasoning, we present a new spatio-temporal function.

Definition 4.5 (Semantics of STPL): Consider $\mathbf{M} = \langle \mathbf{T}, \mathbf{V}, \mathcal{D}, \epsilon, \zeta, \tau \rangle$ as a topological temporal model in which \mathbf{T} is a topological space, \mathbf{V} is a spatial valuation function, \mathcal{D} is a data-object stream, $\zeta : V_o \to \mathbb{N} \cup \{\bot\}$ is an object evaluating function, and $\epsilon : V_t \cup V_o \to \mathbb{N}$ is a frame-evaluating function. Here, $i \in \mathbb{N}$ is the index of current frame, $\tau : \mathbb{N} \to \mathbb{R}^+$ is a mapping function from frame numbers to their physical times, $\phi, \phi_1, \phi_2 \in STPL$ (i.e., formulas belong to the language of the grammar of STPL), V_t is a set of time variables, and V_o is a set of object variables. The quality value of formula ϕ with respect to \mathcal{D} at frame i with evaluations ϵ and ζ is recursively assigned as follows:

2) Semantics of Temporal Operators:

$$\begin{split} \llbracket \tau \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \tau \\ \llbracket x.\phi \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \llbracket \phi \rrbracket (\mathcal{D}, i, \epsilon [x \Leftarrow i], \zeta, \tau) \\ \llbracket \exists id@x.\phi \rrbracket (\mathcal{D}, i) (\mathbb{I}, \epsilon, \zeta, \tau) &\coloneqq \\ \bigvee_{k \in \mathcal{OI}(\mathcal{D}(i))} \left(\llbracket \phi \rrbracket (\mathcal{D}, i, \epsilon [id \Leftarrow k, x \Leftarrow i], \zeta [id \Leftarrow i]) \right) \\ \llbracket \exists id.\phi \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \bigvee_{k \in \mathcal{OI}(\mathcal{D}(i))} \left(\llbracket \phi \rrbracket (\mathcal{D}, i, \epsilon [id \Leftarrow k], \zeta [id \Leftarrow \bot]) \right) \\ \llbracket \tau - x \sim n \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \begin{bmatrix} \tau & \text{if } \tau(i) - \tau(x) \sim n \\ \bot & \text{otherwise} \end{bmatrix} \\ \llbracket \mathcal{F} - x \sim n \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \begin{bmatrix} \tau & \text{if } i - x \sim n \\ \bot & \text{otherwise} \end{bmatrix} \\ \llbracket \mathcal{F} - x \% c \sim n \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \begin{bmatrix} \tau & \text{if } (i - x) \% c \sim n \\ \bot & \text{otherwise} \end{bmatrix} \\ \llbracket id_j = id_k \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \\ \llbracket \tau & \text{if } \epsilon(id_j) = \epsilon(id_k) \\ \bot & \text{otherwise} \end{bmatrix} \\ \llbracket -\phi \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \lor \phi_2 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \llbracket \phi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\vdash \\ \rrbracket (\varphi_1 \rrbracket (\varphi_1$$

$$\begin{split} \llbracket \phi_1 \ \mathcal{U} \ \phi_2 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) \coloneqq \\ & \bigvee_{i \leq j} \left(\llbracket \phi_2 \rrbracket (\mathcal{D}, j, \epsilon, \zeta, \tau) \land \bigwedge_{i \leq k < j} \llbracket \phi_1 \rrbracket (\mathcal{D}, k, \epsilon, \zeta, \tau) \right) \\ \llbracket \bigcirc \phi \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) \coloneqq \llbracket \phi \rrbracket (\mathcal{D}, i + 1, \epsilon, \zeta, \tau) \end{aligned}$$

3) Semantics of Past-Time Operators: Many applications require requirements that refer to past time. Past time operators are particularly relevant to online monitoring algorithms.

$$\begin{split} \llbracket \phi_1 \ \mathcal{S} \ \phi_2 \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) \coloneqq \\ & \bigvee_{i \ge j} \left(\llbracket \phi_2 \rrbracket (\mathcal{D}, j, \epsilon, \zeta, \tau) \land \bigwedge_{j < k \le i} \llbracket \phi_1 \rrbracket (\mathcal{D}, k, \epsilon, \zeta, \tau) \right) \\ \llbracket \odot \phi \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) \coloneqq \llbracket \phi \rrbracket (\mathcal{D}, i - 1, \epsilon, \zeta, \tau) \end{split}$$

4) Semantics of Spatio-Temporal Operators: Now we define the operators and functions that are needed for capturing the requirements that reason over different properties of data objects in a data stream. Note that in some of the following definitions, we used V to avoid rewriting the semantics we built up upon the syntax and semantics as part of $MTL \times S4_u$ logic. The semantics of STPL will be based on the valuation function V of $MTL \times S4_u$ with the change that V will now also accept the environments ϵ and ζ , and we extend the definition of V over the spatial term $\sigma(id)$. That is, we replace in the semantics of the $MTL \times S4_u$ the valuation of spatial propositions p with:

$$\mathbf{V}(\sigma(id), \mathcal{D}, i, \tau, \epsilon, \zeta) = \mathfrak{U}(\epsilon(id), k),$$

where $k \leftarrow \zeta(id)$ if $\zeta(id)$ is specified, and $k \leftarrow i$ otherwise. The semantics for the rest of the spatial operators are:

$$\begin{split} \llbracket [\boxdot \mathcal{T}]](\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \mathbf{V}(\exists \mathcal{T}, \mathcal{D}, i, \tau, \epsilon, \zeta) \\ \llbracket Area(\mathcal{T}) \sim r \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \left\{ \begin{array}{l} \bot \text{ if } f_{Area} \big(\mathbf{V}(\mathcal{T}, \mathcal{D}, i, \tau, \epsilon, \zeta) \big) \text{ is unspecified} \\ \top \text{ if } f_{Area} \big(\mathbf{V}(\mathcal{T}, \mathcal{D}, i, \tau, \epsilon, \zeta) \big) \sim r \\ \bot \text{ otherwise} \\ \llbracket C(id) &= r \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \left\{ \begin{array}{l} \top \text{ if } f_C(id, \mathcal{D}, i, \epsilon, \zeta) \big) &= r \\ \bot \text{ otherwise}^* \\ \llbracket P(id) \sim r \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \left\{ \begin{array}{l} \top \text{ if } f_P(id, \mathcal{D}, i, \epsilon, \zeta) \big) \sim r \\ \bot \text{ otherwise}^* \\ \llbracket Dist(id_j, \operatorname{CRT}_1, id_k, \operatorname{CRT}_2) \sim r \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \left\{ \begin{array}{l} \top \text{ if } f_{Dist}(id_j, id_k, \operatorname{CRT}_1, \operatorname{CRT}_2, \mathcal{D}, i, \epsilon, \zeta) \big) \sim r \\ \bot \text{ otherwise}^* \\ \llbracket LAT(id, \operatorname{CRT}) \sim r \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \left\{ \begin{array}{l} \top \text{ if } f_{LAT}(id, \operatorname{CRT}, \mathcal{D}, i, \epsilon, \zeta) \big) \sim r \\ \bot \text{ otherwise}^* \\ \llbracket LON(id, \operatorname{CRT}) \sim r \rrbracket (\mathcal{D}, i, \epsilon, \zeta, \tau) &\coloneqq \\ \left\{ \begin{array}{l} \top \text{ if } f_{LON}(id, \operatorname{CRT}, \mathcal{D}, i, \epsilon, \zeta) \right) \sim r \\ \bot \text{ otherwise}^* \end{array} \right\} \end{split}$$

where $\sim \in \{>, <, =, \geq, \leq\}$, and "otherwise*" has a higher priority to become true in the *if statements* if $\epsilon(id) \notin OI(D(i))$.

Note that we omit the semantics for some of the spatiotemporal operators in the production rule Θ except the ones stated above due to their similarity to the semantics of the presented operators.

Definition 4.6 (STPL Satisfiability): We say that the data stream \mathcal{D} satisfies the STPL formula φ under the current environment iff $[[\varphi]](\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) = \top$, which is equivalent to denote $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \varphi$.

Note that by ϵ_0 and ζ_0 , we reset all the variables to be empty. This enables the use of the presented proof system of TPTL by Dokhanchi et al. (2016).

Example 4.1 (Evaluating a simple spatio-temporal STPL formula): A sample data stream of three frames time stamped at times t_0 , t_1 , and t_2 is illustrated in Figure 4. We labeled rectangular geometric shapes in each frame by $\sigma(1)$ and $\sigma(2)$, and will refer to them by p_1 and p_2 , respectively. Object p_1 does not change its geometric shape in all the frames, but p_2 evolves in each frame. The location of the p_1 is the same in the first two frames, but in the third frame, it moves to the bottom-right corner of the frame. The location of the p_2 changes constantly in each frame, that is, it first horizontally expands and moves to the bottom-left of the frame, and then expands horizontally and vertically and moves to the right-center of the frame. We are going to compute and evaluate the formula

$$\phi \coloneqq \exists id_1 . \exists id_2 . \exists \left(\sigma(id_1) \ \mathcal{U}^s_{[0,2]} \ \sigma(id_2) \right)$$

according to the semantics in Def. 2.6 and Def. 4.5. An English statement equivalent to the above formula is:

"There exist two objects $\sigma(id_1)$ and $\sigma(id_2)$ in the first frame such that, $\sigma(id_2)$ is non-empty in the first frame; or, for the next two frames, its intersection with $\sigma(id_1)$ in the previous frames is non-empty (given that $\sigma(id_1)$ does not change)".

In Figure 5, we demonstrate the result of evaluating the subformula $\sigma(id_1) \ U^s_{[0,2]} \ \sigma(id_2)$ for i = 1 to 3, where we assign to $id_1 \leftarrow 1$ and $id_2 \leftarrow 2$ from the four possible assignments. We used p_1 and p_2 to refer to $\sigma(id_1 \leftarrow 1)$ and $\sigma(id_2 \leftarrow 2)$ in each frame, respectively. That is, p_1 at t = 2 is equivalent to $\mathfrak{U}(\epsilon(1), 2)$. The evaluation of STPL formula ϕ starts with the following equation

$$\begin{bmatrix} \exists \sigma(id_1) \ \mathcal{U}^s_{[0,2]} \ \sigma(id_2) \end{bmatrix} (\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \coloneqq \\ \mathbf{V} (\exists p_1 \ \mathcal{U}^s_{[0,2]} \ p_2, \mathcal{D}, 0, \tau, \epsilon, \zeta)$$

where, $\epsilon_0[id_1 \leftarrow 1, id_2 \leftarrow 2]$ and $\zeta_0[id_1 \leftarrow \bot, id_2 \leftarrow \bot]$.

The evaluating of the until subformula is computed as below

$$\mathbf{V}(p_1 \ \mathcal{U}^s_{[0,2]} \ p_2, \mathcal{D}, 0, \tau, \epsilon, \zeta) \coloneqq \bigcup_{t' \in \{0,1,2\}} \left(\mathbf{V}(p_2, \mathcal{D}, t', \tau, \epsilon, \zeta) \cap \bigcap_{0 \leq t'' < t'} \mathbf{V}(p_1, \mathcal{D}, t'', \tau, \epsilon, \zeta) \right),$$

where $\tau(0) = 0, \tau(1) = 1$, and $\tau(2) = 2$. The above formula is equal to the below disjunctive normal formula as shown in Fig. 5

$$\mathbf{V}(p_2, \mathcal{D}, 0, \tau, \epsilon, \zeta)$$

 \cup

$$(\mathbf{V}(p_2, \mathcal{D}, 1, \tau, \epsilon, \zeta) \cap \mathbf{V}(p_1, \mathcal{D}, 0, \tau, \epsilon, \zeta)) \cup (\mathbf{V}(p_2, \mathcal{D}, 2, \tau, \epsilon, \zeta) \cap \mathbf{V}(p_1, \mathcal{D}, 0, \tau, \epsilon, \zeta) \cap \mathbf{V}(p_1, \mathcal{D}, 1, \tau, \epsilon, \zeta)).$$

The final evaluation of the above formula is depicted as hashed regions s_0 , s_1 and s_2 in Fig. 6. The union of the regions is a non-empty set, hence, the spatial existential operator returns *true*.

C. MTL/STL Equivalences in STPL

Below, we represent some rewriting rules to translate time interval based formulas in MTL/STL to their equivalent STPL formulas.

$$\phi_1 \ \mathcal{U}_{[a,b]} \ \phi_2 \equiv x.\phi_1 \ \mathcal{U}((a \le \tau - x \le b) \land \phi_2)$$

$$\phi_1 \ \mathcal{R}_{[a,b]} \ \phi_2 \equiv x.\phi_1 \ \mathcal{R}((a \le \tau - x \le b) \implies \phi_2)$$

$$\diamond_{[a,b]} \ \phi \equiv x. \top \ \mathcal{U}((a \le \tau - x \le b) \land \phi)$$

$$\diamond_{[a,b]} \ \phi \equiv x. \Rightarrow ((a \le \tau - x \le b) \land \phi)$$

$$\Box_{[a,b]} \ \phi \equiv x. \bot \ \mathcal{R}((a \le \tau - x \le b) \implies \phi)$$

$$\Box_{[a,b]} \ \phi \equiv x. \Box \ ((a \le \tau - x \le b) \implies \phi)$$

In the next section, we use specific examples of perception systems to explain each aspect of the logic individually gradually and later combined.

V. CASE STUDY

In this section, we are going to gradually demonstrate the expressivity of the STPL language using a sample perception data stream corresponding to the image frames in Figure 1. There are six frames in this case-study that are taken from the KITTI² databases (Geiger et al. (2013a)). We adopt a perception system based on SqueezeDet by which some desirable classes of objects in image frames are recognized. The perception classifies an object as one class from three desirable classes *Car*, *Cyclist*, and *Pedestrian*. It also assigns to each detected object a confidence level (normalized between 0 and 1), and a bounding box that surrounds the object. We also manually associate a unique object identifier to the detected objects of each frame. The data is available to the object data stream function \mathcal{D} as represented in Table II, and visualized in the frames in Fig. 1.

In the following sections, we focus on STPL specifications that use quantifiers, and spatial and temporal operators. We present the formalization of 15 requirements in total. A highlight is the STPL formula (16) that formalizes Req. 15 and its utilization on the KITTI dataset in our monitoring tool. Our monitoring tool was able to detect some mismatches in the labeled data. All of the STPL formulas and their monitoring result except the last two are distributed with our monitoring framework (STPL (2023)).

Note that in the rest of the sections, when we translate a requirement into STPL, we will refer to the following assumptions. These assumptions enable us to write smaller

²http://www.cvlibs.net/datasets/kitti/



Fig. 4: An object-data stream \mathcal{D} of three frames illustrated in three time steps t_0, t_1 , and t_2 . In each frame, there are two rectangular geometric shapes denoted by identifiers 1 and 2. The lighter blue colored rectangle and the darker blue colored rectangle are identified by p_1 and p_2 , respectively.



Fig. 5: The step-by-step computation of $\mathbf{V}(p_1 \mathcal{U}_{[0,2]}^s p_2, \mathcal{D}, 0, \tau, \epsilon)$ for i = 0 to 2. Here, p_1 and p_2 are the same as in Fig. 4.



Fig. 6: $s_2 = \mathbf{V}(p_2, \mathcal{D}, 0, \tau, \epsilon), \ s_0 = \mathbf{V}(p_2, \mathcal{D}, 1, \tau, \epsilon) \cap \mathbf{V}(p_1, \mathcal{D}, 0, \tau, \epsilon), \ s_1 = \mathbf{V}(p_2, \mathcal{D}, 2, \tau, \epsilon) \cap \mathbf{V}(p_1, \mathcal{D}, 0, \tau, \epsilon) \cap \mathbf{V}(p_1, \mathcal{D}, 1, \tau, \epsilon), \ \mathbf{V}(p_1 \mathcal{U}_{[0,2]}^s p_2, \mathcal{D}, 0, \tau, \epsilon) = s_0 \cup s_1 \cup s_2.$

formulas. They can be relaxed at the expense of more complex requirements.

Assumption 5.1: There are different perception modules to detect, track and classify the objects. That is, the object detector detects objects, and the object tracker assigns unique identifiers to the detected objects, and the classifier assigns classes to the detected objects.

Assumption 5.2: Object detector always detects objects.

Assumption 5.3: Each detected and tracked object has a unique id.

