Formalizing and evaluating requirements of perception systems for automated vehicles using spatio-temporal perception logic

Mohammad Hekmatnejad1, Bardh Hoxha2, Jyotirmoy V. Deshmukh3, Yezhou Yang1 and Georgios Fainekos2

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.

Keywords

Programming environment, recognition, sensing and perception computer vision, simulation, interfaces and virtual reality, visual tracking

Received 23 August 2021; Revised 21 November 2023; Accepted 5 December 2023

Senior Editor: Luca Carlone
Associate Editor: Guillermo Gallego

1. 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 s, 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 Figure 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

---

1School of Computing and Augmented Intelligence (formerly CIDSE), Arizona State University, Tempe, AZ, USA
2Research & Development, Toyota Motor North America, Ann Arbor, MI, USA
3Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Corresponding author:
Mohammad Hekmatnejad, School of Computing and Augmented Intelligence (formerly CIDSE), Arizona State University, 699 S Mill Ave, Tempe, AZ 85281, USA.
Email: mhekmatn@asu.edu
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 four out of five frames (or any other desired frequency). The first challenge for an automated testing and 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 search-based 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 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, for example, Basset et al. (2020), or to assess the real-to-sim gap in testing, for example, 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.

Figure 1. A series of image frames taken from KITTI (Geiger et al. (2013a)) augmented with the data about classified data-objects using SqueezeDet (Wu et al. (2017)). Here, \( \tau \) is a function that maps each frame \( f \) to its captured time.
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, for example, 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 $S4_4$ 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 three frames and for at least 10% of the area.**

which is satisfiable for the image frames in Figure 1.

Beyond expressing requirements on the functional performance of perception systems, STPL can be used to compare 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+ $S4_4$), 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 4 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); Elgyut et al. (2018); Ghorbel and Prabhu (2022)).

Another source of computational complexity in STPL is the use of object/region quantification and set operations, that is, 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 $S4_4$ formula depends on the set representation and the dimensionality of the space (e.g., polytopes Baotic (2009), orthogonal polyhedra Bournez et al. (1999), quad trees or octrees Aluru (2018), etc.).

In this paper, we introduce an offline monitoring algorithm for STPL in Section 6, and we study its computational complexity in Section 6.5. In Section 6.6, 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 7 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 as follows:

1. 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 4).
2. We present examples of STPL specifications of increasing complexity as a tutorial (Section 5), and we demonstrate that STPL can find label inconsistencies in publicly available AV datasets.
3. We introduce an offline monitoring algorithm (Section 6), and we propose to use different set representations for efficient computations (Section 6.5).
4. 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 6.5).
5. We experimentally study offline monitoring runtimes for a range of specifications and present the trade-offs in using the different STPL operators (Section 7).
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 (LLM) 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 (LTL).

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. (2015, 2018)).

2. 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).

2.1. 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.

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 \(i\), \(D(i)\) is the set of data-objects for frame \(i\), where \(i\) is the order of the frame in the data-object stream \(D\). We assume that a data stream is provided by a perception system, in which a function \(R\) retrieves data attributes for an identified object. A data-attribute 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 \(R(D(i), id).Prob\), we refer to the probability that an object identified by \(id\) in the \(i\)’th frame is correctly classified,
- by \(R(D(i), id).PC\), 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

---

**Figure 2.** Let \(X = X_1 \cup X_2\) with \(CX = X\). Left: the whole rectangle represents the universe (colored gray) \(U\), the regions \(X_1\) and \(X_2\) without the dashed lines (colored light blue) represent the interior \(I_X\) of the set \(X\), the dashed lines (colored darker blue) represent the boundary \(CX - I_X\) of the set \(X\), and the remaining region represents \(U - X\). Right: the tightest bounding box around set \(X\) that includes the four unique elements of the set as in Def. 2.4.
Throughout the paper, we will be using the time attribute for the frame stamp, that is, 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 \(D\) is represented in Figure 1, then the below equalities hold:

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

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

### 2.2. 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 \(S_4\) logic which are adopted from Gabelaia et al. (2005) and Kontchakov et al. (2007). Later, we build our \(MIL \times S_4\) definition on this notation.

**Definition 2.1.** Topological Space. A topological space is a pair \(T = (\mathcal{U}, \mathcal{I})\) in which \(\mathcal{U}\) is the universe of the space, and \(\mathcal{I}\) is the interior operator on \(\mathcal{U}\). The universe is a non-empty set, and \(\mathcal{I}\) satisfies the standard Kuratowski axioms where \(X, Y \subseteq \mathcal{U}\):

\[
\mathcal{I}(X \cap Y) = \mathcal{I}X \cap \mathcal{I}Y, \quad \mathcal{I}X = \mathcal{I}X \subseteq X, \quad \text{and} \quad \mathcal{I} \cup \mathcal{U} = \mathcal{U}
\]

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

- \(\mathcal{I}X\) is the interior of a set \(X\),
- \(\mathcal{C}X\) is the closure of a set \(X\),
- \(X\) is called open if \(X = \mathcal{I}X\),
- \(X\) is called closed if \(X = \mathcal{C}X\),
- If \(X\) is an open set then its complement \(\mathcal{C}X = \mathcal{U} - X\) is a closed set and vice versa,
- For any set \(X \subseteq \mathcal{U}\), its boundary is defined as \(\mathcal{C}X - \mathcal{I}X(X\text{'s complement have the same boundary})\).

### 2.3. 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 \((\mathbb{U}, \mathcal{I})\) 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 two-dimensional (2D) TO topological space \(T\), \(p = (x, y) \in \mathbb{U}\) 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:

\[
(x_1, y_1) \lt_{2D} (x_2, y_2) \iff y_1 < y_2 \lor (y_1 = y_2 \land x_1 \leq x_2).
\]

Note that even though Def. 2.3 considers the coordinates over the reals, that is, \(\mathbb{R}_{\geq 0}\), in practice, the image space is defined over the integers, that is, \(\mathbb{N}^2\) (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 \(x\) - axis and 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 Figure 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 \(U \subseteq \mathcal{U}\) from a 2D TO topological space \(T = (\mathbb{U}, \mathcal{I})\), we define the top-most, bottom-most, left-most, right-most, and center-point functions by \(\mathcal{TM}, \mathcal{BM}, \mathcal{LM}, \mathcal{RM}, \mathcal{CT} : 2^\mathbb{U} \rightarrow \mathbb{U}\), respectively, such that