The tracker can check the detected objects through the sequence of frames but it does not imply that the classifier will assign the same class to them in all the frames. If we cannot assume the existence of unique IDs on the same object, then we can still write requirements in a more complicated form to achieve more conservative results. For example, in the requirements that follow (Req 3-18), whenever we check for the presence of the same ID, e.g.,

$$\psi_{=} = \Box \forall Id_{i}. \Leftrightarrow \exists Id_{j}. (Id_{i} = Id_{j} \implies \varphi),$$

then this precondition can be replaced with one where we quantify over all objects that intersect with the original object in the previous frame, e.g.,

$$\psi_{\sqcap} = \square \forall Id_i@x. \Leftrightarrow \exists Id_j.$$

$$\left(\left(\exists (\sigma(Id_i) \sqcap \sigma(Id_j)) \land C(Id_i) = C(Id_j) \right) \Longrightarrow \varphi \right).$$

Notice the increase in complexity between the two formulas. In $\psi_{=}$, we just need to store the ID of the object in Id_i and just check if in the future there is another object (Id_j) with the same ID. In ψ_{\sqcap} , we need to store the time x that we observed the object Id_i , so that we can retrieve its bounded box and check for intersection with future objects along with checking the agreement of classes. Clearly, $\psi_{=}$ and ψ_{\sqcap} are not syntactically or semantically equivalent. Nevertheless, under the assumption of sufficient sampling rate, then the objects that satisfy $\psi_{=}$ will also satisfy ψ_{\sqcap} . There exist other possible formulas under which we can relax the requirement for persistent unique IDs. However, in the following presentation, we will always be using the assumptions that result in the simplest STPL formulas.

For the requirements that are formalized into STPL in the rest of this section, we used our STPL monitoring tool to apply the data stream in Table II to each STPL formula, and presented the result in Table I.

TABLE I: The result of monitoring STPL formula φ on the data stream \mathcal{D} in Table II. $[[\varphi]]$ abbreviates $[[\varphi]](\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau)$. * We used the dataset "0018" from KITTI tracking benchmark for evaluating formula in Eq.(16).

φ	$\llbracket \varphi \rrbracket$	φ	$\llbracket \varphi \rrbracket$	φ	$\llbracket \varphi \rrbracket$	φ	$\llbracket \varphi \rrbracket$
Eq.(1)	Т	Eq.(2)	T	Eq.(3)	T	Eq.(4)	T
Eq.(5)	Т	Eq.(9)	Т	Eq.(10)	Т	Eq.(11)	T
Eq.(12)	Т	Eq.(13)	T	Eq.(14)	T	Eq.(15)	T
Eq.(16)	Т	*Eq.(16)	T	Eq.(17)	Т	Eq.(18)	Т

A. Object Quantifier Examples

First, we present a requirement that enables search through a data stream to find a frame in which there are at least two unique objects from the same class.

Req. 3: There is at least one frame in which at least two unique objects are from the same class. **STPL** (using all the assumptions):

$$\phi_3 = \Diamond \exists Id_1 . \exists Id_2 . (Id_1 \neq Id_2 \land C(Id_1) = C(Id_2)) \tag{1}$$

In the above formula, the object variable constraint requires a unique assignment of object identifiers to the variables Id_1 and Id_2 . Additionally, the equivalence proposition is valid only if the classes of a unique pair of objects are equal. Also, the existential quantifier requires one assignment of objects to the object variables that satisfy its following subformula. Lastly, the eventually operator requires one frame in which its following subformula is satisfiable.

Checking the Formula on Real Data:: From the data stream \mathcal{D} in Table II, and as depicted in the frames in Fig. 1, the frame numbers 0, 2 and 3 have two pedestrians, and frame number 3 has two cars in them. Therefore, the formula is satisfiable for the given object data stream and we can denote it as $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \phi_3$. Note that we can push the \diamond operator after the existential quantifier (i.e., $\diamond \exists Id_1. \exists Id_2. \phi \equiv \exists Id_1. \exists Id_2. \diamond \phi$) and still expect the same result.

B. Examples with Time Constraints

Next, we formalize a requirement in which we should use nested quantifiers and time/frame constraints in our formalization.

Req. 4: Always, for all objects in each frame, their class probability must not decrease with a factor greater than 0.1 in the next two seconds, unless Frame Per Second (FPS) drops below 10 during the first second.

STPL (using Assumptions 5.1-5.3):

$$\phi_4 = \Box \forall I d_1 @x. \left(\psi_{41}^{Id_1, x} \implies \psi_{42}^x\right) \tag{2}$$

where

$$\psi_{41}^{Id_1,x} \equiv \Diamond \exists Id_2. (Id_1 = Id_2 \land \tau - x \leq 2 \land P(Id_2) < 0.9 \times P(Id_1))$$
$$\psi_{42}^x \equiv \bigcirc \Diamond (Ratio(\mathcal{F} - x, \tau - x) < 10 \land \tau - x \leq 1)$$

Notice that the function "Ratio($\mathcal{F}-x, \tau-x$)" is not directly allowed by the syntax of STPL (Def. 4.1). However, the arithmetic expression uses a single freeze time variable x and, hence, its evaluation is no more computationally expensive than of the individual expressions " $(\mathcal{F} - x)$ " and " $(\tau - x)$ ". We remark that the function Ratio introduces the possibility of division by zero. In the formula ψ_{42}^x , the issue is avoided in the specification by using the next time operator (\bigcirc) . However, an implementation of the function Ratio should handle the possibility of division by zero. The subformula $\psi_{41}^{Id_1,x}$ searches for a vehicle for which within 2 sec in the future, i.e., $(\tau - x \le 2)$, its classification probability drops below the desired threshold. Notice that with the expression $P(Id_2) < 0.9 \times P(Id_1)$, we compare probabilities for the same object across different points in time. The subformula ψ_{42}^x checks whether within 1 sec in the future the FPS drops below 10. The implication $\psi_{41}^{Id_{1},x} \implies \psi_{42}^{x}$ enforces that if there is a vehicle for which the classification probability drops, then at the same time the FPS should be low.

Checking the Formula on Real Data:: Except for the object with ID = 2 in the first frame in Table II, the rest of the objects satisfy the requirement. Therefore, the whole requirement is not satisfiable, that is $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \neq \phi$. Formula (2) is too strict. That is, it checks the maximum allowed drop in probabilities for a classified object in all the frames rather than for the newly classified objects. For example, assume a sample data stream of size three with only one object classified as a car with the probabilities 0.7, 0.9, 0.64 during the first, second, and third frames, respectively. Formula (2) without the always operator is not satisfiable starting from the second frame (more than 28% probability decrement), but satisfiable for the first and the last frame.

We can relax Formula (2) by adding a precondition to the antecedent of the implication to only consider the newly detected objects. The relaxed requirement is possible by using the *weak previous operator* \odot_w . The weak previous operator does not become unsatisfiable in the first frame on a data stream where there is no previous frame. Similarly, by \bigcirc_w , we denote the *weak next operator*. It is similar to the next operator, but it does not become unsatisfiable in the last frame on a finite data stream. For more information about weak temporal operators and past LTL see the works by Eisner and Fisman (2006) and Cimatti et al. (2004), respectively.

The following formula is a revised version of Formula (2): **STPL** (using Assumptions 5.1-5.3):

$$\phi_4' = \Box \forall Id_1 @x. \left(\psi_{40}^{Id_1} \Rightarrow \left(\psi_{41}^{Id_1, x} \Rightarrow \psi_{42}^x\right)\right) \tag{3}$$

where

$$\psi_{40}^{Id_1} \equiv \bigcirc_w \forall Id_3. (Id_1 \neq Id_3)$$

In formula ϕ'_4 , the antecedent subformula $\psi^{Id_1}_{40}$ checks if an object did not exist in the previous frame. Therefore, the consequent $\psi^{Id_1,x}_{41} \implies \psi^x_{42}$ is only checked for new objects that appear in the frame for the first time.

Req. 5: Always, each object in a frame must exist in the next 2 frames during 1 second.

STPL (using Assumption 5.2-5.3):

$$\phi_{5} = \Box \forall Id_{1}@x. \left(\left(\odot_{w} \forall Id_{3}. (Id_{1} \neq Id_{3}) \right) \Rightarrow \\ \Box \left(\left(\tau - x \leq 1 \land \mathcal{F} - x \leq 2 \right) \Rightarrow \qquad (4) \\ \exists Id_{2}. \left(Id_{1} = Id_{2} \right) \right) \right)$$

Similar to the previous example, the equalities that need to hold are in the consequent of the second implication, but its antecedent is a conjunction of time and frame constraints. In this formula, there is a freeze time variable after the first quantifier operator, which is followed by an always operator. Thus, the constraints apply to the elapsed time and frames between the freeze time operator and the second always operator. Therefore, for any three consecutive frames, if the last two frames are within 1 second of the first frame, then all the objects in the first frame have to reappear in the second and third frames.

Running Formula in Real Data Stream:: The same result as in the previous example holds here for the data stream \mathcal{D} in Table II that is $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \notin \phi_5$.

Below, we translate the requirement previously presented as in Req. 1.

Req. 6: Whenever a new object is detected, then it is assigned a class within 1 sec, after which the object does not change class until it disappears.

STPL (using Assumptions 5.1-5.3):

$$\phi_{6} = \Box \forall Id_{1}@x.$$

$$\left(\left(C(Id_{1}) = 0 \Rightarrow \psi_{61}^{Id_{1},x} \right) \land \\ \left(C(Id_{1}) > 0 \Rightarrow \psi_{62}^{Id_{1},x} \right) \right)$$
(5)

where the subformulas are defined as:

$$\psi_{61}^{Id1,x} \equiv \diamondsuit \left(\left(\tau - x \le 1 \land \mathcal{F} - x \ge 1 \right) \land \\ \Box \exists Id_2 . \left(Id_1 = Id_2 \land C(Id_2) > 0 \right) \right) \\ \psi_{62}^{Id1,x} \equiv \Box \forall Id_3 \left(\left(\mathcal{F} - x \ge 1 \land Id_3 = Id_1 \right) \Rightarrow \\ C(Id_1) = C(Id_3) \right) \end{cases}$$

In the above formula, we evaluate the two implication-form subformulas for any objects in all the frames. If there is an object that is not classified (i.e., $C(Id_1) = 0$), then we check the consequence of the corresponding subformula. The subformula corresponding to the subformula $\psi_{61}^{Id_{1,x}}$ requires that when an unclassified object was observed, then eventually in less than a second afterward, the object always has a class being assigned to it. If there is an object that is already classified, then the second predicate has to be evaluated. The subformula equivalent to the $\psi_{62}^{Id_{1,x}}$ requires that when a classified object was observed, then afterward the same object only can take the same class.

Running Formula in Real Data Stream:: In the frame 1 in Table II, the object with ID = 2 changes its class from *cyclist* to *pedestrian*. Therefore, the data stream \mathcal{D} does not satisfy the requirement $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \neq \phi_6$. TABLE II: Data stream \mathcal{D} of image frames in Fig. 1. Each row for the column headers: Frame, τ , ID (object identifiers), class, and Prob represent the frame number, the sampling time (e.g., here we assume that frame-per-second is 25 fps), the associated identifier to the objects, the classification confidence, respectively. Moreover, the other four headers are the data attributes (minimum and maximum lateral and longitudinal positions of coordinates of the bounding hoves) by which a

attributes (minimum and maximum lateral and longitudinal positions of coordinates of the bounding boxes) by which a bounding box is associated with each classified object. Note that some objects are not tracked correctly throughout the data stream.

Frame	au	ID	Class	Prob	x_{min}	y_{min}	x_{max}	y_{max}
0	0	1	car	0.88	58	151	220	287
0	0	2	cyclist	0.75	479	124	690	382
0	0	3	pedestrian	0.63	522	130	632	377
0	0	4	pedestrian	0.64	861	133	954	329
1	0.04	1	car	0.88	61	152	217	283
1	0.04	2	cyclist	0.57	493	111	699	383
1	0.04	3	pedestrian	0.64	877	136	972	330
2	0.08	1	car	0.89	58	143	220	271
2	0.08	2	pedestrian	0.65	511	107	724	367
2	0.08	3	pedestrian	0.64	911	115	1001	340
3	0.12	1	car	0.92	56	139	216	266
3	0.12	2	cyclist	0.59	493	111	705	380
3	0.12	3	pedestrian	0.72	541	125	649	351
3	0.12	4	car	0.58	926	107	1004	302
3	0.12	5	pedestrian	0.76	938	118	998	332
4	0.16	1	car	0.91	53	139	217	265
4	0.16	2	pedestrian	0.80	551	126	658	356
5	0.2	1	car	0.92	52	140	216	264
5	0.2	2	cyclist	0.62	506	104	695	368
5	0.2	3	pedestrian	0.68	552	115	669	362

C. Examples of Space and Time Requirements with 2-D Images

In the following examples, we want to demonstrate the expressivity and usability of the STPL language to capture requirements related to the detected objects and their spatial and temporal properties.

1) Basic Spatial Examples without Quantifiers.: Let us assume that we have a spatial proposition p as in Fig. 3.

Example 5.1: One way to formalize that the sampling rate is sufficient is to check if the bounding box of an object overlaps with itself across consecutive frames. The following requirement can be used as a template in more complex requirements.

Req. 7: The intersection of a spatial predicate p with itself across all frames must not be empty. **STPL:**

$$\phi_7 = \exists \ \Box^s p \tag{6}$$

The above requirement is clearly too strict. In order to make the requirement more realistic, we introduce frame intervals (see Rem. 2.8) and also use quantification over objects. If we change the above requirement to: "All the objects in all the frames must intersect with themselves in the next three frames", then we can modify Eq. (6) to the following:

$$\phi_7' = \Box \forall Id_1. \exists \tilde{\Box}_{[0,3]}^s \sigma(Id_1) \tag{7}$$

Example 5.2: It is interesting to check the occupancy of an object across a series of frames. This can be used for identifying regions of interest.

Req. 8: *The union of a spatial proposition p with itself in all the frames is not equal to the universe.* **STPL:**

$$\phi_{\mathbf{8}} = \neg \forall \diamond^{s} p \tag{8}$$

The result of applying the formula $\diamondsuit^s p$ on a sequence of three frames is shown in Fig. 3. The resulting set is not equal to the universe; hence; the formula ϕ_8 evaluates to *true*.

2) Basic Spatial Examples with Quantifiers.: In the following examples, the spatial requirements cannot be formalized without the use of quantifiers.

Example 5.3: We can check if some properties hold for all the objects across all the frames.

Req. 9: Always, all the objects must remain in the bounding box (x_1, y_1, x_2, y_2) .

STPL (using Assumption 5.2):

$$\phi_{9} = \Box \forall Id. (LAT(Id, LM) \ge x_{1} \land LAT(Id, RM) \le x_{2} \land LON(Id, TM) \ge y_{1} \land LON(Id, BM) \le y_{2})$$
(9)

Running Formula in Real Data Stream:: The result of evaluating the above formula depends on the size and position of the bounding box. For example, if the bounding box is the same as the image frame in the data stream \mathcal{D} in Table II, then we have $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \phi_9$.

Example 5.4: The combination of eventually/globally operators and a quantifier operator is very useful for specifying properties of an object across time.

Req. 10: Eventually, there must exist an object that at the next frame shifts to the right.

STPL (using Assumptions 5.2-5.3):

$$\phi_{10} = \Diamond \exists Id_1 @x. \bigcirc \exists Id_2. ($$
$$Id_1 = Id_2 \land LAT(Id_1, LM) < LAT(Id_2, LM)) \quad (10)$$

Note that in the above formula, the Id_1 is quantified in a freeze time quantifier to get access to the data-object values in the frozen time. If we change the first existential quantifier to the universal quantifier, then the formula represents the following requirement: "Eventually, there must exist a frame in which all the objects shift to the right in the next frame".

Running Formula in Real Data Stream: The data stream \mathcal{D} as in Table II satisfies the above formula $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \phi_{10}$. For example, the pedestrian with ID = 3 moved to the right in the frame 1. Note that the pedestrian and the recording camera both moved to the left, while the camera's movement was faster.

Example 5.5: The requirement below checks the robustness of the tracking algorithms in a perception system using time and spacial constraints.

Req. 11: If an object disappears from the right of the image in the next frame, then during the next 1 seconds, it can only reappear from the right.

STPL (using Assumptions 5.2-5.3):

$$\phi_{11} = \Box \forall Id_1 @x. \bigcirc_w \forall Id_2 @y. \left(\psi_{111}^{Id_1, Id_2} \Rightarrow \psi_{112}^{Id_1, Id_2, y}\right)$$
(11)

where

$$\begin{split} \psi_{111}^{Id_1,Id_2} &\equiv Id_1 = Id_2 \land \\ & LAT(Id_1, LM) < LAT(Id_2, LM) \land \\ & LAT(Id_2, LM) > \frac{3}{4}W \land \\ & \bigcirc_w \forall Id_3. \ (Id_3 \neq Id_1) \\ \psi_{112}^{Id_1,Id_2,y} &\equiv \bigcirc_w \Box \\ & \left(\left((\tau - y < 1) \land \forall Id_4.(Id_2 \neq Id_4) \land \\ & \bigcirc_w \exists Id_5.(Id_2 = Id_5) \right) \Rightarrow \\ & \bigcirc_w LAT(Id_5, LM) > \frac{3}{4}W \right) \end{split}$$

Note that in the above formula, the Id_1 and Id_2 are quantified in a nested freeze time quantifier structure, and hence, it is not an AAN formula. There, W is a constant denoting the width of the image in pixels. This formalization will check the reappearance of an object even if it happened more than once. In order to force the formula to only check once for the reappearance of an object, one should use the release operator.

Checking the Formula on Real Data:: In Table II, in the third frame, the pedestrian with ID = 3 has disappeared in the fourth frame and reappeared in the last frame. However, the preconditions captured in subformula $\psi_{111}^{Id_1, Id_2}$ for that object do not hold assuming that W = 1100 for the data stream. That is the least x position of the bounding box for the object with ID = 3 is 911 in frame 2, and then it changes to 541 in frame 3 which violates the condition for shifting toward right before disappearing (the LAT condition in the subformula $\psi_{111}^{Id_1, Id_2}$). Note that, from the second frame afterward, the tracking algorithm mixes the ID association to the objects are not consistent. The inconsistent object tracking makes the subformula $\psi_{111}^{Id_1, Id_2}$ not hold for viable cases. Therefore, for the data stream \mathcal{D} in Table II, we have $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \phi_{11}$.

For the examples in the rest of this section, we focus on the real-time requirements that concern detected objects in perception systems.

Example 5.6: The requirement below checks the robustness of the classification of the pedestrians in a perception system using time and spatial constraints.

Req. 12: If a pedestrian is detected with probability higher than 0.8 in the current frame, then for the next 1 second, the probability associated with the pedestrian should not fall below 0.7, and the bounding box associated with the pedestrian in the future should not overlap with another detected bounding box.



(a) The car inside the red rectangle is identified with ID 4 in frame (b) The same car as in Image (a) is bounded by the yellow rectangle, 10, and it is annotated as "partly occluded".

and it is annotated as "largely occluded" in frame 11.



(c) The car inside the red rectangle is identified with ID 5 in frame (d) The same car as in Image (c) is bounded by the yellow rectangle, 14, and it is annotated as "partly occluded". and it is annotated as "largely occluded" in frame 15.



(e) The car inside the red rectangle is identified with ID 16 in frame (f) The same car as in Image (e) is bounded by the yellow rectangle, 260, and it is annotated as "partly occluded". and it is annotated as "largely occluded" in frame 261.

Fig. 7: We used the labels corresponding to the 390 images in the folder "0008" from the KITTI tracking benchmark. Our STPL monitoring tool detected frames 11, 15, and 261 with "largely occluded" labels as inconsistent annotations. The training data and their format are available from the links in the footnote.

STPL (using Assumptions 5.2-5.3):

$$\phi_{12} = \Box \forall Id_1 @x. ((C(Id_1) = Ped \land P(Id_1) > 0.8) \Longrightarrow$$
$$\Box (\tau - x \le 1 \Longrightarrow$$
$$\exists Id_2. (Id_1 = Id_2 \land P(Id_2) > 0.7 \land C(Id_2) = Ped \land$$
$$\forall Id_3. (Id_2 \neq Id_3 \Longrightarrow$$
$$\neg \exists (\sigma(Id_2) \sqcap \sigma(Id_3)))))) (12)$$

Running Formula in Real Data Stream:: In Table II, there is no pedestrian associated with probability higher than 0.8, therefore the data stream \mathcal{D} satisfies the above formula $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \vDash \phi_{12}.$

Example 5.7: The requirement below represents a situation in which all the adversarial cars in the frames from the current time forward are moving at least as fast as the ego car and in the same direction.

Req. 13: Always all the bounding boxes of the cars in the image frames do not expand.

STPL (using Assumptions 5.2-5.3):

$$\phi_{13} = \Box \forall Id_1 @x. (C(Id_1) = Car \Longrightarrow)$$
$$\Box \forall Id_2. ((Id_1 = Id_2 \land C(Id_2) = Car) \Longrightarrow)$$
$$Area(Id_1) \ge Area(Id_2))) \quad (13)$$

Notice that in the above example, the geometric position of the objects in the space is not required.

Running Formula in Real Data Stream:: In Table II, the bounding box of the car with ID = 1 has expanded in the frame 2. Therefore, the data stream \mathcal{D} does not satisfy the above formula $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \neq \phi_{13}$.

To ease the presentation in the formalization of the requirements in the following examples, we introduce some derived spatial operators:

- Subset: $p_1 \subseteq p_2 \equiv \forall (\overline{p_1} \sqcup p_2)$ Set equality: $p_1 = p_2 \equiv p_1 \subseteq p_2 \land p_2 \subseteq p_1$

Example 5.8: We want to formalize a requirement specification that applies restrictions on the positions and movements of the adversarial and the ego car in all the image frames. The following requirement can be thought of as a template to be used within other requirements.

Req. 14: The relative position and velocity of all the cars are fixed with respect to the ego car.

STPL (using Assumptions 5.2-5.3):

$$\phi_{14} = \forall Id_1 @x. \Box \exists Id_2. (Id_1 = Id_2 \land \sigma(Id_1) = \sigma(Id_2))$$

Notice that we quantify over all objects in the first frame (Id_1) , and then we require that for all future times there is an object (Id_2) with the same ID and bounding box. If in addition, we would like to impose the requirement on any

new objects that we observe, then the formalization of the requirement would be:

$$\phi_{14}' = \Box \forall Id_1@x. \Box \exists Id_2.$$

$$\left(Id_1 = Id_2 \land \sigma(Id_1) = \sigma(Id_2)\right)$$
(14)

Another formalization for the last requirements is

$$\phi_{14}^{\prime\prime} = \Box \forall Id_1. \Big(\Box^s \sigma(Id_1) = \diamondsuit^s \sigma(Id_1) \Big)$$
(15)

Running Formula in Real Data Stream: All of the objects in Table II have different bounding boxes in different frames. Thus, the data stream \mathcal{D} does not satisfy the above formulas $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \notin \phi_{14}$.

Example 5.9: In this example, we represent a sample requirement for detecting occluded objects. The example is about a high confidence OBJECT which suddenly disappears without getting close to the borders. The requirement checks if such an object existed before and next disappeared, then it has to be occluded by a previously close proximity object.

Req. 15: If there exists a high confidence OBJECT, and in the next frame, it suddenly disappears without being close to the borders, then it must be occluded by another object. **STPL** (using Assumptions 5.2-5.3):

$$\phi_{15} = \Box \forall Id_1 @x. \left(\left(\varphi_{high}^{prob} \land \varphi_{far}^{borders} \land \bigcirc \varphi_{disap} \right) \Longrightarrow \varphi_{occ} \right) \quad (16)$$

and the predicates are defined as:

$$\begin{split} \varphi_{high}^{prob} &\equiv P(Id_1) > 0.8\\ \varphi_{far}^{borders} &\equiv LON(Id_1, \text{TM}) > d_1 \land LON(Id_1, \text{BM}) < d_2\\ \land LAT(Id_1, \text{LM}) > d_3 \land LAT(Id_1, \text{RM}) < d_4\\ \varphi_{disap} &\equiv \forall Id_2.(Id_1 \neq Id_2)\\ \varphi_{occ} &\equiv \exists Id_3. \exists Id_4. \Big(Id_1 \neq Id_3 \land Id_1 = Id_4 \land \\ &\equiv \Big(\sigma(Id_4) \sqcap \big(\sigma(Id_3) \sqcup \bigcirc^s \sigma(Id_3) \big) \Big) \Big) \end{split}$$

In the above formulas, the subformula $\varphi_{far}^{borders}$ checks if the object identified by Id_1 is close to the borders of the image frame at the current time, where d_1 , d_2 , d_3 , and d_4 are some integers. The subformula $\forall Id_2.(Id_1! = Id_2)$ is to check if an object disappeared. In the subformula φ_{occ} , the Id_4 refers to the object which disappears in the next frame, and the Id_3 refers to the object that occludes the other object. The spatial subformulas check if the bounding box of the two objects intersect each other in the current frame, or one bounding box in the next frame. Similarly, we can formalize the occlusion without using STE operators by rewriting φ_{occ} as below

$$\varphi_{occ} \equiv \exists Id_3. \exists Id_4. (Id_1 \neq Id_3 \land Id_1 = Id_4 \land Dist(Id_4, CT, Id_3, CT) < d_5) \quad (17)$$

Note that the second formalization is less realistic because it puts a threshold on the *Euclidean distance* (i.e., d_5 is a positive real number) between two objects to infer their overlap.

Running Formula in Real Data Stream: All the objects in Table II are close to the borders of the images, and the data stream does not satisfy the subformula $\varphi_{far}^{borders}$. Therefore, the data stream \mathcal{D} satisfies the the above formula $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \phi$.

In the following, we used the above formula to partially validate the correctness of a training data stream that is used for object tracking.

Running Formula in Real Data Stream:: We used the formula in Eq. (16) to verify the correctness of the labeled objects as "*largely occluded*" in the KITTI tracking benchmark. From the 21 datasets, except for 4 of them (i.e., datasets "0013", "0016", "0017", and "0018"), we detected inconsistencies in labeling objects as "*largely occluded*". For example, in the dataset "0008" (to refer to as data stream \mathcal{D}), we identified 3 frames in each a car was labeled as "*largely occluded*", while they were inconsistent with the rest of the occluded labels. That is, we have $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \notin \phi$. Our STPL monitoring tool detected the wrong labels by applying the STPL formula in Eq. (16) on the data stream in less than 2 seconds. The result of this experiment is shown in the Figure 7. We provide all the datasets as part of our monitoring tool.

Below, we translate the requirement previously presented as in Req. 2.

Req. 16: The frames per second of the camera is high enough so that for all detected cars, their bounding boxes self-overlap for at least 3 frames and for at least 10% of the area. We interpret the above requirement to require an overlap check over a batch of three frames rather than three consecutive frames.

STPL (using Assumptions 5.2-5.3):

$$\phi_{16} = \Box \forall Id_1 @x. \left(\left(\bigcirc_w \forall Id_3. (Id_1 \neq Id_3) \right) \Longrightarrow \right)$$
$$\Box \left(\left(\mathcal{F} - x \ge 1 \land \mathcal{F} - x \le 3 \right) \Longrightarrow \right)$$
$$\forall Id_2. (Id_1 = Id_2 \Longrightarrow \\Ratio(Area(\sigma(Id_1) \sqcap \sigma(Id_2)), Area(\sigma(Id_2))) \ge 0.1) \right)$$
(18)

In the above formula, the spatial subformula is in the form of a non-equality *ratio* function. Its first parameter represents an occupied area for the intersection of the bounding boxes of any two objects obj_1 and obj_2 referred to by Id_1 and Id_2 , respectively. The second parameter denotes the area for obj_2 . For the non-equality to be satisfiable, first, there must be a non-empty intersection between the two objects. Second, the ratio of the intersected area to the area of the second object must be at least 10%.

Running Formula in Real Data Stream:: The data stream \mathcal{D} in Table II satisfies the above formula $(\mathcal{D}, 0, \epsilon_0, \zeta_0, \tau) \models \phi_{16}$.

D. Examples of Space and Time Requirements with 3D environment

³Training data: http://www.cvlibs.net/datasets/kitti/eval_tracking.php, and data format: https://github.com/JonathonLuiten/TrackEval/blob/master/docs/ KITTI-format.txt



(e) Colored semantic segmented of the LiDAR data at time t_0 .

(f) Colored semantic segmented of the LiDAR data at time t_1 .

Fig. 8: Two LiDAR and Camera frames annotated data along with the semantic segmentation data are taken from NuScenes LidarSeg dataset (Scene-0247) (Caesar et al. (2020)).



(a) *x-y plane* bird-eye view of (b) *x-y plane* bird-eye view of LiDAR point cloud at time t_0 . LiDAR point cloud at time t_1 .

Fig. 9: The bird-view of LiDAR point clouds for the two consecutive frames in Fig. 8 where the drivable area in front of the ego is shown.

1) Missed classification scenario: Here, we are going to show by example how 3D reasoning is possible using STPL. Specifically, we are going to show by example if the perception system misses some important objects. The six images in Figure 8 illustrates annotated information of two consecutive frames that are taken from nuScenes-lidarseg⁴ dataset (Caesar et al. (2020)). For each detected class of objects, they adopt color coding to represent the object class in the images. The color orange represents cars, and the color cyan represents drivable regions. The color-coded chart of the classes, along with their frequency in the whole dataset can be found online (Motional (2019)). In the first frame, at time t_0 , the LiDAR annotated image is shown in Fig. 8(a). The corresponding camera image for the LiDAR scene is shown in Fig. 8(c). There is a car in this image that we pointed to in a red arrow. As one can check, there is no point cloud data associated with the identified car in the LiDAR image, and as a result, it is not detected/recognized as a car. In the next images illustrated in Fig. 8(b,d), the undetected car at t_0 is identified and annotated as a car at time t_1 . Unlike before, there are some point cloud data associated with the car by which it was detected. The semantic segmentation of the corresponding images is shown in Fig. 8(e-f). First, we will discuss how to decide if there is a missed object in the datasets, similar to what we presented here. Second, we represent a requirement

⁴https://www.nuscenes.org/nuscenes?sceneId=scene-0274&frame=0& view=lidar *Example 5.10:* For this and the next example scenarios, we have 3D coordinates of bounding volumes and 3D point clouds representing each detected object in the frames. However, the height of those points and cubes are not helpful in these cases, and therefore, we mapped them to the *x-y plane* by ignoring the *z-axis* of sensor positions (i.e., see Motional (2019)).

Let assume that the origin of the coordinate system is the center of the ego vehicle, and the *y*-axis is positive for all the points in front of the ego car, and the *x*-axis is positive for all the points to the right of the ego. That is, we have 2D bounding boxes of classified objects that are mapped to a flat ground-level environment as shown in Fig. 9. Therefore, we do not need to define new spatial functions for 3D reasoning.

Req. 17: If there is a car driving toward the ego car in the current frame, it must exist in the previous frame. **STPL** (using Assumptions 5.2-5.3):

$$\phi_{17} = \Box \forall Id_1. \forall Id_3 @x. \left(\left(\varphi_{exist}^{car} \land \varphi_{close}^{dist} \right) \Longrightarrow \\ \odot_w \varphi_{existed}^{before} \right)$$
(19)

and the predicates are defined as:

$$\varphi_{exist}^{car} \equiv \left(C(Id_1) = Car \right) \land \left(C(Id_3) = Drv \right)$$

$$\begin{split} \varphi^{dist}_{close} &\equiv LON(Id_1, \text{BM}) \leq 1.2 \times LON(Id_3, \text{BM}) \land \\ LON(Id_1, \text{TM}) \geq 0.5 \times LON(Id_3, \text{BM}) \land \\ LAT(Id_1, \text{CT}) \geq LAT(Id_3, \text{LM}) \land \\ LAT(Id_1, \text{CT}) \leq LAT(Id_3, \text{RM}) \end{split}$$

$$\varphi_{existed}^{before} \equiv \exists Id_2. (C(Id_2) = Car \land Id_1 = Id_2 \land Dist(Id_2, CT, \mathbb{U}, CT) > Dist(Id_1, CT, \mathbb{U}, CT))$$

Note that in the above 2D spatial functions, we used BM and TM in contrast to their original semantics in the previous examples due to the change of origin of the universe. Also, we assume that the perception system uses an accurate object tracker that assigns unique IDs to objects during consecutive frames. We used Id_1 , Id_2 , and Id_3 to refer to a car in the current frame, a car in the previous frame, and the drivable region. Note that we needed to use the previous frame operator to check if an existing car was absent in the previous frame. That is, in $\varphi_{existed}^{before}$, we check if the car identified by Id₂ matches with the same car in the previous frame. The antecedent of the formula as in Eq. (19) holds in the current frame, but the consequent of it fails because the car that exists in the current frame did not exist in the previous frame. Thus, the data stream shown as image frames in Fig. 9 falsifies the formula.

It is more realistic to formulate the same requirement using lane-based coordinate systems, if the perception system detects different lanes. Consequently, we can simplify the subformula φ_{close}^{dist} to encode if the cars drive in the same lane.

2) 3D occlusion scenario: In Figure 10, there are four frames captured at t_0 to t_3 . In the second and third frames, an occluded car (pointed to a red arrow) is detected. The perception system detected a car for which there is no information from the sensors (the cubes in frames 2 and 3 are empty, and the cameras cannot see the occluded vehicle). Although this can be the desired behavior for a perception system that does object tracking, here we are going to formalize a requirement that detects similar cases as wrong classifications.