- \(\mathcal{TM}(U) = (x_p, y_p) \in \mathbb{U}\) s.t.
  \[
  \forall s = (x_s, y_s) \in \mathbb{U}, y_p < y_s \lor (y_p = y_s \land x_p \geq x_s)
  \]
- \(\mathcal{BM}(U) = (x_p, y_p) \in \mathbb{U}\) s.t.
  \[
  \forall s = (x_s, y_s) \in \mathbb{U}, y_p > y_s \lor (y_p = y_s \land x_p \leq x_s)
  \]
- \(\mathcal{LM}(U) = (x_p, y_p) \in \mathbb{U}\) s.t.
  \[
  \forall s = (x_s, y_s) \in \mathbb{U}, x_p < x_s \lor (x_p = x_s \land y_p \leq y_s)
  \]
- \(\mathcal{RM}(U) = (x_p, y_p) \in \mathbb{U}\) s.t.
  \[
  \forall s = (x_s, y_s) \in \mathbb{U}, x_p > x_s \lor (x_p = x_s \land y_p \geq y_s)
  \]

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

In Def. 2.4, the notation \(2^A\) for a set \(A\) denotes the set of all subsets of \(A\), that is, 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 \(\mathcal{LM}(X), \mathcal{TM}(X), \mathcal{RM}(X), \text{and} \mathcal{BM}(X)\) 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.
2.4. Spatio-temporal logic MTL × S4u

Next, we introduce the logic MTL × S4u over discrete-time semantics. MTL × S4u is a combination of Metric Temporal Logic (MTL) (Koymans (1990)) with the S4u logic of topological spaces. From another perspective, it is an extension to the logic PTL × S4u (see Gabelaia et al. (2005)) by adding time/frame intervals into the spatio-temporal operators. Even though MTL × S4u 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 × S4u. Henceforth, the symbol p refers to a spatial proposition, that is, it represents a set in the topological space. In addition, the symbol T represents spatial expressions that evaluate to subsets of the topological space. We refer to T as a spatial term.

**Definition 2.5.** Syntax of MTL × S4u as a temporal logic of topological spaces. Let Π be a finite set of spatial propositions over \( T = (\cup, I) \), then a formula \( \varphi \) of MTL × S4u can be defined as follows:

\[
T ::= p \mid \overline{T} \mid T \cap T \mid I \mid T \mid T_u T \mid x_i T \mid x_i T
\]

\[
\varphi ::= \exists T \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi_u \varphi \mid x_i \varphi
\]

where \( p \in \Pi \) is a spatial proposition.

In the above definition, the grammar for MTL × S4u consists of two sets of production rules \( T \) and \( \varphi \). The spatial production rule \( T \) in Def. 2.5 contains a mix of spatial \((\cap, I)\) and spatio-temporal \((U, x_i)\) operators. Here, \( \overline{T} \) is the spatial complement of the spatial term \( T \), \( \cap \) is the spatial intersection operator, and \( I \) is the spatial interior operator. The \( (U, x_i) \) operators are the spatio-temporal until and the spatio-temporal next, respectively. Here, the subscript \( I \) 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 \( I = (0, +\infty) \), or remove the subscript \( I \) from the notation. Intuitively, the spatio-temporal until and next operators introduce spatial operations over time. For instance, the expression \( p_1 U_{1,3} p_2 \) 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 spatio-temporal operators \( U \) and \( x_i \), 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 \( \varphi \) in Def. 2.5, the spatially exists \( \exists \) checks if the spatial term following it evaluates to a non-empty set. That is, \( \exists \) checks if there exist some points in the set represented by \( T \). In \( \varphi \), except for the spatially exists \( \exists \), the other operators are the same as in MTL. That is \( \neg, \land, U \), and \( x_i \) 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 × S4u, the expression \( \exists p_1 U_{0.5,2.5} \exists p_2 \) is true if the set represented by \( p_2 \) is non-empty 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 × S4u. The definition uses the spatial semantics of S4u while extending the temporal fragment (PTL) with time constraints over finite traces as in MTL. The semantics are defined over a data-object stream \( D \). However, for consistency with PTL × S4u, we will assume the existence of a spatio-temporal valuation function \( \Pi : \Pi \times \mathbb{N} \rightarrow 2^U \) that associates with every proposition \( p \) and time frame \( i \) a subset of the topological space. In the definition of STPL in Section 4, the sets \( \Pi(p, i) \) will correspond to identified objects in the environment, that is, bounding boxes, bounding rectangles, or even regions in semantic segmentation. In this section, the sets \( \Pi(p, i) \) 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 \( Q = (T, \Pi, D, \tau) \), where \( T = (\cup, I) \) is a TO topological space, \( \Pi \) is a spatio-temporal valuation function, \( D \) is a data-object stream of size \( |D| \), \( \tau : \mathbb{N} \rightarrow \mathbb{R}^+ \) maps frame numbers to their physical times, and \( I \) is any non-empty interval of \( \mathbb{R}_{\geq 0} \) over time.

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

\[
V(p, D, i, \tau) := U(p, i) \\
V(T_1 \cup C2 T_2, D, i, \tau) := \bigcup_{i' \in \mathbb{N} | \tau(i') + 1 = \tau(i) + \tau} V(T_1, D, i', \tau) \cap V(T_2, D, i, \tau)
\]

where \( t + \tau = \{ i' \mid \exists t' \in \mathbb{T}, t' = t + i' \} \).

The valuation function \( V \) definition is straightforward for the spatial operations, that is, \( , \cap, \cup, I \). \( V \) is just applying the corresponding set theoretic operations, that is, \( , \cap, \cup \). The more interesting cases are the spatio-temporal operators. The spatial eventually operator \( \exists T \) first checks if the next sample \( (i+1) \) satisfies the timing constraints, that is, \( \tau(i) + \tau(i + 1) \in \tau(i) + \tau \), and if so, it returns the set that \( T \) 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 one of the commonly used derived operators:

- The spatially for all operator \( \forall \) checks if the spatial expression \( T \) represents a set which is the same as the universe: \( \forall T = \neg \exists T \)
- The spatial union operator: \( \exists T \cup \exists T = \exists T \cap \exists T \)
- The spatial closure operator: \( \exists T = \neg \exists T \)
- The spatial eventually operator: \( \exists T = \exists T \cup \exists T \)
- The spatial always operator: \( \exists T = \exists T \cup \exists T \)

Notice that in the definition of the spatial eventually operator, we used the universe set \( U \) as a terminal even though the syntax in Def. 2.5 does not explicitly allow for that. The universe (\( U \)) and the empty set (\( \emptyset \)) can be defined as derived spatial expressions, that is, \( U = \exists \) and \( \emptyset = \exists \). Therefore, if we replace \( U \) in the definition of \( V \) for \( \exists T \), we can observe that \( \exists T \) corresponds to the spatial union of the expression \( T \) over the time interval \( I \).

**Example 2.2.** A simple example is presented in Figure 3 for a data stream with four frames. The spatial expression \( \exists T \) corresponds to the union of all the sets represented by \( p \) over time (gray set in Figure 3) since there are no timing constraints. On the other hand, the spatial expression \( \exists T(0, 1) \) 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 \( \exists T(0, 1) \) is represented as a black box in Figure 3.