For this example, we need to define a new 3D spatial function *Visible* by which we can check if given two cars are visible from the view of a third car. Additionally, we need to add a data attribute *Empty* to the data-object stream \mathcal{D} that determines if a bounding volume of an object is empty (e.g., $\mathcal{R}(\mathcal{D}(i), ID).PC = \emptyset$ checks if there is no point could data associated with an object identified by ID) or not. For instance, by $\mathcal{R}(\mathcal{D}(i), ID).Empty$ we check if an object identified by ID in the *i*'th frame is empty. We use function E(id) as a new syntax that serves as above.

$$\begin{bmatrix} V(id, CRT, id, id) \sim r \end{bmatrix} (\mathcal{D}, i, \epsilon, \zeta, \tau) \coloneqq \\ \begin{cases} \top \text{ if } f_{visible}(id, CRT, id, id, \mathcal{D}, i, \epsilon, \zeta) \end{pmatrix} \sim r \\ \perp \text{ otherwise}^* \end{cases}$$

 $f_{visible}(id_1, CRT, id_2, id_3, \mathcal{D}, i, \epsilon, \zeta)$ returns *true* if the $\epsilon(id_2)$ and $\epsilon(id_3)$ objects are visible from the view point CRT of $\epsilon(id_1)$ in the k'th frame, where $k \leftarrow \zeta(id_1)$ if $\zeta(id_1)$ is specified, and $k \leftarrow i$ otherwise. If one or both are invisible from the viewpoint of object $\epsilon(id_1)$, then the function returns *false*.

Example 5.11: The below requirement checks the robustness of the object detection in a perception system using spatial constraints and functions on point cloud data.

Req. 18: Any detected car (using the LiDAR data) driving the same direction on the left side of the ego car must include some point cloud data in its bounding volume unless it is occluded.

STPL (using Assumptions 5.1-5.3):

$$\phi_{18} = \Box \forall Id_1.\forall Id_2. \left(\left(\neg E(Id_1) \land \neg E(Id_2) \land (Id_1 \neq Id_2) \right) \\ \land \left(C(Id_1) = Car \right) \land \left(C(Id_2) = Car \right) \\ \land LAT(Id_1, RM) < -1 \land LAT(Id_2, RM) < -1 \\ \land LAT(Id_1, RM) > -6 \land LAT(Id_2, RM) > -6 \\ \land MD(Id_1) = MD(Id_2) \\ \land V(\mathbb{U}, CT, Id_1, Id_2) \\ \land \bigcirc \left(\neg V(\mathbb{U}, CT, Id_1, Id_2) \right) \right) \Longrightarrow \\ \bigcirc \left(V(\mathbb{U}, CT, Id_1, Id_2) \right) \\ \mathcal{R} \\ \left(E(Id_1) \land \neg E(Id_2) \right) \right)$$
(20)

In the above formula, we keep track of two cars using Id_1 and Id_2 that are to the left of the ego car (their distance has to be limited between 1 to 6 meters from the left side of the ego car). We assume that the perception system returns



(e) Annotated LiDAR point (f) Annotated LiDAR point (g) Annotated LiDAR point (h) Annotated LiDAR point cloud image at time t_1 . cloud image at time t_2 . cloud image at time t_3 .

Fig. 10: Two LiDAR and Camera frames annotated data are taken from NuScenes LidarSeg dataset (Scene-0399) (Caesar et al. (2020)). The images are taken by the back-left camera as represented in sensor-position image available at nuScenes website (Motional (2019)).

the moving direction of each classified object, and we use function MD to retrieve the direction for a given object identifier. Note that this function can be replaced by a spatial formula that uses the coordinates of an object in different frames to calculate its moving direction. The subformula $V(\mathbb{U}, CT, Id_1, Id_2)$ requires a visible angle between the two cars from the point of view of the ego car (i.e., the center point of the \mathbb{U}) to return true. We draw a sample visible angle in the first and fourth frames in yellow. For the two cars that we could find a visible angle for them in the first frame, there is none in frames 2 and 3. Additionally, there is no point cloud in the occluded car's bounding volume in frames 2 and 3 that satisfies the whole formula until the implies operator. The consequence of the implication is a release formula to be satisfiable from the next time/frame. In the release formula, the right-hand side requires the occluded object to be empty, and only when it becomes visible again, it can be non-empty. The whole formula is satisfiable for the pair of cars we discussed here. The data stream shown as image frames in Fig. 10 satisfies the formula in Eq. (20).

VI. MONITORING ALGORITHM

The offline monitoring algorithm for STPL is one of the main contributions of this work. We use a dynamic programming (DP) approach similar to Dokhanchi et al. (2016); Fainekos et al. (2012) since we are operating over data streams (timed state sequences). To keep the presentation as succinct as possible, we focus the presentation only to the future fragment of STPL. We divide our monitoring algorithm into four algorithms that work in a modular way. We present the pseudocode of the algorithms to analyze the worst case complexity of our STPL monitoring framework. In the following, we are going to use the formula

$$\varphi \coloneqq \Box \forall Id_1 @x. \Box \exists Id_2. \left(\forall \left(\overline{\sigma(Id_1)} \sqcup \sigma(Id_2) \right) \land \\ \forall \left(\overline{\sigma(Id_2)} \sqcup \sigma(Id_1) \right) \land Id_1 = Id_2 \right)$$

to explain some of the aspects of the future fragment of our monitoring algorithm.

A. Algorithm-1

This algorithm represents the main dynamic programming body of the monitoring algorithm. It receives an STPL formula φ , a data stream $\hat{\rho}$, and dynamic programming tables M and F. The formula is divided into subformulas as in the example in Fig. 11. That is, the formula is divided into subformulas based on the presence of freeze time operators (φ_i) and spatial quantifiers (φ'_i). It is important to note that the nested subformulas have lower index, e.g., φ_1 is nested in φ_2 as in Fig. 11. Notice that when we have subindexes, i.e., $\varphi_{i,j}$, we refer to the formula in the scope of *i*'th freeze variable and the *j*'th index from the root in the depth-first-search order.

The main loop of Algorithm 1 has 4 nested loops (lines 4-20). Further nested loops exist in *ComputeFnExpr* (Algorithm 3). These nested loops are also the source of the computational complexity in our monitoring algorithm. The loops in lines 4 and 5 are responsible for exploring all possible assignments to the freeze time variables starting from the inner nested freeze time operators. Line 6 explores the data stream backward in time to resolve the future fragment of Linear Temporal Logic (LTL) and the $S4_u$ terms. Lines 10-14 identify whether the subformula $\varphi_{k,j}$ is a spatio-temporal operator and calls the respective function.

B. Algorithm-2

This algorithm computes the values of the DP table M as defined in the temporal semantics of the STPL logic in Def. IV-B2. For a given subformula, first, it determines the operator



Fig. 11: A parse tree for the formula in Eq. 14. Distinct regions represent subformulas φ_2 , φ_1 , φ_1' , and φ_2' . In each region, the subformulas subscript into the operators and predicates levels.

and then fills the current columns of the DP tables accordingly. Most of the lines of the algorithm are straightforward where there is no DP table F in the assignments. The idea of using the separate DP table F for the freeze subformulas is to store the result of their evaluation at each frozen time, and then update the DP table M accordingly. Table M has two columns for each subformula, one for the current time and another for the next time. We use u_{01} and u_{10} to refer to the current column (current frame) for updating and the next column (next frame) for reading, respectively. Therefore, we need to toggle the value of these variables (between 0 and 1) to change the read column to write column and vice versa. Table F has two rows for each frame, one for the current time and the other for the next time. We use t_{01} and t_{10} to refer to the current row for updating and the next row for reading, respectively. Therefore, we need to toggle the value of these variables (between 0 and 1) to change the read row to write row and vice versa. The $M[j, u_{01}]$ val data-member of the DP table M keeps the current evaluating result of the j'th subformula φ_i at the current frame. The other data-members are used to update min/max values for the quantifier operators. The IT data-member is a table of size $(S_O)^{|V_{id}|}$, where S_O is the maximum number of objects in all input frames, and V_{id} is the set of the Id variables in the scope of the evaluating subformula φ_j . In Alg. 2, the only computationally expensive part is the update of the table in line 24. In practice, the table update is not an issue since the number of *id* variables is not high (at most 4-5 object variables in the examples in this paper) and the table update can be parallelized.

C. Algorithm-3

This algorithm computes the identifier table IT values of the DP table M. Here, we compute spatial functions and quantifier operators. For a given spatial function Fn(...), if it has nested

21

Algorithm 1 STPL Monitor

Input: φ , $\hat{\rho}$; Global variables: $M_{|\varphi|\times 2}$, $F_{2\times|\hat{\rho}|}$; Output: M[1,0]Comments: $\overline{M}_{|\varphi'| \times |\hat{\rho}|}$ is a table used to store the evaluation of spatial subformulas. For all the tables, the out-of-index accesses are handled case by case (i.e., \top or \perp is a default value) **Procedure** STPL-MONITOR(φ , $\hat{\rho}$) 1: $u_{01} \leftarrow 0$ and $t_{01} \leftarrow 0$ \triangleright toggle between 0 and 1 2: ← Collect all spatial terms, arithmetic expressions, and functions 3: $V \leftarrow$ Collect all freeze time variables in φ 4. for $k \leftarrow 1$ to |V| do 5: for $t \leftarrow 0$ to $|\hat{\rho}| - 1$ do \triangleright freeze (store) the frame for $u \leftarrow |\hat{\rho}| - 1$ to t do 6: 7: ▷ backward iteration for the future fragment of STPL 8: for $j \leftarrow \varphi_k.max$ down to $\varphi_k.min$ do 9. > iterate over subformulas under each freeze time operator 10: if $\varphi_j \in \varphi'$ then $ComputeFnExpr(\varphi, j, u, u_{01}, t, \hat{\rho})$ 11: 12: else $ComputeLTL(\varphi, j, u, u_{01}, t, t_{01}, k)$ 13: 14: end if 15: end for 16: $u_{01} \leftarrow toggle(u_{01})$ end for 17: 18: end for 19: $t_{01} \leftarrow toggle(t_{01})$ 20: end for 21: $k \leftarrow |V| + 1$ ⊳ handle the root node 22: if φ_k is not a freeze operator then 23: Repeat steps 6 to 17 for t = 024: else 25: $u_{01} \leftarrow toggle(u_{01})$, $t_{01} \leftarrow toggle(t_{01})$ 26: $M[1][u_{01}] = F[t_{01}][0]$ 27: end if 28: return $M[1, u_{01}]$ end procedure

function operators, then this algorithm recursively computes the final values. The IT table stores the values for any functional expression based on the combinatorial assignments of objects to the object variables. In line 10, if the combination of object identifiers does not satisfy the correct range of existing objects in the current frame or a frozen frame, then it assigns \perp (false). Otherwise, if any parameter is a spatial formula, then it runs algorithm 4. The size of the IT table is the maximum number of combinations for objects with respect to the object variables. For example, if there are a maximum of five objects per frame in an object stream, and a formula has a maximum of three nested object variables, then the size of IT is $5^3 = 125$. We choose the size of the IT table based on the worst case scenario, but for efficient computation, we only compute as many as needed combinations for each quantifier subformula based on its scope. Therefore, we mark the remaining cells of the table with NaN for extended computation that merges these tables for different subformulas in Algorithm 2 by calling function UpdateIdTable at line 24.

D. Algorithm-4

This algorithm computes the values of the DP table \overline{M} . It evaluates spatial-temporal formulas similar to the temporal formulas of Algorithm 2 with the modification that now set operations are used as opposed to Boolean operations. The data structures for the set representation and the corresponding set operations can affect the computational complexity of 4. Depending on the application, e.g., 2D vs 3D, a number of

Algorithm 2 LTL Monitor

Input: φ , *j*, *u*, *u*₀₁, *t*, *t*₀₁, *k*; **procedure** COMPUTELTL(φ , j, u, u_{01} , t, t_{01} , k) 1: $u_{10} \leftarrow toggle(u_{01}), t_{10} \leftarrow toggle(t_{01})$ 2: if $\varphi_j \equiv \top$ then 3: $M[j, u_{01}].val \leftarrow \top$ 4: else if $\varphi_j \equiv \tau - v_k \sim r$ then if $(\tau_u - \tau_t) \sim r$ then 5: 6: $M[j, u_{01}].val \leftarrow \mathsf{T}$ 7: else 8: $M[j, u_{01}].val \leftarrow \bot$ 9: end if 10: else if $\varphi_j \equiv \mathcal{F} - v_k \sim r$ then if $(u-t) \sim r$ then 11: 12: $M[j, u_{01}].val \leftarrow \top$ 13: else $M[j, u_{01}].val \leftarrow \bot$ 14: 15: end if 16: else if φ_j is a freeze subformula then 17: $M[j, u_{01}] \leftarrow M[j+1, u_{01}]$ if φ_j has an \exists quantifier then 18: 19: $M[j, u_{01}].val \leftarrow M[j, u_{01}].max$ 20: else $M[j, u_{01}].val \leftarrow M[j, u_{01}].min$ 21: 22: end if 23: $F[t_{01}][t] \leftarrow M[j, u_{01}]$ $UpdateIdTable(M[j, u_{01}].IT)$ 24: 25: else if $\varphi_j \equiv \neg \varphi_m$ then ▷ Apply ¬ to val, min, max, IT 26: $M[j, u_{01}] = \neg M[m, u_{01}]$ 27: else if $\varphi_j \equiv \varphi_m \lor \varphi_n$ then $M[j, u_{01}] \leftarrow \max(M[m, u_{01}], M[n, u_{01}])$ 28: 29: else if $\varphi_j \equiv \bigcirc \varphi_m$ then 30: if $u = |\hat{\rho}| - 1$ then 31: $M[j, u_{01}] \leftarrow \bot$ 32: else if φ_m is followed by $@\{x, ...\}$ then 33: $M[j, u_{01}] \leftarrow F[t_{10}, u+1]$ 34: else 35: $M[j, u_{01}] \leftarrow M[m, u_{10}]$ end if 36. 37: else if $\varphi_i \equiv \varphi_m \mathcal{U} \varphi_n$ then 38: if $u = |\hat{\rho}| - 1$ then 39: $M[j, u_{01}] \leftarrow M[n, u_{01}]$ 40: else 41: if φ_m is followed by x. then 42: $M[m, u_{01}] \leftarrow F[t_{10}, u]$ 43: end if if φ_n is followed by x. then $44 \cdot$ 45: $M[n, u_{01}] \leftarrow F[t_{10}, u]$ end if 4647: $M[j, u_{01}] \leftarrow$ $\max(M[n, u_{01}], \min(M[m, u_{01}], M[j, u_{10}]))$ 48. end if 49: end if end procedure

solutions are possible ranging from a linked list of pairs of 2D points (i.e., union of rectangles) to quadtrees or octrees (see next section).