Given the definition of the valuation function \( V \) for QTMM, we can proceed to define the semantics of MTL \( \times \) S4\( _u \). Recall that the valuation of spatial expressions returns sets from some topological space. On the other hand, MTL \( \times \) S4\( _u \) formulas are interpreted over Boolean values True (\( \top \)) / False (\( \bot \)), that is, the formulas are satisfied or are not satisfied. To evaluate MTL \( \times \) S4\( _u \) formulas, we use a valuation function \( [ \phi ]_V \) which takes as an input an MTL \( \times \) S4\( _u \) formula \( \phi \), a data stream \( D \), a sample \( i \), and a timestamp function \( \tau \), and returns True (\( \top \)) or False (\( \bot \)).

**Definition 2.7.** MTL \( \times \) S4\( _u \) semantics. Given an MTL \( \times \) S4\( _u \) formula \( \phi \), a QTMM \( Q = \langle T, U, D, \tau \rangle \), and a frame \( i \in \mathbb{N} \), the semantics of \( \phi \) are defined recursively as follows:

\[
[\top](D, i, \tau) := \top \\
[\bot](D, i, \tau) := \bot \\
[\neg \phi](D, i, \tau) := \neg \phi \\
[\phi_1 \lor \phi_2](D, i, \tau) := [\phi_1](D, i, \tau) \lor [\phi_2](D, i, \tau) \\
[\phi_1 \land \phi_2](D, i, \tau) := [\phi_1](D, i, \tau) \land [\phi_2](D, i, \tau) \\
[\phi_1 \land \phi_2](D, i, \tau) := \bigcup_{i \in \mathbb{N} | \tau(i) + \tau} \left( [\phi_2](D, i, \tau) \land [\phi_1](D, k, \tau) \right)
\]

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 \( \phi_1 \lor \phi_2 \equiv \neg (\neg \phi_1 \land \neg \phi_2) \), eventually as
\[ \square_I \phi \equiv \top \cup_I \phi, \text{ and always as } \square_I \neg \phi \equiv \neg \square_I \neg \phi. \text{ Finally, the constant true is defined as } \top \equiv \bigvee U. \]

**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, that is, in \( \square_{fi} \), \( \Diamond_{fi} \), \( \top_{fi} \), and \( \neg \square_{fi} \) the interval \( I \) is over frame interval. When reasoning over frames, \( \tau(i) \) equals \( i \), that is, \( \tau \) is the identity function.

**Example 2.3.** Revisiting Example 2.2, we can now introduce and explain some MTL \( \times S_{4_\mu} \) formulas. The formulas \( \square \Diamond p \) and \( \square_{fi} \top_{fi} p \) evaluate to true since the sets \( \Diamond p \) and \( \top_{fi} p \) are not empty. On the other hand, the formula \( \square \Box_{fi} p \) is false because the set that corresponds to \( \Box_{fi} p \) is empty (in Figure 3, there is no common subset for \( p \) across all frames). Notice that the MTL \( \times S_{4_\mu} \) formula \( \square \top_{fi} p \) is true since the set \( p \) is not empty at every frame. Finally, considering time stamps versus frames, the set \( \top_{fi} p \), which considers timing constraints, is the same as the set \( \Box_{fi} p \), which considers frame constraints.

### 3. Problem definition

Given a data stream \( D \) as defined in 2.1, 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.

#### 3.1. Assumptions

Given a data stream \( D \), 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.

### 3.2. Overall solution

We define syntax and semantics of STPL over sets of points in topological space, and quantifiers over objects. We build a monitoring algorithm over TPTL, MTL, and \( S_{4_\mu} \), 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 8), 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.

### 4. Spatio-temporal perception logic

In this section, we present the syntax and semantics of the STPL. Our proposed logic evaluates the geometric relations of objects in a data stream and their evolution over time. Theoretically, any set-based 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 data stream as annotations or attributes for each object or region identified by the perception system. For instance, \( R(D(i), id).CS \) may store a rectangular axis-aligned convex-polygon that is associated with an object \( id \) in the \( i \)th frame. Alternatively,

![Figure 4](image_url)  
*Figure 4.* An object data stream \( D \) 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.
4.1. STPL syntax

The syntax of STPL is based on the syntax of TQTL (Dokhanchi et al. (2018a)) and S4v (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 \( \sigma \) to map these objects to their corresponding sets. Hence, the syntax of spatial terms \( T \) in STPL is the same as in MTL \( \times S4v \), but the function symbols \( \sigma \) 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 data stream, for example, classes, probabilities, estimated velocities, or even compute some basic functions on object attributes, for example, 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 \( \Theta \).

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 data stream. Such functionality support is going to be among the goals of future software releases.

**Definition 4.1. STPL Syntax over Discrete-Time Signals.**

Let \( V_i \) and \( V_o \) be sets of time variables and object variables, respectively. Assume that \( x \in V_i \) is a time variable, \( id \in V_o \) is an object variable, \( I \) is any non-empty interval of \( \mathbb{R}_{\geq 0} \) over time. The syntax for STPL formulas is provided by the following grammar starting from the production rule \( \phi \):

The syntax for spatial terms is:

\[
T ::= \sigma(id) \mid \neg T \mid T \sqcap T \mid T \sqcup T \mid T^\prime \mid O_{\lambda} \mid T
\]

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

\[
\mathcal{A} ::= Area(T) \sim r \mid Area(T) \sim r \times Area(T)
\]

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

\[
\text{CRT} ::= \text{LM} \mid \text{RM} \mid \text{TM} \mid \text{BM} \mid \text{CT}
\]

The syntax for spatio-temporal functions are:

\[
\Theta ::= \text{Dist}(id, CRT, id, CRT) \sim r \mid Lat(id, CRT) \sim r \mid Lon(id, CRT) \sim r
\]

The syntax for the STPL formula is:

\[
\phi ::= \top \mid x.\phi \mid \exists id@x.\phi \mid \exists id.\phi \mid id = id \mid \sim \phi \mid \phi \vee \phi \mid \tau-x \sim t \mid F-x \sim n \mid F-x \% c \sim n \mid \exists \phi \mid \phi U \phi \mid \phi S \phi \mid \Theta \mid \exists \[ T \mid \mathcal{A}
\]

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

The syntax of STPL is substantially different from the syntax of MTL \( \times S4v \). 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 \( \exists id@x.\phi \) or the rule \( \exists id.\phi \) is the Existential object quantifier. When the formula \( \exists id@x.\phi(id) \) is evaluated, then it is satisfied (true) when there is an object \( id \) at frame/time \( x \) that makes \( \phi(id) \) true. The formula \( \exists id.\phi(id) \) is also searching for an object that makes \( \phi(id) \) 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 \( \forall id@x.\phi \equiv (\exists id@x.\neg \phi) \) or \( \forall id.\phi \equiv (\exists id.\neg \phi) \). The universal quantifier requires that all the objects in a frame satisfy the subformula \( \phi \). Notice that the syntax of STPL cannot enforce that all uses of time or object variables are bound, that is, 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 \( \times S4v \), 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:

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

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

\[
\exists id. (C(id) = \text{car} \land 0.9) \times P(id) > 0.9).
\]
\[ \exists \text{id}\in x. (C(id) = \text{car} \land 0(\tau - x \leq 1.5 \land P(id) > 0.9)) \]

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 \( F/C_{0} \) but distinguish their type based on how they are used in the constraint (\( \tau \) represents the current time, and \( F \) represents the current frame number). The operator \( \% \) in the expression \( F - x \) % \( C(id) \) is used to specify periodic constraints. For reasoning over the past time, we added \( \bigcirc \) (previous) and \( S \) (since) operators as duality for the \( \bigcirc \) and \( U \) operators, respectively.