E. Correctness and Complexity Analysis

Let $\hat{\rho}$ be an input signal of size $|\hat{\rho}|$, φ be an AAN STPL formula of size $|\varphi|$ (note that size of a formula is the summation of the number of spatial, temporal and spatio-temporal subformulas), $|V_t|$ be the size of freeze time variables (or 1 if there is none), S_{id} be the maximum number of used object variables in the scope of a quantifier operator, and S_{obj} be the maximum number of objects in a single frame in $\hat{\rho}$. If φ is a TPTL formula, then it is known from Dokhanchi et al. (2016) that the upper bound time complexity for the variable-bounded TPTL monitoring algorithm is $O(|V_t| \times |\varphi| \times |\hat{\rho}|^2)$,

Algorithm 3 Compute Function Expression Input: φ , k, u, u_{01} , t, $\hat{\rho}$;

Input: φ , k, u, u_{01} , t, $\hat{\rho}$; **procedure** COMPUTEFNEXPR($\varphi, k, u, u_{01}, t, \hat{\rho}$) 1: $\varphi_k \equiv Fn(\Omega_1, \ldots, \Omega_n) \sim r$ where $r \in \mathbb{R}, ~ \in \{>, <, \ge, \le, =, ! =\}$ and Fn is a reserved function name 2: $P_{n \times 1}$ is a list used to store the evaluated arguments of Fn3: $S_{id} \leftarrow (S_O)^{|V_{id}|}$ where S_O is the maximum number of objects in all the frames and V_{id} is the set of the object variables in the scope of φ_k 4: $S_{O}^{cf} \leftarrow$ number of objects in the current frame u 4: $S_{\vec{O}} \leftarrow number of objects in the current frame u$ $5: <math>S_{\vec{O}}^{ff} \leftarrow number of the objects in the freeze frame t$ 6: $F_{ID} \leftarrow$ is a map s.t. $\forall Id \in V_{id}, \{F_{ID}[Id] = true \mid \exists ``\exists Id@v_x.'' \in \varphi$ or " $\forall Id@v_x$." $\in \varphi$ }, $v_x \in V_t$ } 7: for $i \leftarrow 1$ to S_{id} do 8: ▷ iterates over combinatorial assignments of values to the IDs $\{ Id_1, \dots Id_{|V_{id}|} \} \leftarrow \text{ i'th combinatorial of } \{1, \dots S_O\}$ to $\{ Id_j \mid Id_j \in V_{id}, 1 \le j \le |V_{id}| \}$ 9: if $\exists Id \in V_{id}$ and $((F_{ID}[Id] = true \text{ and } Id > S_O^{ff})$ or 10: $(F_{ID}[Id] = false \text{ and } Id > S_O^{cf})$ then $M[k, u_{01}].IT[i-1] \leftarrow \bot$ 11: 12: else 13: for $i \leftarrow 1$ to n do \triangleright iterates over Fn arguments if Ω_j is a const number or reserved word then 14: 15: $\check{P}[j].val \leftarrow \Omega_j$ else if $\Omega_i \equiv Fn(\ldots)$ then 16: 17: P[j].val \leftarrow $ComputeFnExpr(\Omega_i, k, u, u_{01}, t, \hat{\rho})$ 18: else if $\Omega_i \equiv \sigma(Id)$ then 19: P[j].region \leftarrow BoundingBox(Id, F_{ID}, u, t) 20: else if Ω_i is a spatial formula then 21: P[j].region \leftarrow $Compute SpatioTemporal(\Omega_i, u, t, i, \hat{\rho})$ 22. end if 23: end for $M[k, u_{01}].IT[i-1] \leftarrow Fn(P) \sim r$ 24. end if 25: 26: end for 27: $S_{ID} \leftarrow (S_O)^L$ where L is the maximum number of object variables that can be used in the scope of any $@\{...\}$ subformula 28: for $i \leftarrow S_{id}$ to $S_{ID} - 1$ do \triangleright iterates over the remaining of the ID Table $M[k, u_{01}].IT[i] \leftarrow \text{NaN}$ 29. 30: end for end procedure

which is polynomial. Additionally, the upper bound space complexity of the presented algorithm by Dokhanchi et al. (2016) is $O(|\varphi| \times |\hat{\rho}|)$. Our STPL monitoring algorithm is founded based on the TPTL monitoring algorithm, but there are two major additions to the TPTL syntax and semantics. The first addition to the STPL grammar presented in Def. 4.1 is the existential/universal quantifiers that precedes the freeze time operator. The second addition is the production rule Θ . We call this extension of the TPTL language the Quantifiable TPTL (QTPTL). Next is the production rule \mathcal{T} that produces purely spatio-temporal formulas (Π and $\exists \mathcal{T}$ quantify the spatio-temporal in a constant time). The last addition to the QTPTL results in the STPL language. Therefore, we only analyze the time and space complexity of our algorithm for parts that concern the two aforementioned additions.

Our proposed monitoring algorithm is based on the Dynamic Programming (DP) algorithm by Dokhanchi et al. (2016) and, therefore, to consider the first addition, we need to evaluate each object variable related subformula at most

Algorithm 4 Spatio-Temporal Monitor (with intervals)

Input: $\theta, u', t, i, \hat{\rho}$; **procedure** COMPUTESPATIOTEMPORAL($\theta, u', t, i, \hat{\rho}$) 1: $\{Id_1, \dots Id_{|V_{id}|}\} \leftarrow i$ 'th combinatorial assignment of $\{1, \dots S_O\}$ to object variables 2: for $u \leftarrow |\hat{\rho}| - 1$ down to u' do \triangleright iterates backward over time stamps $S_{O}^{cf} \leftarrow$ number of objects in the current frame u $S_{O}^{ff} \leftarrow$ number of the objects in the freeze frame t 3: 4: 5: for $j \leftarrow \theta.max$ down to $\theta.min$ do ▷ iterates over subformulas if $\varphi_j \equiv \sigma(Id)$ then 6: if $((F_{ID}[Id] = true \text{ and } Id > S_O^{ff}) \text{ or }$ 7: $(F_{ID}[Id] = false \text{ and } Id > S_O^{cf})$ then 8: $\overline{M}[j,u] \leftarrow \emptyset$ ▷ mark as NaN 9: else 10: $\overline{M}[j, u] \leftarrow BoundingBox(Id, F_{ID}, u, t)$ end if 11: 12: else if $\varphi_j \equiv \overline{\varphi_m}$ then $\overline{M}[j, u] \leftarrow Complement(\overline{M}[m, u])$ 13: 14: else if $\varphi_j \equiv \varphi_m \sqcap \varphi_n$ then $\overline{M}[j, u] \leftarrow Intersection(\overline{M}[m, u], \overline{M}[n, u])$ 15: 16: else if $\varphi_j \equiv \varphi_m \sqcup \varphi_n$ then $\overline{M}[j, u] \leftarrow Union(\overline{M}[m, u], \overline{M}[n, u])$ 17: 18. else if $\varphi_j \equiv \mathbf{I} \varphi_m$ then $\overline{M}[j, u] \leftarrow Interior(\overline{M}[m, u])$ 19: else if $\varphi_j \equiv \mathbf{C} \ \varphi_m$ then 20: 21: $\overline{M}[j, u] \leftarrow Closure(\overline{M}[m, u])$ else if $\varphi_j \equiv \varphi_m \ \tilde{\mathcal{U}}_{\mathcal{I}}^s \varphi_n$ then if $u = |\hat{\rho}| - 1$ and $0 \in \mathcal{I}$ then 22: 23: 24: $\overline{M}[j,u] \leftarrow \overline{M}[n,u]$ 25: else if $u = |\hat{\rho}| - 1$ and $0 \notin \mathcal{I}$ then 26: $\overline{M}[j,u] \leftarrow \bot$ else if $\mathcal{I} = [0, +\infty)$ then 27: 28: $\overline{M}[j,u] \leftarrow \overline{M}[n,u] \sqcup$ $(\overline{M}[m,u] \sqcap \overline{M}[j,u+1])$ 29. else $b_l \leftarrow min\{j + \mathcal{I}\}; b_u \leftarrow max\{j + \mathcal{I}\}$ 30: 31: $r_{min} \leftarrow \sqcap_{j \le j' < b_l} M[m, j']$ $\overline{M}[j,u] \leftarrow \bot$ 32: for $j' \leftarrow b_l$ to b_u do $\overline{M}[j,j'] \leftarrow \overline{M}[j,j'] \sqcup (\overline{M}[n,j'] \sqcap r_{min})$ 33: ▷ iterates over frame indices 34: $r_{min} \leftarrow r_{min} \sqcap \overline{M}[m,j']$ 35: 36. end for 37: if $\sup \mathcal{I} = +\infty$ then $\overline{\dot{M}}[j,u] \leftarrow \overline{M}[j,u] \sqcup (\overline{M}[m,u] \sqcap \overline{M}[j,u+1])$ 38: 39. end if 40: end if end if 41: 42: end for 43: end for 44: return $\overline{M}[\theta.min, u']$ end procedure

 $(S_{obj})^{S_{id}}$ times. This also requires extra space to build the DP tables. Therefore, the upper bound time and space complexity of the QTPTL algorithms increases to $O((S_{obj})^{S_{id}} \times |V_t| \times |\varphi| \times |\hat{\rho}|^2)$ and $O((S_{obj})^{S_{id}} \times |\varphi| \times |\hat{\rho}|)$, respectively.

Finally, we consider spatial and spatio-temporal subformulas denoted as φ_s in addition to temporal ones denoted as φ_t to do complexity analysis of the STPL algorithm. It is easy to see that the production rule \mathcal{T} has the same grammar as MTL/STL, except that the logical operators are replaced with spatial ones. Therefore, the time/space complexity of monitoring these formulas follows the same complexity as in MTL/STL monitoring algorithms except for the spatial operations. In MTL/STL, all the logical operations compute in constant time. However, for spatial operations, depending on the used data structure for representing the spatial terms, this might not hold. In Appendix VI.

Appendix: Complexity Analysis of STE formulas, we calculate exponential lower bound for some spatio-temporal formulas where the linked-list was used to represent the spatial terms. That is, if we do not exploit the geometrical properties of the spatial terms while storing them, then we get exponential complexity for the spatial operations. Whereas, if we use some geometry-sensitive data structures such as region Quadtrees and region Octrees for storing and computing 2D and 3D spatial terms (e.g., see Aluru (2018); Shneier (1981)), respectively, then, we get a polynomial time and space complexity, e.g., Bournez et al. (1999). Assume that we can decompose our *d*-dimensional topological space (dimension is fixed to be 2D or 3D) into r^d cells, where r is the constant resolution along each axis. Construction of an image Quadtree/Octree is linear in the size of the image (see Shneier (1981)). The union and intersection algorithm for Quadtrees, in the worst case, requires visiting all the nodes in the trees, which can be done in K times. For computing a spatio-temporal formula φ_s , at each time-step, in the worst case, it requires as many spatial operations as linear to the size of the formula.

Therefore, the time complexity of computing the formula φ_s against an input signal $\hat{\rho}$ is as follows:

- $O(|\hat{\rho}| \times |\varphi_s| \times K)$, if there is no time/frame intervals in the formula and no frozen object variable is used in the formula. Note that K is a big constant (i.e., K is a function of the dimension and the resolution of the space) and we did not omit it to emphasize its impact on the computation.
- O(|ρ̂| × |φ_s| × K × c), if there are time/frame intervals in the formula and no frozen object variable is used in the formula. Here, c is defined as

$$c = \max_{0 \le j \le |\mathcal{D}|, \mathcal{I} \in T(\phi)} \left| \left[j, \max J(j, \mathcal{I}) \right] \right|$$

where $T(\phi)$ contains all the timing constraints \mathcal{I} which are either attached on the temporal operators in formula ϕ , or are timing constraints of the form $\tau - x \sim n$ or $\mathcal{F} - x \sim n$ in the scope of a temporal operator in ϕ . For example, if $\phi = \Box_{[0.1,\infty)}\psi$, then $[0.1,\infty) \in T(\phi)$, or if $\phi = x \ \Box \ (\tau - x \leq 5.5 \Rightarrow \psi)$, then $[0,5.5] \in T(\phi)$. Intuitively, the function J returns all the samples that satisfy a constraint \mathcal{I} from the set $T(\phi)$ starting from a sample j. Formally,

$$J(j,\mathcal{I}) = \begin{cases} \tau^{-1}((\tau(j) +_R \mathcal{I})) \\ (\tau(j+1) +_R \mathcal{I})) & \text{if } \sup \mathcal{I} = +\infty \\ \tau^{-1}(\tau(j) +_R \mathcal{I}) & \text{otherwise} \end{cases}$$

with $t +_R \mathcal{I} = \{t'' \in R \mid \exists t' \in \mathcal{I}, t'' = t + t'\}$. Note that when considering constraints on the number of frames, i.e., $\mathcal{F} - x \sim n$, then the timestamp mapping τ is the identity function. For a discussion on *c* for STL/MTL with discrete time semantics see Fainekos et al. (2012).

- O(|p̂|×|φ_s|×K×|V_t|), if there is no time/frame intervals in the formula, but there are frozen object variables used in the formula.
- O(|ρ̂|×|φ_s|×K×c×|V_t|), if there are time/frame intervals in the formula, and there are frozen object variables used in the formula.

Overall, the upper bound time and space complexity of the STPL algorithm are $O((S_{obj})^{S_{id}} \times |V_t| \times |\varphi| \times |\hat{\rho}|^2 \times K \times c)$ and $O((S_{obj})^{S_{id}} \times |\varphi| \times |\hat{\rho}| \times K)$, respectively.

In our implementation of the monitoring algorithms (discussed in Monitoring Algorithm), we focus on the future fragment of the STPL logic to avoid complexity (e.g, by excluding past-time operators as presented in Appendix VI. Appendix: STPL Future Syntax). That is, we improved the space complexity of STPL formulas without spatial terms by decoupling the DP tables into two separate tables: one dedicated to the values of subformulas at the current time step, and the other for their frozen values along the time horizon. Therefore, we reduced the space complexity to $O((S_{obj})^{S_{id}} \times (|\varphi_t| + |\hat{\rho}|))$ for non-spatial STPL formulas and spatial formulas without time/frame intervals in them. For improving the exponential complexity of the spatial STPL formulas, we merge the fragmented subsets after spatial operations. Additionally, if there is no frozen object variable used in a spatial formula, we only evaluate it once. Some optimizations can be done based on the content of the formulas, for example, if a temporal formula does not have globally, eventually, until and release operators in it, then we deduce the needed horizon length of the input signal accordingly (i.e., next time operator only requires the evaluation of the first two frames). Also, we interpret the time/frame constraints in a formula to possibly ignore the evaluation of the affected subformulas accordingly.

The correctness of the algorithm with respect to the presented syntax and semantics of STPL can be proven by using the correctness proofs that are presented for the TPTL and MTL monitoring algorithms by Dokhanchi et al. (2016) and Fainekos et al. (2012).

We have released an open-source version of the code for both Linux and Windows OS in a public repository in GitLab (see STPL (2023)). Our tool can be run in standalone mode or as part of Matlab. Additionally, there are data stream files and input configuration files that cover most of the examples in the previous sections, as well as the sensitivity analysis result in the following section.

F. A Polynomial Time/Space Fragment

In this section, we identify STPL specification templates for which the monitoring problem becomes polynomial time in the worst case. In the template below, we assume that the size of the spatial formula φ_s is bounded.

1) Example: The complexities of evaluating an arbitrary SPE formula φ_s (the formula is in conjunctive form) on a data stream $\hat{\rho}$ is

- O(2^a3^b) for time and space, if there is no complement operator in the formula, where a, b ∈ N, and we have: arg max (2^a3^b | (|φ_s| + 1)/2 = 2a + 3b). For instance, φ_s := (𝒯 ⊔ 𝒯 ⊔ 𝒯) ∩ (𝒯 ⊔ 𝒯 ⊔ 𝒯).
 O(4^{((|φ_s|+1)/3)}) for time and space, if there are com-
- $O(4^{((|\varphi_s|+1)/3)})$ for time and space, if there are complement operators in the NNF formula. For instance, $\varphi_s := \overline{\mathcal{T}} \sqcap \overline{\mathcal{T}} \sqcap \overline{\mathcal{T}} \sqcap \overline{\mathcal{T}} \sqcap \overline{\mathcal{T}} \sqcap \overline{\mathcal{T}}.$

2) *Example:* An arbitrary *l*-level nested spatial globally formula φ_s^{\Box} , with a spatial term as its right-most subformula,

is of $O(|\hat{\rho}|)$ time and O(1) space complexity.

$$\varphi_s^{\Box} \coloneqq \Box_{\mathcal{I}}^s \Big(\mathcal{T} \sqcap \Box_{\mathcal{I}}^s \big(\mathcal{T} \sqcap \Box_{\mathcal{I}}^s \big(\mathcal{T} \sqcap \Box_{\mathcal{I}}^s \mathcal{T} \big) \big) \Big)$$

3) Example: An arbitrary *l*-level nested spatial eventually formula φ_s^{\diamondsuit} , with a spatial term as its right-most subformula, is of $O(|\hat{\rho}|^{(l+1)})$ time and space complexity.

$$\varphi_s^{\diamondsuit} \coloneqq \diamondsuit_{\mathcal{I}}^s \big(\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s \big(\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s (\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s \mathcal{T}) \big) \big)$$

4) Example: An arbitrary *l*-level nested spatial eventually subformula φ_s^{\diamond} followed by an arbitrary *k*-level nested spatial globally subformula φ_s^{\Box} , with a spatial term as its right-most subformula, is of $O(|\hat{\rho}|^{(l+1)})$ time and space complexity.

$$\begin{split} \varphi_s^{\diamondsuit,\square} &\coloneqq \diamondsuit_{\mathcal{I}}^s \Big(\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s \big(\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s \big(\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s \big(\mathcal{T} \sqcup \diamondsuit_{\mathcal{I}}^s \big) \Big) \Big) \Big) \\ \Big(\square_{\mathcal{I}}^s \big(\mathcal{T} \sqcap \square_{\mathcal{I}}^s \big(\mathcal{T} \sqcap \square_{\mathcal{I}}^s \mathcal{T} \big) \big) \big) \big) \Big) \end{split}$$

5) *Example:* A right-hand-side *l*-level nested spatial until formula $\varphi_s^{\mathcal{U}}$, where the left-hand-side of all the until operators are a single spatial term, is of $O(|\hat{\rho}|^{(l+2)})$.

$$\varphi_s^{\mathcal{U}} \coloneqq \mathcal{T} \ \mathcal{U}_{\mathcal{I}}^s \big(\mathcal{T} \ \mathcal{U}_{\mathcal{I}}^s \big(\mathcal{T} \ \mathcal{U}_{\mathcal{I}}^s \big(\mathcal{T} \ \mathcal{U}_{\mathcal{I}}^s \left(\mathcal{T} \ \mathcal{U}_{\mathcal{I}}^s \left(\mathcal{T} \ \mathcal{U}_{\mathcal{I}}^s \right) \right) \big)$$

6) Example: A left-hand-side *l*-level nested spatial release formula $\varphi_s^{\mathcal{R}}$, where the right-hand-side of all the until operators are a single spatial term, is of $O(|\hat{\rho}|^{(l+2)})$.

$$\varphi_s^{\mathcal{R}} \coloneqq \left(\left(\left(\mathcal{T} \, \mathcal{R}_{\mathcal{I}}^s \, \mathcal{T} \right) \mathcal{R}_{\mathcal{I}}^s \, \mathcal{T} \right) \mathcal{R}_{\mathcal{I}}^s \, \mathcal{T} \right) \mathcal{R}_{\mathcal{I}}^s \, \mathcal{T} \right)$$

VII. EXPERIMENTS AND RESULTS

We selected some of the presented example formulas to cover different possible combinations for the operators and quantifiers, and to demonstrate how the computation time scales concerning the size of the data stream and the formulas. For this experimental analysis, we used the DeepDrive dataset ⁵ (Yu et al. (2020)). As indicated in the last column of Table III, the performance of the STPL monitoring algorithm for hundreds of monitoring frames (including thousands of objects) is feasible for offline monitoring. Some statistics about the experiments are summarized in Table III. We used a Windows 10 machine with Intel Core i7 CPU 8550U @ 1.8GHZ, 16GB RAM, and Matlab R2020a. The STPL monitoring algorithm is implemented in *C* language and compiled for Matlab.