For STPL formulas \( \psi, \phi \), we define \( \psi \land \phi \equiv (\neg \psi \lor \neg \phi) \), \( \bot \equiv \top \) (False), \( \psi \rightarrow \phi \equiv \psi \lor \phi \) (\( \psi \) Implies \( \phi \)), \( \phi \land \psi \equiv (\neg \phi \lor \neg \psi) \) (\( \phi \) releases \( \psi \)), \( \phi \lor \psi \equiv \phi \lor \psi \) (\( \phi \) non-strictly releases \( \psi \)), \( \neg \psi \equiv \top \U \psi \) (Eventually \( \psi \)), \( C(Id) \), \( P(Id, c) > a \) 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

\[
\begin{align*}
C(id) & = c \mid P(id) \sim a \mid P(id) \sim r \times P(id) \\
\text{in } \Theta \\
c \in C(id) & \mid P(id, c) \sim a \mid P(id, c) \sim a \times P(id, c).
\end{align*}
\]

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,

\[
\begin{align*}
\psi_1 & = x_1. (F - x_1 > 2 \land x_2. 0(\tau - x_2 < 0.01)) \\
\psi_2 & = \square \exists Id_1 \in x_1. (\square \forall Id_2 . (Id_1 = Id_2) \land (\tau - x_2 > 2)).
\end{align*}
\]

are AAN formulas. In formula \( \psi_1 \), the time variable \( x_1 \) is not used in the scope of the \( x_2 \) freeze time operator. In formula \( \psi_2 \), there are no nested quantifier/freeze time operators. Therefore, they are both AAN STPL formulas. On the other hand,

\[
\begin{align*}
\psi_3 & = x_1 . (F \land x_2 . 0(F - x_1 > 2 \land x_3 < 0.01)) \\
\psi_4 & = (\forall Id_1 \in x_1 . \bigcirc \forall Id_2 \in x_2 . \square \forall Id_1 . (P(Id_1) > P(Id_2) > P(Id_2) < P(Id_2))
\end{align*}
\]

are not AAN formulas. That is because in \( \psi_3 \), the variable \( x_1 \) is used in the scope of the nested quantifier operator “\( \forall \)”. In \( \psi_4 \), the \( Id_1 \) is used in the scope of the second nested quantifier operator “\( \forall \)”.

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 are 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,

\[
\begin{align*}
\phi_2 & = \square x. \square y. ((\tau - x > 1) \Rightarrow \square (\tau - y > 2))
\end{align*}
\]

which is not an AAN formula can be rewritten as AAN

\[
\phi_2’ = \square x. \square y. ((\tau - x > 1) \Rightarrow y. \square (\tau - y > 2)).
\]

On the other hand, although

\[
\begin{align*}
\phi_6 & = \square \exists Id_1 \in x. \square \forall Id_2 . (C(Id_1) = C(Id_2) \Rightarrow P(Id_2) > 0.9)
\end{align*}
\]

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

\[
\begin{align*}
\phi_6’ & = \square \exists Id_1 \in x. \square \forall Id_2 . (C(Id_1) = C(Id_2) \Rightarrow P(Id_2) > 0.9)
\end{align*}
\]

4.2. 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 \( \Theta \). 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.

4.2.1. Spatio-temporal functions. We define functions \( f \) as follows:

- Class of an object:

\[ f_C(id, D, i, e, \zeta) \] returns the \( \mathcal{R} (D(k), e(id)).\text{Class} \) as the class of the \( e(id) \) object in the \( k \)’th frame, where \( k \leftrightarrow \zeta(id) \) if \( \zeta(id) \) is specified, and \( k \leftrightarrow i \) otherwise.
• Probability of a classified object:

\[ f_{\text{Prob}}(id, D, i, e, \zeta) = \text{Prob}(D(k), e(id)) \]

returns the \( \mathcal{R}(D(k), e(id)) \). \( \text{Prob} \) as the probability of the \( e(id) \) object in the \( k \)'th frame, where \( k \leftarrow \zeta(id) \) if \( \zeta(id) \) is specified, and \( k \leftarrow i \) otherwise.

• Distance between two points from two different objects:

\[ f_{\text{dist}}(id_1, id_2, CRT_1, CRT_2, D, i, e, \zeta) \]

computes and returns the Euclidean Distance between the \( CRT_1 \) point of the \( e(id_1) \) object and \( CRT_2 \) point of the \( e(id_2) \) object in the \( k \)'th frame and \( k \) frame, respectively; and we have \( k_1 \leftarrow \zeta(id_1) \) if \( \zeta(id_1) \) is specified, and \( k_1 \leftarrow i \) otherwise, and \( k_2 \leftarrow \zeta(id_2) \) if \( \zeta(id_2) \) 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 2.3):

\[ f_{\text{LAT}}(id, CRT, D, i, e, \zeta) \]

computes and returns the Lateral Distance of the \( CRT \) point of the \( e(id) \) object in the \( k \)'th frame from the Longitudinal axis, where \( k \leftarrow \zeta(id) \) if \( \zeta(id) \) is specified, and \( k \leftarrow i \) otherwise.

• Longitudinal distance of a point that belongs to an object in a coordinate system (for image space, see section 2.3):

\[ f_{\text{LONG}}(id, CRT, D, i, e, \zeta) \]

computes and returns the Longitudinal Distance of the \( CRT \) point of the \( e(id) \) object in the \( k \)'th frame from the Lateral axis, where \( k \leftarrow \zeta(id) \) if \( \zeta(id) \) is specified, and \( k \leftarrow i \) otherwise.