The formulas in Table III were selected based on the types of operators and their computational complexity. The maximum number of nested quantifiers is a source of exponential complexity (i.e., $(S_{obj})^{S_{id}}$) in our monitoring algorithm. The highest number of nested quantification operators was 3 in Formulas (16) and (17). Another source of complexity is the number of spatial operators in the formula. For instance, the worst execution time is observed for the Formula (14) due to its higher number of spatial operators. These results are not surprising and are in agreement with our theoretical analysis in Section VI-E.

In order to study the impact of the number of objects in each frame on the monitoring algorithm, we manipulated the

TABLE III: Statistics on execution-time for different formulas and data stream sizes. We used the Berkeley DeepDrive (BDD) dataset to compute the results. m-time and e-time represent the required time (in second) for releasing memories and executing the monitoring algorithm, respectively.

$ \hat{ ho} $	$ \varphi_t $	$ \varphi_s $	$ V_t $	S_{obj}	S_{id}	m-time	e-time		
quantifier-formed STPL Formula (9) without spatial operators									
25	9	0	0	20	1	0	0.002		
50	9	0	0	20	1	0	0.001		
100	9	0	0	23	1	0	0.005		
200	9	0	0	24	1	0	0.008		
n	mix-formed STPL Formula (10) without spatial operators								
25	7	0	1	20	2	0	0.132		
50	7	0	1	20	2	0	0.519		
100	7	0	1	23	2	0	2.76		
200	7	0	1	24	2	0	11.31		
	mix-fo	rmed ST	ГPL For	mula (14)) with S	PE operator	rs		
25	9	8	1	20	2	0.004	1.25		
50	9	8	1	20	2	0.003	4.13		
100	9	8	1	23	2	0.005	16.32		
200	9	8	1	24	2	0.005	63.52		
q	uantifier	-formed	STPL I	Formula (15) with	STE opera	ators		
25	3	4	0	20	1	0	0.006		
50	3	4	0	20	1	0	0.029		
100	3	4	0	23	1	0	0.119		
200	3	4	0	24	1	0	0.176		
	mix-fe	ormed S	TPL For	rmula (<mark>16</mark>) with s	patial terms	8		
25	29	6	1	20	3	0.023	2.23		
50	29	6	1	20	3	0.023	4.97		
100	29	6	1	23	3	0.037	16.07		
200	29	6	1	24	3	0.043	46.88		
	mix-for	med ST	PL Forn	nula (<mark>17</mark>)	without	spatial terr	ns		
25	29	0	1	20	3	0	1.16		
50	29	0	1	20	3	0	2.73		
100	29	0	1	23	3	0	10.41		
200	29	0	1	24	3	0	33.69		

BDD dataset to create artificial datasets with specific number of objects. The result is presented in Table IV. In the first row, the maximum number of objects in all the frames is 5. For all the following rows, this number is doubled. Based on the $O((S_{obj})^{S_{id}})$, the *e-time* of rows should increase with the ratio of $2^3 = 8$. To remedy the exponential complexity of the nested quantified formulas, we can use parallel computation to efficiently evaluate quantified subformulas. More specifically, Algorithm 3 can be parallelized.

Since the theoretical time complexity of offline STPL monitoring is the same as the time complexity of online pasttime STPL monitoring (see Balakrishnan et al. (2021) for a toolbox), the problem of online monitoring of bounded time short duration STPL properties is practically feasible and relevant. However, the offline STPL monitoring problem over extremely large perception datasets may not be feasible without further optimizations or expressivity restrictions. One promising direction is to prefilter very large perception datasets

$ \hat{ ho} $	$ arphi_t $	$ \varphi_s $	$ V_t $	S_{obj}	S_{id}	m-time	e-time
25	29	6	1	5	3	0.001	0.060
25	29	6	1	10	3	0.005	0.441
25	29	6	1	20	3	0.043	2.419
25	29	6	1	40	3	0.312	20.027

and extract subsequences that are interesting for STPL specifications. One such possibility for filtering is the perception query language SpRE (Anderson et al. (2023)) – see Section VIII for a brief discussion. We plan to pursue such a direction in the future.

VIII. RELATED WORKS

In this section, we provide a detailed overview of related works. We primarily focus on comparing STPL with other spatio-temporal logics. In Table V, we provide a summary comparison of the most relevant logics for easy reference. We conclude the section with some references on promising directions on the analysis of perception systems which are not directly related to spatio-temporal logics.

Region Connection Calculus by Cohn et al. (1997), Shape Calculus by Bartocci et al. (2010), and Situation Calculus by Bhatt and Loke (2008) are just some examples of the vast literature on logics about topology and space. Comprehensive surveys and comparisons of reasoning methods about topology and time are provided by Dylla et al. (2017); Aiello et al. (2007). Spatio-temporal logics and calculi are also frequently used in robotics for human-robot communication (Kress-Gazit and Pappas (2010); Summers-Stay et al. (2014)) and for specifying and verifying properties of AV (Linker and Hilscher (2013); Loos et al. (2011)). All the aforementioned works that deal with topology and time primarily focus on deductive reasoning, theorem proving, knowledge representation, axiomatization, and - in some cases - planning. In contrast, STPL requires computationally efficient tools for spatio-temporal reasoning in order to monitor data streams from perception systems.

Even though there exist spatio-temporal logics that can process spatio-temporal data (offline or online) such as SpaTeL (Haghighi et al. (2015)), SSTL (Nenzi et al. (2015)), or SaSTL (Ma et al. (2020)) (for a short survey see Bartocci et al. (2018)), or even images, e.g., SLCS (Buonamici et al. (2019)), all these logics are application dependent and cannot support the topological reasoning needed for perception data in AV. To highlight the fundamental differences between the aforementioned logics and STPL, we provide a detailed comparison with SpaTeL and SSTL. The differences with the other listed logics and conceptually similar in scope. The two spatio-temporal languages (SSTL and SpaTeL) are explicitly developed for describing high-level spatial patterns that evolve. Both languages are founded based on a graphbased representation of discrete models of space. For the SSTL, undirected weighted graphs are used to model space,

Language	Temporal Foundation	∃∀ (data)	∃∀ (spatial)	Spatial Foundation	Domain in practice	Applications
STPL	AAN-TPTL	\checkmark	\checkmark	$S4_u$	Sets in Euclidean spaces	Perception systems
TQTL	AAN-TPTL	\checkmark		Predicates	Perception data	Vision based (2D) perception
SpRE	FSM		\checkmark	$S4_u$	Sets in Euclidean spaces	Perception systems
STSL	STL		\checkmark	$S4_u$	Distances in Euclidean spaces	System level requirements for CPS
SpaTel	STL		\checkmark	TSSL	Quadrants	Pattern recognition
SSTL	STL		\checkmark	Graph-based modeling	Distances in Euclidean spaces	Pattern recognition
GSTL	STL		\checkmark	Mereotopology	Cubics	Knowledge representation

and in SpaTeL, a networked system whose states encapsulate an image are represented as quad transition systems.

In more detail, in SSTL, the syntax of the language adds two spatial operators $\otimes_{[w_1,w_2]}\varphi$ and $\varphi_1 S_{[w_1,w_2]}\varphi_2$ into Signal Temporal Logic (STL) (Maler and Nickovic (2004)), which are named bounded somewhere and bounded surround, respectively. The first operator requires φ to hold in a location that can be reached from the current location with a cost between w_1 and w_2 . The cost is usually the distance between the two locations. For the second operator, the notation of external boundary of a set is required. An external boundary of a given set of nodes is defined as the set of nodes that are directly connected to the elements of the given set but are not members of it. The semantics of the second operator requires that for the current location l and a given trace x, l belongs to a set of locations A that all satisfy the formula φ_1 , and for all the locations in the external boundary of A, they satisfy the formula φ_2 . An SSTL formula can be arbitrarily nested.

In SpaTeL, a combination of Tree Spatial Superposition Logic (TSSL) (Gol et al. (2014)) and STL is proposed to reason on spatial patterns. TSSL uses quad-trees to represent the space by partitioning the space recursively into quadrants. The TSSL logic is similar to the classic Computation Tree Logic (CTL) (e.g., see Huth and Ryan (2004)), with the main difference that the next and until operators are not temporal, but spatial. That is, evolution happens by a change of resolution (or zoom in). All the spatial operators are augmented by a set B that selects the spatial directions (i.e., NW, NE, SW, and SE) in which the operators are allowed to work. Additionally, similar to temporal operators, there is a parameter k that limits the operator's evaluation range on a finite sequence of states. For example, $\exists_B \varphi_1 U_k \varphi_2$ means that there exists a set of directions in B by which the *i*-th label of a path π^B satisfies the formula φ_2 and all the other labels on the path until *i* satisfy the formula φ_1 . A difference between the former and the latter is that in the former one, the TSSL fragment of a formula does not include temporal subformulas.

In summary, the key differences of these logics with our proposed STPL logic are:

- *Modeling*: SSTL and SpaTeL are not designed to model physical objects in 2D or 3D spaces. On the other hand, our logic is explicitly designed to handle physical objects.
- *Expressivity*: SpaTeL is inherently less expressive than SSTL due to its modeling and traversing approach on quad-trees and the decoupled syntax for spatial and temporal formulas. Therefore, we are going to compare STPL with SSTL. There are two significant differences. The first one is the presence of time freeze operator and time variables and, hence, STPL is more expressive. The second one is the presence of the quantifiers and set operations over spatial terms/locations. As an example, SSTL cannot reason on whether the same object over two different frames overlaps or not with itself.
- *Application*: SSTL and SpaTeL are mostly helpful for pattern recognition purposes, while STPL is a more general-purpose language. Quantitative semantics: there is quantitative and qualitative semantics for SSTL and SpaTeL, but currently, we only presented qualitative semantics for STPL. The graph-based modeling of the spatial environment and the fixed metric properties such as distance makes it more straightforward to define quantitative semantics for their underlying logic.

In another line of work, a graph-based spatio-temporal logic – GSTL by Liu et al. (2020) – is presented for knowledge representation and reasoning. GSTL deals with spatial elements as regions, and uses *mereotopology* (combination of *mereology* and *topology* to support parthood and connectivity, respectively) to represent relations between spatial elements. It exploits rectangle/cubic algebra to represent spatial objects. GSTL combines STL temporal logic with mereotopology-based spatial operators enriched with interval algebra. The satisfiability problem for GSTL is decidable by restricting the evolution of spatial elements. GSTL was primarily designed for model checking which restricts its expressivity for decid-

ability reasons.

The works closest to ours stem from combining temporal logics with spatial logics (for a historical overview and a discussion on $S4_u$ see Kontchakov et al. (2007)). Gabelaia et al. (2005) combine Linear Temporal Logic (LTL) (Manna and Pnueli (1992)) with $S4_u$ to define the logic $\mathcal{PTL} \times S4_u$. They further define several fragments of $\mathcal{PTL} \times S4_u$ and they study the decidability of the satisfiability problem. However, the problem of offline monitoring is not investigated in this line of work.

More recently, STSL was proposed by Li et al. (2021) where STL is combined with $S4_u$. Even though the monitoring problem is studied for STSL, STSL falls short of the goals of STPL in multiple directions. First and foremost, STSL does not support generic data and quantification over such data. That is, it is not possible to express a property such as Req. 2 where we need to quantify over the bounding boxes of all the cars in a frame. Second, STPL is based on TPTL which is a strictly more expressive logic than STL used in STSL. Third, a theoretical or experimental computational complexity analysis is not presented for STSL to identify what fragments are computationally important while still being practically relevant. Finally, and most importantly, the applications presented for STSL are restricted to properties over numerical trajectories produced by Cyber-Physical Systems, and it is clear that with the metric space chosen for STSL, the corresponding formal specifications can be expressed in STL. That is, in practice, there is no gain in expressive power in STSL over STL.

Another line of research relevant to our work is formal languages for analysis of perception systems. Timed Quality Temporal Logic (TQTL) by Dokhanchi et al. (2018a) was designed to reason over streams of perception data. TQTL is built upon the AAN fragment of Timed Propositional Temporal Logic (AAN-TPTL) (Dokhanchi et al. (2016)) by introducing quantification (\exists, \forall) over the objects in each frame, and by introducing functions that retrieve data relevant to each object, e.g., class, probabilities, bounding box coordinates, etc. The AAN fragment of TPTL was chosen due to its polynomial time complexity while still being strictly more expressive then STL. Note that further algorithmic improvements on AAN-TPTL are possible, e.g., see Elgyutt et al. (2018); Ghorbel and Prabhu (2022). Nevertheless, TOTL cannot reason directly about properties of bounding boxes. For example, TQTL cannot reason about self-overlap of bounding boxes across time, i.e., TQTL cannot express Req. 2. More generally, TQTL cannot reason about 3D scenes (e.g, bird-eye view of the world) since this requires a mechanism to relate spatially different objects in the environment. STPL resolves these shortcomings of TQTL to enable a versatile framework to reason about perception systems in both 2D and 3D (along with other state variables included in the perception data).

Spatial Regular Expressions (SpREs) were recently introduced by Anderson et al. (2023) to find patterns in streaming data. That is, given a SpRE, the goal is to find all the sequences of frames that match the pattern specified by SpRE. SpREs were designed to closely resemble regular expressions, and, hence, the underlying model of computation for processing streaming perception data is automata (Sipser (2006)). The current version of SpRE does not support quantification over data in order to enable online real-time processing. However, SpRE supports $S4_u$ operators on per frame basis. Clearly, SpRE is less expressive than STPL, but we envision an interplay between the two languages. SpRE can potentially find very quickly the subsequences of streaming data over which we need to run the more expressive STPL requirements.

The PerSyS monitoring system by Antonante et al. (2021) presents a mathematical model for fault detection in perception systems. The base of their work is Perfect Minicomputer Corporation (PMC) model in multiprocessor systems, which is generalized to account for models with heterogeneous outputs (i.e., perception systems), and equipped with temporal dimensions to support interaction among PMC models. Their system supports consistency checking among different sensory outputs of a perception system with some formal guarantees on the maximum number of inconsistencies. In PerSyS, it is possible to design models that identify faults, but, similar to any other graph-based modeling technique, it is highly reliant on a correct model to begin with, and then adding formalized requirements as a set of constraints on the models (i.e., constraints on the edges of the PMC graphs). STPL monitoring goals are orthogonal to PerSyS. STPL is a specification language that formalizes assumptions and guarantees on the functional performance of the perception system. As such a language, it is more expressive than the constraints used in PerSyS. As a byproduct, STPL can also function as a comparison framework between different perception stacks.

Finally, the language Scenic by Fremont et al. (2018) has been developed for creating single scene images for testing object detection and classification algorithms. However, our work is complementary to languages that generate static scenes.

IX. CONCLUSIONS

In this paper, we presented Spatio-Temporal Perception Logic (STPL), which is a logic which is specifically designed for reasoning over data streams of perception systems. STPL merges and extends other practical logics such as TPTL (Alur and Henzinger (1994)) and $S4_u$ (Aiello et al. (2007)) with data (object) quantification, functions and relations that enable topological reasoning over time. Our new logic can be used for specifying correctness requirements on perception systems, as well as to compare different machine learning stacks on their performance beyond the standard metrics (e.g., see Mallick et al. (2023)). We have identified fragments of STPL which are efficiently monitorable for perception applications, and we have demonstrated that practically relevant requirements which do not fall within these fragments can still be efficiently monitorable in practice. An open source publicly available toolbox has been developed STPL (2023) which can be used for offline perception system analysis. An online monitor for the past fragment of STPL is also available (Balakrishnan et al. (2021)). Using STPL, we have been able to discover inconsistencies in publicly available training datasets for AV/ADAS.