• Area of a region:

\[ f_{\text{Area}}(T) \]

computes and returns the area of a spatial term \( T \) if \( T \) 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 \( D, i, e, \text{and } \zeta \) as arguments. Functions need the data stream \( D \) 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”) \( e \) to store and retrieve the values assigned to variables (both for time and objects), and, finally, a data structure \( \zeta \) to acquire the time/frame at which an object was selected through quantification. The data structure \( \zeta \) is used to distinguish whether the time that an object was selected is needed or not (see semantics for \( \exists \) in Section 4.2.2). 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) \( \epsilon_0 \) and \( \zeta_0 \) are empty. That is, no information is stored about any object identifier of the clock variable. The environment \( \zeta \) is needed for the comparison of objects, probabilities, etc. over different points in time, and environment \( \epsilon \) 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 \( M = \langle T, V, D, e, \zeta, \tau \rangle \) as a topological temporal model in which \( T \) is a topological space, \( V \) is a spatial valuation function, \( D \) is a data-object stream, \( \zeta : V_\tau \rightarrow N \cup \{0\} \) is an object evaluating function, and \( e : V_\tau \rightarrow N \) is a frame-evaluating function. Here, \( i \in N \) is the index of current frame, \( \tau : N \rightarrow R^+ \) is a mapping function from frame numbers to their physical times, \( \phi_1, \phi_2, \phi_2 \in \text{STPL} \) (i.e., formulas belong to the language of the grammar of STPL), \( V_\tau \) is a set of time variables, and \( V_\sigma \) is a set of object variables. The quality value of formula \( \phi \) with respect to \( D \) at frame \( i \) with evaluations \( e \) and \( \zeta \) is recursively assigned as follows:

#### 4.2.2. Semantics of temporal operators.

\[
[[T]],[D, i, e, \zeta, \tau] := T
\]

\[
[[x, \phi]] [(D, i, e, \zeta, \tau) := [[\phi]] \langle (D, i, e, x \leftarrow i) \rangle (\zeta, \tau)
\]

\[
[[\exists id \in x, \phi]] [(D, i, e, \zeta, \tau) := \bigvee_{id \in \zeta(V_\phi)} \langle [\phi] \rangle [D, i, e, \epsilon[id \leftarrow k, x \leftarrow i] \rangle (\zeta[\epsilon[id \leftarrow i])]
\]

\[
[[\tau = x \sim n]] [(D, i, e, \zeta, \tau) := \begin{cases} \top & \text{if } \tau(i) - \tau(0) \sim n \\ \bot & \text{otherwise} \end{cases}
\]

\[
[[F \sim c \sim n]] [(D, i, e, \zeta, \tau) := \begin{cases} \top & \text{if } c \sim n \\ \bot & \text{otherwise} \end{cases}
\]

\[
[[id \leftarrow id_1]] [(D, i, e, \zeta, \tau) := \begin{cases} \top & \text{if } e(id) = e(id_1) \\ \bot & \text{otherwise} \end{cases}
\]

\[
[[\neg \phi]] [(D, i, e, \zeta, \tau) := \neg [[\phi]] (D, i, e, \zeta, \tau)
\]

\[
[[\phi_1 \lor \phi_2]] [(D, i, e, \zeta, \tau) := [[\phi_1]] \lor [[\phi_2]] (D, i, e, \zeta, \tau)
\]

\[
[[\phi_1 \land \phi_2]] [(D, i, e, \zeta, \tau) := [[\phi_1]] \land [[\phi_2]] (D, i, e, \zeta, \tau)
\]

\[
\exists_j \phi[[\phi]] [(D, i, e, \zeta, \tau) := \exists_j [[\phi]] (D, i, e, \zeta, \tau)
\]
4.2.3. Semantics of past-time operators. Many applications require requirements that refer to past time. Past-time operators are particularly relevant to online monitoring algorithms.

\[
[[\phi_1 \land \phi_2]]((D, i, \epsilon, \zeta, t)) :=
\nu_j \nu_i \left( [[\phi_2]]((D, j, \epsilon, \zeta, t)) \land \nu_j \nu_i [[\phi_1]]((D, k, \epsilon, \zeta, t)) \right)
\]

\[
[[\neg \phi]]((D, i, \epsilon, \zeta, t)) := [[\phi]]((D, i - 1, \epsilon, \zeta, t))
\]

4.2.4. 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 \( \times S_4 \), logic. The semantics of STPL will be based on the valuation function \( V \) of MTL \( \times S_4 \), with the change that \( V \) will now also accept the environments \( \epsilon \) and \( \zeta \), and we extend the definition of \( V \) over the spatial term \( \sigma(id) \). That is, we replace in the semantics of the MTL \( \times S_4 \), the valuation of spatial propositions \( p \) with:

\[
V(\sigma(id), D, i, t, \epsilon, \zeta) = \Pi(\epsilon(id), k),
\]

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:

\[
[[\text{T}]][D, i, \epsilon, \zeta, t)] := V([\text{T}] D, i, \epsilon, \zeta)
\]

\[
[[\text{Area}(T) \sim r]][D, i, \epsilon, \zeta, t)] :=
\begin{cases}
\top & \text{if } f_{\text{area}}(V(T, D, i, t, \epsilon, \zeta)) \text{ is unspecified} \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[[\text{C}(id) = r]][D, i, \epsilon, \zeta, t)] :=
\begin{cases}
\top & \text{if } f_{c}(id, D, i, \epsilon, \zeta)) = r \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[[\text{P}(id) \sim r]][D, i, \epsilon, \zeta, t)] :=
\begin{cases}
\top & \text{if } f_{p}(id, D, i, \epsilon, \zeta)) \sim r \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[[\text{Dist}(id_1, CRT_1, id_2, CRT_2) \sim r]][D, i, \epsilon, \zeta, t)] :=
\begin{cases}
\top & \text{if } f_{\text{dist}}(id_1, id_2, CRT_1, CRT_2, D, i, \epsilon, \zeta)) \sim r \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[[\text{LAT}(id, CRT) \sim r]][D, i, \epsilon, \zeta, t)] :=
\begin{cases}
\top & \text{if } f_{\text{LAT}}(id, CRT, D, i, \epsilon, \zeta)) \sim r \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[[\text{LON}(id, CRT) \sim r]][D, i, \epsilon, \zeta, t)] :=
\begin{cases}
\top & \text{if } f_{\text{LON}}(id, CRT, D, i, \epsilon, \zeta)) \sim r \\
\bot & \text{otherwise}
\end{cases}
\]

with \( \sim \in \{>, <, =, \leq, \geq \} \), and “otherwise” has a higher priority to become true in the if statements if \( \epsilon(id) \notin CI(D(i)) \).

Note that we omit the semantics for some of the spatio-temporal operators in the production rule \( \Theta \) 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 \( D \) satisfies the STPL formula \( \phi \) under the current environment if \( [[\phi]]((D, 0, \epsilon_0, \zeta_0, t)) = \top \), which is equivalent to denote \((D, 0, \epsilon_0, \zeta_0, t) \models \phi \).

Note that by \( \epsilon_0 \) and \( \zeta_0, \) 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

\[
\phi := \exists id_1, \exists id_2, \exists (\sigma(id_1) \cup \sigma(id_2))
\]

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 seconds, 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 \( \sigma(id_1) \cup \sigma(id_2) \) 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 \( \sigma(id_1) \) and \( \sigma(id_2) \) in each frame, respectively. That is, \( p_1 \) at \( t = 2 \) is equivalent to \( \Pi(\epsilon(1), 2) \). The evaluation of STPL formula \( \phi \) starts with the following equation

\[
[[\text{LAT}(id_1) \cup \text{LAT}(id_2)) \sim r]][D, 0, \epsilon_0, \zeta_0, t)] := V(\exists p_1 \cup \exists p_2, D, 0, r, \epsilon, \zeta)
\]

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:
The step-by-step computation of $V(p_1 U_{[0,2]} p_2, D, 0, \tau, \epsilon, \zeta)$ for $i = 0$ to $2$. Here, $p_1$ and $p_2$ are the same as in Figure 5.

Figure 5. The step-by-step computation of $V(p_1 U_{[0,2]} p_2, D, 0, \tau, \epsilon, \zeta)$ for $i = 0$ to $2$. Here, $p_1$ and $p_2$ are the same as in Figure 5.

The step-by-step computation of $V(p_1 U_{[0,2]} p_2, D, 0, \tau, \epsilon, \zeta)$ for $i = 0$ to $2$. Here, $p_1$ and $p_2$ are the same as in Figure 5.
ψ₁ = \forall Idᵢ. ∃ Idⱼ. (Idᵢ = Idⱼ ⇒ φ₀),

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

ψ₉ = □∀Idᵢ.∀x.∃Idⱼ.

(∃[σ(Idᵢ)v σ(Idⱼ)] ∩ C(Idᵢ) = C(Idⱼ) ⇒ φ₀)

Notice the increase in complexity between the two formulas. In ψ₁, we just need to store the ID of the object in Idᵢ, and just check if in the future, there is another object (Idⱼ) with the same ID. In ψ₉, we need to store the time x that we observed the object Idᵢ, so that we can retrieve its bounded box and check for intersection with future objects along with checking the agreement of classes. Clearly, ψ₁ and ψ₉ are not syntactically or semantically equivalent. Nevertheless, under the assumption of sufficient sampling rate, then the objects that satisfy ψ₁ will also satisfy ψ₉. 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 1 to each STPL formula, and presented the result in Table 2.

5.1. 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):

φ₃ = 0 ∃ Id₁. ∃ Id₂. (Id₁ ≠ Id₂ ∧ C(Id₁) = C(Id₂)) (1)

Figure 6. s₂ = V(p₂, D, 0, τ, ε, ζ),

s₀ = V(p₂, D, 1, τ, ε, ζ) \cap V(p₁, D, 0, τ, ε, ζ),

s₁ = V(p₂, D, 2, τ, ε, ζ) \cap V(p₁, D, 0, τ, ε, ζ) \cap V(p₁, D, 1, τ, ε, ζ),

V(p₁[0, 2], p₂, D, 0, τ, ε, ζ) = s₀ \cup s₁ \cup s₂.

The car inside the red rectangle is identified with ID 4 in frame 10, and it is annotated as "partly occluded".

The same car as in image (a) is bounded by the yellow rectangle, and it is annotated as "largely occluded" in frame 11.

The car inside the red rectangle is identified with ID 5 in frame 14, and it is annotated as "partly occluded".

The same car as in image (c) is bounded by the yellow rectangle, and it is annotated as "largely occluded" in frame 15.

The car inside the red rectangle is identified with ID 16 in frame 200, and it is annotated as "partly occluded".

The same car as in image (e) is bounded by the yellow rectangle, and it is annotated as "largely occluded" in frame 261.

Figure 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.
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 $D$ in Table 1, and as depicted in the frames in Figure 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 $(D, 0, e_0, \xi_0, \tau) \models \phi_3$. Note that we can push the $\Diamond$ operator after the existential quantifier (i.e., $\exists Id_1 \exists Id_2, \phi \equiv \exists Id_1 \exists Id_2, \Diamond \phi$) and still expect the same result.

Table 1. Data stream $D$ of image frames in Figure 1.

| Frame | $\tau$ | 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        |

Each row for the column headers: Frame, $\tau$, 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 boxes) by which a bounding box is associated with each classified object. Note that some objects are not tracked correctly throughout the data stream.

Table 2. The result of monitoring STPL formula $\phi$ on the data stream $D$ in Table 1.

| $\phi$ | [ $\phi$ ] | $\phi$ | [ $\phi$ ] | $\phi$ | [ $\phi$ ] | $\phi$ | [ $\phi$ ] |
|--------|------------|--------|------------|--------|------------|--------|------------|
| Eq. (1) | $\top$ | Eq. (2) | $\bot$ | Eq. (3) | $\bot$ | Eq. (4) | $\bot$ |
| Eq. (5) | $\bot$ | Eq. (9) | $\top$ | Eq. (10) | $\top$ | Eq. (11) | $\bot$ |
| Eq. (12) | $\top$ | Eq. (13) | $\bot$ | Eq. (14) | $\bot$ | Eq. (15) | $\bot$ |
| Eq. (16) | $\top$ | Eq. (16)$^a$ | $\bot$ | Eq. (17) | $\top$ | Eq. (18) | $\top$ |

$^a$We used the dataset “0018” from KITTI tracking benchmark for evaluating formula in Eq. (16). [ $\phi$ ] abbreviates [ $\phi$ ] $D$, $0, e_0, \xi_0, \tau$. 

5.2. 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 2 s, unless Frame Per Second (FPS) drops below 10 during the first second.

STPL (using Assumptions 5.1–5.3):

$$\phi_4 = \Box \forall Id_i \in X, (\psi_{41, x}^{Id_i} \Rightarrow \psi_{42}^x)$$

where

$$\psi_{41, x}^{Id_i} = \Diamond \exists Id_2, (Id_1 = Id_2 \land \tau - x \leq 2 \land P(Id_2) < 0.9 \times P(Id_1))$$

$$\psi_{42}^x = \exists \phi(Ratio(x, \tau - x) < 10 \land \tau - x \leq 1)$$
Notice that the function “Ratio \((F - x, \tau - x)\)” 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 “\((F - x)\)” and “\((\tau - x)\).” We remark that the function Ratio introduces the possibility of division by zero. In the formula \(\psi_{a3}^t\), the issue is avoided in the specification by using the next time operator (\(\Box\)). However, an implementation of the function Ratio should handle the possibility of division by zero. The subformula \(\psi_{a4,t,x}^{id1,x}\) searches for a vehicle for which within \(2\) s in the future, that is, \((\tau - x \leq 2)\), 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 \(\psi_{a5}^t\) checks whether within \(1\) s in the future, the FPS drops below \(10\). The implication \(\psi_{a4,t,x}^{id1,x} \Rightarrow \psi_{a5}^t\) 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 1, the rest of the objects satisfy the requirement. Therefore, the whole requirement is not satisfiable, that is, \((D, 0, \epsilon_0, \zeta_0, \tau) \models \phi_f\). 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 \(0.7, 0.9,\) and \(0.64\) during the first, second, and third frames, respectively. Formula (2) without the always operator is not satisfactory 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 \(\bigcirc\). The weak previous operator does not become unsatisfiable in the first frame on a data stream where there is no previous frame. Similarly, by \(\bigcirc\), 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):**

\[
\phi_{a6} = \Box \forall Id_1 \forall x. \left( \psi_{a4}^{id1,x} \Rightarrow \psi_{a5}^t \right)
\]

where

\[
\psi_{a4}^{id1,x} = \bigcirc \forall Id_1. (Id_1 \neq Id_3)
\]

\[
\psi_{a5}^t = (F - x \leq 1 \land \tau - x \leq 2) \Rightarrow \exists Id_2. (Id_1 = Id_2)
\]

In formula \(\phi_{a6}\), the antecedent subformula \(\psi_{a4}^{id1,x}\) checks if an object did not exist in the previous frame. Therefore, the consequent \(\psi_{a5}^t\) 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 two frames during \(1\) s.