Since STPL formulas are rather complex even for experts, we have have been working toward developing a Domain Specific Language (DSL) called PyFoReL (Anderson et al. (2022)) for easier elicitation and maintenance of STPL requirements. PyFoReL provides a Pythonic language to compose requirements in a modular way while enforcing that they are valid STPL formulas. The next step would be to interface PyFoReL and/or the STPL syntax with Large Language Models (LLM). Similar work has been done in the past for LTL and STL with practically relevant success (e.g., see Pan et al. (2023); Fuggitti and Chakraborti (2023)). In addition, verification and debugging tools for STPL formulas will be needed since LLMs cannot be trusted to always produce correct translations. In the past, we have done similar work for STL/MTL specifications in Dokhanchi et al. (2018b). We expect to achieve further computational improvements on our monitoring algorithms by parallelization and by filtering relevant sequences of data through our new query language SpRE (Anderson et al. (2023)) before the STPL tools are used. Finally, it would also be interesting to see if meaningful robust semantics (Bartocci et al. (2018)) could be defined in order to support test case generation or even self-supervised training of neural networks.

ACKNOWLEDGMENT

This work was partially supported by NSF under grants CNS-2039087, CNS-2038666, IIP-1361926, and the NSF I/UCRC Center for Embedded Systems.

REFERENCES

- Abbas H, O'Kelly ME, Rodionova A and Mangharam R (2017) A driver's license test for driverless vehicles. *Me-chanical Engineering* 139: S13–S16.
- Aiello M, Pratt-Hartmann IE and van Benthem JF (2007) Handbook of spatial logics. Springer.
- Alur R and Henzinger TA (1994) A really temporal logic. J. ACM 41: 181–203.
- Aluru S (2018) Quadtrees and octrees. In: *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, pp. 309–326.
- Anderson J, Fainekos G, Hoxha B, Okamoto H and Prokhorov D (2023) Pattern matching for perception streams. In: 23rd International Conference on Runtime Verification (RV).
- Anderson J, Hekmatnejad M and Fainekos G (2022) PyFoReL: A domain-specific language for formal requirements in temporal logic. In: *IEEE 30th International Requirements Engineering Conference (RE).*
- Antonante P, Spivak DI and Carlone L (2021) Monitoring and diagnosability of perception systems. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 168–175.
- Balakrishnan A, Deshmukh J, Hoxha B, Yamaguchi T and Fainekos G (2021) Percemon: Online monitoring for perception systems. In: *International Conference on Runtime Verification (RV), LNCS*, volume 12974.
- Baotic M (2009) Polytopic computations in constrained optimal control 50: 119–134.
- Bartocci E, Corradini F, Berardini MRD, Merelli E and Tesei L (2010) Shape calculus. a spatial mobile calculus for 3d shapes. *Scientific Annals of Computer Science* 20: 1–31.

- Bartocci E, Deshmukh J, Donzé A, Fainekos G, Maler O, Nickovic D and Sankaranarayanan S (2018) Specificationbased monitoring of cyber-physical systems: A survey on theory, tools and applications. In: *Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS*, volume 10457. Springer, pp. 128–168.
- Bashetty SK, Amor HB and Fainekos G (2020) DeepCrashTest: turning dashcam videos into virtual crash tests for automated driving systems. In: *International Conference on Robotics and Automation (ICRA)*.
- Basin D, Klaedtke F, Müller S and Zălinescu E (2015) Monitoring metric first-order temporal properties 62(2).
- Bhatt M and Loke S (2008) Modelling dynamic spatial systems in the situation calculus. *Spatial Cognition & Computation* 8: 86–130.
- Bournez O, Maler O and Pnueli A (1999) Orthogonal polyhedra: Representation and computation. In: *International Workshop on Hybrid Systems: Computation and Control.* Springer, pp. 46–60.
- Buonamici FB, Belmonte G, Ciancia V, Latella D and Massink M (2019) Spatial logics and model checking for medical imaging. *International Journal on Software Tools for Technology Transfer*: 1–23.
- Caesar H, Bankiti V, Lang AH, Vora S, Liong VE, Xu Q, Krishnan A, Pan Y, Baldan G and Beijbom O (2020) nuscenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11621–11631.
- Campbell J, Amor HB, Ang MH and Fainekos G (2016) Traffic light status detection using movement patterns of vehicles. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC). IEEE, pp. 283– 288.
- Cimatti A, Roveri M and Sheridan D (2004) Bounded verification of past ltl. In: *International Conference on Formal Methods in Computer-Aided Design*. Springer, pp. 245–259.
- Cohn AG, Bennett B, Gooday J and Gotts NM (1997) Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica* 1(3): 275–316.
- Corso A, Moss RJ, Koren M, Lee R and Kochenderfer MJ (2020) A survey of algorithms for black-box safety validation. *arXiv preprint arXiv:2005.02979*.
- Cosler M, Hahn C, Mendoza D, Schmitt F and Trippel C (2023) nl2spec: Interactively translating unstructured natural language to temporal logics with large language models.
 In: *Computer Aided Verification*, *LNCS*, volume 13965.
 Springer, pp. 383–396.
- DeCastro J, Leung K, Aréchiga N and Pavone M (2020) Interpretable policies from formally-specified temporal properties. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems Conference (ITSC).
- Dokhanchi A, Amor HB, Deshmukh JV and Fainekos G (2018a) Evaluating perception systems for autonomous vehicles using quality temporal logic. In: *International Conference on Runtime Verification*. Springer, pp. 409–416.
- Dokhanchi A, Hoxha B and Fainekos G (2018b) Formal requirement debugging for testing and verification of cyberphysical systems. *ACM Transactions on Embedded Com*-

puting Systems 17. DOI:10.1145/3147451.

- Dokhanchi A, Hoxha B, Tuncali CE and Fainekos G (2016) An efficient algorithm for monitoring practical tptl specifications. In: 2016 ACM/IEEE International Conference on Formal Methods and Models for System Design (MEM-OCODE). IEEE, pp. 184–193.
- Dreossi T, Donzé A and Seshia SA (2019a) Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning* 63: 1031– 1053.
- Dreossi T, Fremont DJ, Ghosh S, Kim E, Ravanbakhsh H, Vazquez-Chanlatte M and Seshia SA (2019b) Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In: *International Conference on Computer Aided Verification*. Springer, pp. 432–442.
- Dylla F, Lee JH, Mossakowski T, Schneider T, Delden AV, Ven JVD and Wolter D (2017) A survey of qualitative spatial and temporal calculi: Algebraic and computational properties. *ACM Computing Surveys* 50.
- Eisner C and Fisman D (2006) Weak vs. strong temporal operators. *A Practical Introduction to PSL* : 27–34.
- Elgyutt A, Ferrere T and Henzinger TA (2018) Monitoring temporal logic with clock variables. In: *Formal Modeling and Analysis of Timed Systems (FORMATS), LNCS,* volume 11022. Springer.
- Fainekos GE, Sankaranarayanan S, Ueda K and Yazarel H (2012) Verification of automotive control applications using s-taliro. In: 2012 American Control Conference (ACC). IEEE, pp. 3567–3572.
- Fremont DJ, Kim E, Pant YV, Seshia SA, Acharya A, Bruso X, Wells P, Lemke S, Lu Q and Mehta S (2020) Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In: 23rd IEEE International Conference on Intelligent Transportation Systems (ITSC).
- Fremont DJ, Yue X, Dreossi T, Ghosh S, Sangiovanni-Vincentelli AL and Seshia SA (2018) Scenic: Languagebased scene generation. Technical report, arXiv:1809.09310.
- Fuggitti F and Chakraborti T (2023) Nl2ltl a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas 37(13): 16428–16430.
- Gabelaia D, Kontchakov R, Kurucz A, Wolter F and Zakharyaschev M (2005) Combining spatial and temporal logics: expressiveness vs. complexity. *Journal of artificial intelligence research* 23: 167–243.
- Geiger A, Lenz P, Stiller C and Urtasun R (2013a) Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research* 32(11): 1231–1237.
- Geiger A, Lenz P, Stiller C and Urtasun R (2013b) Vision meets robotics: The KITTI dataset. *International Journal* of Robotics Research (IJRR) 32: 1231–1237.
- Ghorbel B and Prabhu V (2022) Linear time monitoring for one variable tptl. In: 25th ACM International Conference on Hybrid Systems: Computation and Control (HSCC).
- Gladisch C, Heinz T, Heinzemann C, Oehlerking J, von Vietinghoff A and Pfitzer T (2019) Experience paper: Searchbased testing in automated driving control applications. In: 34th IEEE/ACM International Conference on Automated Software Engineering (ASE).

- Gol EA, Bartocci E and Belta C (2014) A formal methods approach to pattern synthesis in reaction diffusion systems. In: *53rd IEEE Conference on Decision and Control.*
- Gordon D, Farhadi A and Fox D (2018) Re³: Real-time recurrent regression networks for visual tracking of generic objects. *IEEE Robotics and Automation Letters* 3(2): 788–795.
- Haghighi I, Jones A, Kong Z, Bartocci E, Grosu R and Belta C (2015) Spatel: a novel spatial-temporal logic and its applications to networked systems. In: *Proceedings* of the 18th International Conference on Hybrid Systems: Computation and Control. pp. 189–198.
- Havelund K, Peled D and Ulus3 D (2020) First-order temporal logic monitoring with bdds 56.
- He K, Lahijanian M, Kavraki LE and Vardi MY (2015) Towards manipulation planning with temporal logic specifications. In: 2015 IEEE international conference on robotics and automation (ICRA). IEEE, pp. 346–352.
- He K, Lahijanian M, Kavraki LE and Vardi MY (2018) Automated abstraction of manipulation domains for costbased reactive synthesis. *IEEE Robotics and Automation Letters* 4(2): 285–292.
- Hekmatnejad M, Yaghoubi S, Dokhanchi A, Amor HB, Shrivastava A, Karam L and Fainekos G (2019) Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic. In: 17th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE).
- Huth M and Ryan M (2004) *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press.
- Kim E, Shenoy J, Junges S, Fremont DJ, Sangiovanni-Vincentelli A and Seshia SA (2022) Querying labelled data with scenario programs for sim-to-real validation. In: ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS). pp. 34–45.
- Kontchakov R, Kurucz A, Wolter F and Zakharyaschev M (2007) Handbook of spatial logics: Spatial logic + temporal logic = ? Springer, pp. 497–564.
- Koymans R (1990) Specifying real-time properties with metric temporal logic. *Real-Time Systems* 2(4): 255–299.
- Kress-Gazit H and Pappas GJ (2010) Automatic synthesis of robot controllers for tasks with locative prepositions. In: *IEEE International Conference on Robotics and Automation* (*ICRA*). pp. 3215–3220.
- Lee TB (2018) Report: Software bug led to death in uber's self-driving crash. *Ars Technica* May 07.
- Li T, Liu J, Sun H, Chen X, Yin L, Mao X and Sun J (2021) Runtime verification of spatio-temporal specification language 21(26): 2392–2406.
- Linker S and Hilscher M (2013) Proof theory of a multi-lane spatial logic. In: *International Conference on Theoretical Aspects of Computing (ICTAC)*, *LNCS*, volume 8049. Springer, pp. 231–248.
- Liu Z, Jiang M and Lin H (2020) A graph-based spatial temporal logic for knowledge representation and automated reasoning in cognitive robots. arXiv preprint arXiv:2001.07205

Loos SM, Platzer A and Nistor L (2011) Adaptive cruise control: Hybrid, distributed, and now formally verified. In: *Formal Methods*, *LNCS*, volume 6664. Springer, pp. 42–56.

- Ma M, Bartocci E, Lifland E, Stankovic J and Feng L (2020) SaSTL: Spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: *ACM/IEEE 11th International Conference on Cyber-Physical Systems* (*ICCPS*). pp. 51–62.
- Maler O and Nickovic D (2004) Monitoring temporal properties of continuous signals. In: *Proceedings of FORMATS-FTRTFT, LNCS*, volume 3253. pp. 152–166.
- Mallick S, Ghosal S, Balakrishnan A and Deshmukh J (2023) Safety monitoring for pedestrian detection in adverse conditions. In: 23rd International Conference on Runtime Verification (RV).
- Manna Z and Pnueli A (1992) *The Temporal Logic of Reactive* and Concurrent Systems — Specification. Springer.
- Markey N and Raskin JF (2006) Model checking restricted sets of timed paths. *Theoretical Computer Science* 358: 273–292.
- Mehdipour N, Althoff M, Tebbens RD and Belta C (2023) Formal methods to comply with rules of the road in autonomous driving: State of the art and grand challenges 152: 110692.
- Motional (2019) nuScenes dataset. URL https://www. nuscenes.org/nuscenes. Accessed: 2020-11-14.
- Nenzi L, Bortolussi L, Ciancia V, Loreti M and Massink M (2015) Qualitative and quantitative monitoring of spatiotemporal properties. In: *Runtime Verification*. Springer, pp. 21–37.
- Pan J, Chou G and Berenson D (2023) Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Perugini S (2021) Programming languages: Concepts and implementation. Jones & Bartlett Learning.
- Qin B, Chong ZJ, Soh SH, Bandyopadhyay T, Ang MH, Frazzoli E and Rus D (2016) A spatial-temporal approach for moving object recognition with 2d lidar. In: *Experimental Robotics*. Springer, pp. 807–820.
- Richter SR, Hayder Z and Koltun V (2017) Playing for benchmarks. In: *IEEE International Conference on Computer Vision, ICCV.* pp. 2232–2241.
- Rizaldi A, Keinholz J, Huber M, Feldle J, Immler F, Althoff M, Hilgendorf E and Nipkow T (2017) Formalising and monitoring traffic rules for autonomous vehicles in isabelle/hol. In: *Integrated Formal Methods*. Springer, pp. 50–66.
- Schwarting W, Alonso-Mora J and Rus D (2018) Planning and decision-making for autonomous vehicles. *Annual Review* of Control, Robotics, and Autonomous Systems 1: 187–210.
- Shneier M (1981) Calculations of geometric properties using quadtrees. *Computer Graphics and Image Processing* 16(3): 296–302.
- Sipser M (2006) *Introduction to the theory of computation*. 2nd edition. Course Technology.
- STPL (2023) Spatio-Temporal Perception Logic (STPL) Offline Monitoring Tools. https://gitlab.com/vnv-tools/STPL.
- Summers-Stay D, Cassidy T and Voss C (2014) Joint navigation in commander/robot teams: Dialog & task performance

when vision is bandwidth-limited. In: *Third Workshop on Vision and Language*.

- Sun H, Ang MH and Rus D (2019) A convolutional network for joint deraining and dehazing from a single image for autonomous driving in rain. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 962–969.
- Templeton B (2020) Tesla in taiwan crashes directly into overturned truck, ignores pedestrian, with autopilot on. *Forbes* June 2.
- Tuncali CE, Fainekos G, Prokhorov D, Ito H and Kapinski J (2020) Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Transactions on Intelligent Vehicles* 5: 265–280. DOI:10.1109/ TIV.2019.2955903.
- van Benthem J and Bezhanishvili G (2007) Modal logics of space. *Handbook of Spatial Logics* : 217–298.
- Wu B, Iandola F, Jin PH and Keutzer K (2017) Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. pp. 129–137.
- Yadav P and Curry E (2019) Vidcep: Complex event processing framework to detect spatiotemporal patterns in video streams. In: *IEEE International conference on big data*. pp. 2513–2522.
- Yu F, Chen H, Wang X, Xian W, Chen Y, Liu F, Madhavan V and Darrell T (2020) Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 2636–2645.
- Yurtsever E, Lambert J, Carballo A and Takeda K (2020) A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access* 8: 58443–58469.

APPENDIX A APPENDIX: STPL FUTURE SYNTAX

The following definition introduces a future fragment of the introduced STPL syntax in Def. 4.1. Here, we restrict the grammar by including rules that enforce a formula to be an *Almost Arbitrarily Nesting Formula* as in Def. 4.3. Notice that in the following, the grammar rules force the expressions to be indexed to track the level of nesting in quantifier operators.