**STPL (using Assumption 5.2–5.3):**

\[
\phi_5 = \Box \forall Id_1 \forall x. \left( \bigcirc \forall \exists Id_2. (Id_1 \neq Id_3) \right) \Rightarrow \left( \Box \exists Id_3. (Id_1 = Id_2) \right)
\]

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 \(1\) s 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 \(D\) in Table 1, that is, \((D, 0, \epsilon_0, \zeta_0, \tau) \models \phi_5\).

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 \(1\) s, after which the object does not change class until it disappears.

**STPL (using Assumptions 5.1–5.3):**

\[
\phi_6 = \Box \forall Id_1 \forall x. \left( (C(Id_1) = 0 \Rightarrow \psi_{a6}^{id1,x}) \land (C(Id_1) > 0 \Rightarrow \psi_{a1}^{id1,x}) \right)
\]

where the subformulas are defined as:

\[
\psi_{a6}^{id1,x} = \Box (\exists Id_3. (Id_1 = Id_3 \land C(Id_3) > 0))
\]

\[
\psi_{a1}^{id1,x} = \exists Id_1. (F - x \geq 1 \land Id_3 = Id_1) \Rightarrow C(Id_1) = C(Id_3)
\]

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 \(\psi_{a6}^{id1,x}\) 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 \(\psi_{a1}^{id1,x}\) requires that when a classified object was observed, then afterward the same object can only take the same class.

Running Formula in Real Data Stream: In the frame \(1\) in Table 1, the object with \(ID = 2\) changes its class from cyclist to pedestrian. Therefore, the data stream \(D\) does not satisfy the requirement \((D, 0, \epsilon_0, \zeta_0, \tau) \models \phi_6\).
5.3. 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.

5.3.1. Basic spatial examples without quantifiers. Let us assume that we have a spatial proposition \( p \) as in Figure 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.

\[
\phi_7 = \Box \neg \neg \Box p
\]  

(6)

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

**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 across all the frames is not equal to the universe.

\[
\phi_8 = \neg \Box \neg \Box p
\]  

(7)

**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 \((x_1, y_1, x_2, y_2)\).

\[
\phi_9 = \Box \forall Id_1. (\text{LAT}(Id_1, LM) \geq x_1 \wedge \text{LAT}(Id_1, RM) \leq x_2 \wedge \text{LON}(Id_1, TM) \geq y_1 \wedge \text{LON}(Id_1, BM) \leq y_2)
\]  

(8)

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 \( D \) in Table 1, then we have \((D, 0, e_0, \zeta_0, t) \models \phi_9\).

**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):

\[
\phi_{10} = \Diamond \exists Id_1 \forall x \diamond \exists Id_2.
\]  

(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 \( D \) as in Table 1 satisfies the above formula \((D, 0, e_0, \zeta_0, t) \models \phi_{10}\). 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 spatial constraints.

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

**STPL** (using Assumptions 5.2–5.3):

\[
\phi_{11} = \Box \forall Id_1 \forall x \forall y. (\psi_{111} \Rightarrow \psi_{112})
\]  

(11)

where

\[
\psi_{111} = Id_1 = Id_2 \wedge \text{LAT}(Id_1, LM) < \text{LAT}(Id_2, LM) \wedge \text{LAT}(Id_2, LM) > \frac{3}{4} W \wedge \Box_w \forall Id_3. (Id_3 \neq Id_1)
\]

\[
\psi_{112} = \Box_w \forall \Box ((r - y < 1) \wedge \exists Id_4. (Id_4 \neq Id_1) \wedge \Box_w \exists Id_5. (Id_5 = Id_3) \Rightarrow \Box_w \text{LAT}(Id_5, LM) > \frac{3}{4} W
\]

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 1, 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 $\psi_{111}$ 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 $\psi_{111}$). Note that, from the second frame afterward, the tracking algorithm mixes the ID association to the objects, and as a result, the geometrical positions of the objects are not consistent. The inconsistent object tracking makes the subformula $\psi_{111}$ not hold for viable cases. Therefore, for the data stream $D$ in Table 1, we have $(D, 0, \epsilon_0, \zeta_0, \tau) \models \phi_{11}$. 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 s, 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.

**STPL** (using Assumptions 5.2–5.3):

$$\phi_{12} = \Box \forall I d_1 \exists x . ((C(I d_1) = \text{Ped}) \land P(I d_1) > 0.8) \Rightarrow$$

$$\Box (\tau - x \leq 1 \Rightarrow$$

$$\exists I d_2 . (I d_1 = I d_2 \land P(I d_2) > 0.7)$$

$$\land C(I d_2) = \text{Ped} \land$$

$$\forall I d_3 . (I d_1 \neq I d_3 \Rightarrow$$

$$\neg \Box (\sigma(I d_2) \sqsubseteq \sigma(I d_3))))$$

(12)

Running Formula in Real Data Stream: In Table 1, there is no pedestrian associated with probability higher than 0.8; therefore, the data stream $D$ satisfies the above formula $(D, 0, \epsilon_0, \zeta_0, \tau) \models \phi_{12}$.

**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):

$$\phi_{13} = \Box \forall I d_1 \exists x . (C(I d_1) = \text{Car} \Rightarrow$$

$$\Box \forall I d_2 . ((I d_1 = I d_2 \land C(I d_2) = \text{Car}) \Rightarrow$$

$$\text{Area}(I d_1) \geq \text{Area}(I d_2))$$

(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 1, the bounding box of the car with ID = 1 has expanded in the frame 2. Therefore, the data stream $D$ does not satisfy the above formula $(D, 0, \epsilon_0, \zeta_0, \tau) \models \phi_{13}$.