Definition A.1 (STPL AAN Syntax for Discrete-Time Signal): Let V_x and V_o be sets of time variables and object variables, respectively. Let x be a vector of time variables, i.e., $x = [x_0, \ldots, x_{n-1}]^T$, and id be a vector of object variables, i.e., $id = [id_0, \ldots, id_{m-1}]^T$, and \mathcal{I} be any non-empty interval of $\mathbb{R}_{\geq 0}$ over time. The syntax for Spatio-Temporal Perception Logic (STPL) formulas is provided by the following grammar:

$$\begin{split} \Phi_{i,j} &\coloneqq \exists i d_i @x_i . \Phi_i^{f,q} \mid x_i . \Phi_i^f \mid \exists i d_i . \Phi_{i,j}^q \\ & \top \mid \neg \Phi_{i,j} \mid \Phi_{i,j} \lor \Phi_{i,j} \mid \bigcirc \Phi_{i,j} \mid \Phi_{i,j} \ \mathcal{U} \ \Phi_{i,j} \mid \\ & \bigcirc_{\mathcal{I}} \Phi_{i,j} \mid \Phi_{i,j} \ \mathcal{U}_{\mathcal{I}} \ \Phi_{i,j} \mid \bigcap_{\mathcal{I}} \Phi_{i,j} \mid \Phi_{i,j} \ \tilde{\mathcal{U}}_{\mathcal{I}} \ \Phi_{i,j} \end{split}$$

$$\begin{split} \Phi_i^{f,q} &\coloneqq \Phi_i^f \mid \Phi_{i,i}^q \mid \\ & \mathsf{T} \mid \neg \Phi_i^{f,q} \mid \Phi_i^{f,q} \lor \Phi_i^{f,q} \mid \bigcirc \Phi_i^{f,q} \mid \Phi_i^{f,q} \ \mathcal{U} \ \Phi_i^{f,q} \mid \\ & \bigcirc_{\mathcal{I}} \Phi_i^{f,q} \mid \Phi_i^{f,q} \ \mathcal{U}_{\mathcal{I}} \ \Phi_i^{f,q} \mid \bigcirc_{\mathcal{I}} \Phi_i^{f,q} \mid \bigoplus_i^{f,q} \ \mathcal{U}_{\mathcal{I}} \ \Phi_i^{f,q} \end{split}$$

$$\begin{split} \Phi_i^f &\coloneqq \tau - x_i > t \mid \mathcal{F} - x_i > n \mid \Phi_{i+1,i} \mid \\ &\top \mid \neg \Phi_i^f \mid \Phi_i^f \lor \Phi_i^f \mid \bigcirc \Phi_i^f \mid \Phi_i^f \ \mathcal{U} \ \Phi_i^f \mid \\ &\bigcirc_{\mathcal{I}} \Phi_i^f \mid \Phi_i^f \ \mathcal{U}_{\mathcal{I}} \ \Phi_i^f \mid \bigcirc_{\mathcal{I}} \Phi_i^f \mid \bigcirc_{\mathcal{I}} \Phi_i^f \mid \Phi_i^f \ \mathcal{U}_{\mathcal{I}} \ \Phi_i^f \end{split}$$

$$\begin{split} \Phi_{i,j}^{q} &\coloneqq C(\Lambda_{i,j}) = c \mid C(\Lambda_{i,j}) = C(\Lambda_{i,j}) \mid P(\Lambda_{i,j}) \ge r \\ &\mid P(\Lambda_{i,j}) \ge r \times P(\Lambda_{i,j}) \mid \Lambda_{i,j} = \Lambda_{i,j} \mid \Phi_{i+1,j} \mid \\ &\top \mid \neg \Phi_{i,j}^{q} \mid \Phi_{i,j}^{q} \lor \Phi_{i,j}^{q} \mid \bigcirc \Phi_{i,j}^{q} \mid \Phi_{i,j}^{q} \mathrel{\mathcal{U}} \Phi_{i,j}^{q} \mid \\ &\bigcirc_{\mathcal{I}} \Phi_{i,j}^{q} \mid \Phi_{i,j}^{q} \mathrel{\mathcal{U}}_{\mathcal{I}} \Phi_{i,j}^{q} \mid \bigcap_{\mathcal{I}} \Phi_{i,j}^{q} \mid \Phi_{i,j}^{q} \mathrel{\mathcal{U}}_{\mathcal{I}} \Phi_{i,j}^{q} \mid \\ & \exists \Omega_{i,j} \mid \Theta_{i,j} \mid \Pi_{i,j} \end{split}$$

 $\Lambda_{i,j} \coloneqq id_j \mid id_{j+1} \mid \dots \mid id_i$

$$\begin{split} \Omega_{i,j} &\coloneqq \sigma(\Lambda_{i,j}) \mid \varnothing \mid \mathbb{U} \mid \overline{\Omega}_{i,j} \mid \Omega_{i,j} \sqcap \Omega_{i,j} \mid \mathbf{I}\Omega_{i,j} \mid \\ \Omega_{i,j} \; \mathcal{U}_{s} \; \Omega_{i,j} \mid \diamondsuit_{s} \Omega_{i,j} \mid \Box_{s} \; \Omega_{i,j} \mid \bigcirc_{s} \Omega_{i,j} \mid \\ \Omega_{i,j} \; \mathcal{U}_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \diamondsuit_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \Box_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \bigcirc_{\mathcal{I}}^{s} \Omega_{i,j} \mid \\ \Omega_{i,j} \; \widetilde{\mathcal{U}}_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \diamondsuit_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \Box_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \bigcirc_{\mathcal{I}}^{s} \; \Omega_{i,j} \mid \\ \end{split}$$

 $\Pi_{i,j} ::= Area(\Omega_{i,j}) \ge r \mid Area(\Omega_{i,j}) \ge r \times Area(\Omega_{i,j})$

$$\begin{split} \Theta_{i,j} &\coloneqq Dist(\Lambda_{i,j}, \operatorname{CRT}, \Lambda_{i,j}, \operatorname{CRT}) \geq r \mid \\ Lat(\Lambda_{i,j}, \operatorname{CRT}) \geq r \mid Lon(\Lambda_{i,j}, \operatorname{CRT}) \geq r \mid \\ Lat(\Lambda_{i,j}, \operatorname{CRT}) \geq r \times Lat(\Lambda_{i,j}, \operatorname{CRT}) \mid \\ Lon(\Lambda_{i,j}, \operatorname{CRT}) \geq r \times Lon(\Lambda_{i,j}, \operatorname{CRT}) \mid \\ Lat(\Lambda_{i,j}, \operatorname{CRT}) \geq r \times Lon(\Lambda_{i,j}, \operatorname{CRT}) \mid \\ Area(\Lambda_{i,j}) \geq r \mid Area(\Lambda_{i,j}) \geq r \times Area(\Lambda_{i,j}) \end{split}$$

CRT := LM | RM | TM | BM | CT

where $i \ge 0$, and the grammar starts from $\Phi_{0,0}$.

The time and frame constraints of STPL are represented in the form of $\tau - x > r$ and $\mathcal{F} - x > n$, respectively. The freeze time quantifier $x.\phi$ assigns the current frame *i* to time variable *x* before processing the subformula ϕ . The *Existential* quantifier is denoted as \exists , and the *Universal* quantifier is denoted as \forall . The following syntactic equivalences hold for the STPL formulas ψ and ϕ using syntactic manipulation. $\forall \{id\}@x.\phi \equiv \neg(\exists\{id\}@x.\neg\phi), \psi \land \phi \equiv \neg(\neg\psi \lor \neg\phi), \bot \equiv \neg\top$ (False), $\psi \rightarrow \phi \equiv \neg\psi \lor \phi$ (ψ Implies ϕ), $\phi \mathcal{R} \psi \equiv \neg(\neg\phi \mathcal{U} \neg \psi)$ (ϕ releases ψ), $\phi \mathcal{R} \psi \equiv \phi \mathcal{R} (\phi \lor \psi)$ (ϕ non-strictly releases ψ), $\Diamond \psi \equiv \top \mathcal{U} \psi$ (Eventually ψ), $\Box \psi \equiv \neg \Diamond \neg \psi$ (Always ψ). All the other operators with $\tilde{\neg}$ on them are with frame intervals, that is in $\tilde{\Box}_{\mathcal{I}}^{s}, \tilde{\bigtriangledown}_{\mathcal{I}}^{s}, \tilde{\mathcal{U}}_{\mathcal{I}}^{s}, \tilde{\mathcal{O}}_{\mathcal{I}}^{s}, \tilde{\mathcal{U}}_{\mathcal{I}}$, and $\tilde{\bigcirc}_{\mathcal{I}}$ the interval \mathcal{I} is over frame interval.

For parsing a formula using the above grammar, there are two production rules Φ_i^f and $\Phi_{i,j}^q$ in which we can use the initial production rule after increasing the index *i* (i.e., $\Phi_{i+1,j}$). The index i is to force scope for the use of freeze time variables. For example, if in the scope of a variant-quantifier operator we use x_0 , then the index will increases to 1 to avoid use of x_0 in the scope of the next variant-quantifier operator. The index j is used as a pointer to each quantifier operator to track the scope of object variables. For example, in the formula $\varphi \equiv \exists i d_0. \Box (\exists i d_1. \exists i d_2. (\phi_1) \lor \phi_2)$, we have i = 2 and j = 0 while parsing the subformula ϕ_1 , whereas, in ϕ_2 , we have i = 0 and j = 0. Thus any function in ϕ_1 with object variables in it will use the production rule $\Lambda_{2,0}$. Thus, the allowed object variables in ϕ_1 are id_0, id_1 and id_2 . However, while parsing the subformula ϕ_2 , we use $\Lambda_{0,0}$ in which the only allowed object variable is id_0 .

APPENDIX B

APPENDIX: COMPLEXITY ANALYSIS OF STE FORMULAS

Here the assumption is that we use the linked-list data structure to represent a spatial term \mathcal{T} as a union of a finite number of unique subsets. We can compute $V(\tau_1 \ \mathcal{U}_{\mathcal{T}}^s \ \tau_2, \mathcal{D}, t, \tau, \epsilon, \zeta)$ recursively as $\mathbf{V}(\tau_2, \mathcal{D}, t, \tau, \epsilon, \zeta) \cup (\mathbf{V}(\tau_1, \mathcal{D}, t, \tau, \epsilon, \zeta) \cap$ $\mathbf{V}((\tau_1 \ \mathcal{U}^s_{\mathcal{T}} \ \tau_2), \mathcal{D}, t+1, \tau, \epsilon, \zeta))$. For each until formula in each row of Table VI (starting from the row l = 1), for each time step t, we use the recursive evaluation function to calculate the maximum number of bounding boxes as a result of computing the formula. The maximum number of bounding boxes that can be produced by $\tau_1 \cup \tau_2$ is equal to the total number of boxes in the two spatial terms (i.e., $|\tau_1| + |\tau_2|$). Additionally, the maximum number of bounding boxes that can be produced by $\tau_1 \cap \tau_2$ is equal to the product of number of boxes in the two spatial terms (i.e., $|\tau_1| \times |\tau_2|$). Consequently, the maximum number of bounding boxes that can be produced at each time step t for the above until formula is $|\mathbf{V}(\tau_2, \mathcal{D}, t, \tau, \epsilon, \zeta)| +$ $|\mathbf{V}(\tau_1, \mathcal{D}, t, \tau, \epsilon, \zeta)| \times |\mathbf{V}(\tau_1 \ \mathcal{U}_{\mathcal{I}}^s \ \tau_2, \mathcal{D}, t+1, \tau, \epsilon, \zeta)|.$

A. Formulas with Exponential Time/Space Complexities

As it is stated in the first and second rows in Table VI, the number of needed operations grow as in arithmetic sequences.

l	t_n	t_{n-1}	t_{n-2}	t_{n-3}	t_{n-4}	
0	1	1	1	1	1	
1	1	1(1+1) = 2	1(1+1(1+1)) = 3	1(1+ 1(1+ 1(1+1))) = 4	1(1+) 1(1+)(1+) 1(1+))) = 5	
2	1	$2(1+1) = 4, 2 \le 4 \le 3!$	$3(1+2(1+1)) = 15, 2^2 \le 15 \le 4!$	$\begin{array}{r} 4(1+ \\ 3(1+ \\ 2(1+1))) \\ = 64, \\ 2^3 \leq \\ 64 \leq 5! \end{array}$	$5(1+ 4(1+ 3(1+ 2(1+1)))) = 325, 2^4 \le 325 \le 6!$	
3	1	2(1+1) (1+1) = 8	3(1+2(1+1))(1+2(1+1)(1+1))= 135	$\begin{array}{c} 4(1+\\ 3(1+\\ 2(1+1)))\\ (1+\\ 3(1+\\ 2(1+1))\\ (1+\\ 2(1+1)\\ (1+1)))\\ = 8,704 \end{array}$	5(1+ 4(1+ 3(1+ 2(1+1)))) (1+ 4(1+ 3(1+ 2(1+1))) (1+ 3(1+ 2(1+1)) (1+ 2(1+1)) (1+ 2(1+1)) (1+2(1+1))) = 2,829,125	

TABLE VI: DP-based complexity analysis for spatio-temporal until operator with different levels of nesting. At l = 0 we have $\tau = \sigma(Id)$; At l = 1: we have $\tau \ \mathcal{U}_{\mathcal{I}}^s \tau$, and $\tau \ \mathcal{U}_{\mathcal{I}}^s \tau$; At l = 2: we have $(\tau \ \mathcal{U}_{\mathcal{I}}^s \tau) \ \mathcal{U}_{\mathcal{I}}^s (\tau \ \mathcal{U}_{\mathcal{I}}^s \tau)$; Finally, at l = 3: we have $((\tau \ \mathcal{U}_{\mathcal{I}}^s \tau) \ \mathcal{U}_{\mathcal{I}}^s (\tau \ \mathcal{U}_{\mathcal{I}}^s \tau)) \ \mathcal{U}_{\mathcal{I}}^s ((\tau \ \mathcal{U}_{\mathcal{I}}^s \tau) \ \mathcal{U}_{\mathcal{I}}^s (\tau \ \mathcal{U}_{\mathcal{I}}^s \tau))$.

Thus, the time complexity which is the summation of numbers in the rows, are linear and polynomial functions of the number of the time steps for the first and second rows, respectively. Moreover, the space complexity for the first and the second rows are constant and linear functions of number of the time steps, respectively.

In the following, we calculate an upper bound and a lower bound for the maximum number of bounding boxes that can be produced for the third row (level 2) of the until operator. We use the function $f_2(t)$ to denote the maximum number of bounding boxes that are produced at the time step t for the until formula $(\tau \ U_{\mathcal{I}}^s \tau) \ U_{\mathcal{I}}^s (\tau \ U_{\mathcal{I}}^s \tau)$.

$$\sum_{t=1}^{n} f_2(t) =$$

$$1 + 2(1+1) + 3(1 + 2(1+1)) +$$

$$4(1 + 3(1 + 2(1+1))) + 5(1 + 4(1 + 3(1 + 2(1+1)))) + \dots$$

$$+ n(1 + (n-1)(1 + (n-2)(\dots 1 + 2(1+1)))\dots) \quad (21)$$

We repetitively use the inequality (a+1)b > a(1+b) for b > a to derive the following inequality from the above equation

$$\sum_{t=1}^{n} f_2(t) < 1 + 3! + 4! + 5! + 6! + \dots + (n+1)!$$
 (22)

where $n \ge 2$. Therefore, we have

$$\sum_{t=1}^{n} f_2(t) < n \times (n+1)! < (n+2)!$$
(23)

Next, we calculate a lower bound for the maximum number of bounding boxes that can be produced for the level 2 of the until operator. We repetitively use the inequality $2^{(b+1)} <$ 32

 $a(1+2^b)$ for $a \ge 2$ to derive the following inequality from Eq. (21)

$$\sum_{t=1}^{n} f_2(t) > 1 + 2^1 + 2^2 + 2^3 + 2^4 + \dots + 2^{(n-1)}$$
(24)

where $n \ge 2$. Therefore, we have

$$\sum_{t=1}^{n} f_2(t) > 2^n \tag{25}$$

This time inequality suggests that the time/space complexity for any formulas with more than one level of nesting can be exponential.

B. Best Complexity for the Worst Formulas

We can repeat the above method to calculate a lower bound for each row of the table by using the inequality $a^{r+1} < a^r(1+b)$ for b > a in each summation of the elements of rows to derive the below inequality

$$\sum_{t=1}^{n} f_0(t) + \sum_{t=1}^{n} f_1(t) + \dots + \sum_{t=1}^{n} f_l(t) >$$

$$n + \frac{n(n+1)}{2} + \frac{2^n - 1}{2 - 1} + \frac{3^n - 1}{3 - 1} + \dots + \frac{l^n - 1}{l - 1} >$$

$$n + \frac{n(n+1)}{2} + 2^{(n-1)} + 3^{(n-1)} + \dots + l^{(n-1)}$$

$$- \left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{l - 1}\right) >$$

$$> 2^{(n-1)} + 3^{(n-1)} + \dots + l^{(n-1)}$$

where $n \ge 2$. Therefore, we have

$$\sum_{t=1}^{n} f_0(t) + \sum_{t=1}^{n} f_1(t) + \dots + \sum_{t=1}^{n} f_l(t) > l^{(n-1)}$$
(26)

This concludes the complexity of the algorithm to be $\Omega(|\varphi_s|^{(|\hat{\rho}|-1)}).$