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

- Subset: $p_1 \subseteq p_2 \equiv \forall \Box(p_1 \sqsubseteq p_2)$
- Set equality: $p_1 = p_2 \equiv p_1 \& \& p_2 \sqsubseteq p_1 \land p_2 \sqsubseteq p_1$

**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):

$$\phi_{14} = \forall I d_1 \exists x . \Box \exists I d_2 , (I d_1 = I d_2 \land \sigma(I d_1) = \sigma(I d_2))$$

Notice that we quantify over all objects in the first frame $(I d_1)$, and then we require that for all future times, there is an object $(I d_2)$ 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:

$$\phi_{14}'' = \Box \forall I d_1 \exists x . \Box \exists I d_2 , (I d_1 = I d_2 \land \sigma(I d_1) = \sigma(I d_2))$$

(14)

Another formalization for the last requirements is

$$\phi''_{14} = \Box \forall I d_1 , (\Box \exists \sigma(I d_1) = 0 \sigma(I d_1))$$

(15)

Running Formula in Real Data Stream: All of the objects in Table 1 have different bounding boxes in different frames. Thus, the data stream $D$ does not satisfy the above formulas $(D, 0, \epsilon_0, \zeta_0, \tau) \not\models \phi_{14}$.

**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):

$$\phi_{15} = \Box \forall I d_1 \exists x \exists (\psi_{\text{prob}} \land \psi_{\text{frames}} \land \square \psi_{\text{disappear}})$$

(16)

and the predicates are defined as:
while they were inconsistent with the rest of the occluded three frames. In each a car was labeled as dataset "validate the correctness of a training data stream that is used the 21 datasets, except for four of them (i.e., datasets Hekmatnejad et al. 221 proposed this problem method to check if the bounding box of the two objects intersect each other in the current frame, or one bounding box in the current frame intersects with the other bounding box in the next frame. Similarly, we can formalize the occlusion without using STE operators by rewriting $\phi_{occ}$ as below

$$\phi_{occ} = \exists Id_1, \exists Id_4. (Id_1 \neq Id_3 \land Id_1 = Id_4 \land \text{Dist}(Id_4, CT, Id_3, CT) < d_3)$$

Note that the second formalization is less realistic because it puts a threshold on the Euclidean distance (i.e., $d_3$ is a positive real number) between two objects to infer their overlap.

Running Formula in Real Data Stream: All the objects in Table 1 are close to the borders of the images, and the data stream does not satisfy the subformula $\phi_{borders}$. Therefore, the data stream $D$ satisfies the above formula $\phi > 0.8$

$$\phi_{high} = P(Id_1) > 0.8$$

$$\phi_{borders} = LON(Id_1, TM) > d_1 \land LON(Id_1, BM) < d_2 \land \text{LAT}(Id_1, LM) > d_3 \land \text{LAT}(Id_1, RM) < d_4$$

$$\phi_{disapp} = \forall Id_2, (Id_1 \neq Id_2)$$

$$\phi_{occ} = \exists Id_1, \exists Id_3. (Id_1 \neq Id_3 \land Id_1 = Id_3 \land \exists \sigma(\text{Dist}(Id_4, CT, Id_3, CT) < d_3)$$

In the above formulas, the subformula $\phi_{borders}$ 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 $\forall Id_2, (Id_1 \neq Id_2)$ is to check if an object disappeared. In the subformula $\phi_{occ}$, 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 sub-formulas check if the bounding box of the two objects intersect each other in the current frame, or one bounding box in the current frame intersects with the other bounding box in the next frame. Similarly, we can formalize the occlusion without using STE operators by rewriting $\phi_{occ}$ as below

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):**

$$\phi_{16} = \Box \forall Id_1 \in \mathcal{X}. ((\bigcirc \forall Id_2, (Id_1 \neq Id_2)) \Rightarrow$$

$$\forall Id_2, (Id_1 = Id_2 \Rightarrow \text{Ratio} \{\text{Area}(\sigma(Id_1) \cap \sigma(Id_2))\} > 0.1))$$

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 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 $D$ in Table 1 satisfies the above formula $(D, 0, 0, 0, 0, 0) \models \phi_{16}$.

### 5.4. Examples of space and time requirements with 3D environment

#### 5.4.1. 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-coding 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 Figure 8(a). The corresponding camera image for the LiDAR scene is shown in Figure 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 Figure 8(b) and (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 Figure 8(e) and (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 specification in STPL that can formalize such missed-detection scenarios.
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 Figure 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):

\[ \phi_{17} = \Box \forall \text{Id}_1, \forall \text{Id}_3 \forall x. (\left( \phi_{\text{exist}} \land \phi_{\text{dist \ close}} \right) \Rightarrow \diamond \neg \phi_{\text{before \ existed}}) \]

and the predicates are defined as:

\[ \phi_{\text{exist}} = (C(\text{Id}_1) = \text{Car}) \land (C(\text{Id}_3) = \text{Drv}) \]

\[ \phi_{\text{dist \ close}} = LON(\text{Id}_1, BM) \leq 1.2 \times LON(\text{Id}_1, BM) \land \\
LON(\text{Id}_1, TM) \geq 0.5 \times LON(\text{Id}_3, BM) \land \\
LAT(\text{Id}_1, CT) \geq LAT(\text{Id}_1, LM) \land \\
LAT(\text{Id}_1, CT) \leq LAT(\text{Id}_1, RM) \land \\
\phi_{\text{before \ existed}} = \exists \text{Id}_2. (C(\text{Id}_2) = \text{Car} \land \text{Id}_1 = \text{Id}_2 \land \\
\text{Dist}(\text{Id}_2, CT, \emptyset, CT) > \text{Dist}(\text{Id}_1, CT, \emptyset, CT) \]

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 \( \phi_{\text{before \ existed}} \), we check if the car identified by Id_2 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 Figure 9 falsifies the formula.

Figure 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) Annotated LiDAR point cloud image at time \( t_0 \). (b) Annotated LiDAR point cloud image at time \( t_1 \). (c) Annotated camera image for the LiDAR image on the left. (d) Annotated camera image for the LiDAR image on the left. (e) Colored semantic segmented of the LiDAR data at time \( t_0 \). (f) Colored semantic segmented of the LiDAR data at time \( t_1 \).
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 \( \varphi_{\text{close}} \) to encode if the cars drive in the same lane.

### 5.4.2. 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 \( \text{Visible} \) by which we can check if given two cars are visible from the viewpoint of a third car. Additionally, we need to add a data attribute \( \text{Empty} \) to the data-object stream \( D \) that determines if a bounding volume of an object is empty (e.g., \( R(D(i), ID). PC = \emptyset \) checks if there is no point could data associated with an object identified by \( ID \)) or not. For instance, by \( R(D(i), ID). Empty \) 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.

\[
\left[ [V(id, CRT, id, id) \sim r](D, i, \epsilon, \zeta, \tau) := \begin{cases} \top & \text{if } f_{\text{visible}}(id, CRT, id, id, D, i, \epsilon, \zeta) \sim r \\ \bot & \text{otherwise} \end{cases} \right]
\]

\( f_{\text{visible}}(id_1, CRT, id_2, id_3, D, i, \epsilon, \zeta) \) returns true if the \( \epsilon(id_2) \) and \( \epsilon(id_3) \) objects are visible from the viewpoint \( CRT \) 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.

---

**Figure 9.** The bird-view of LiDAR point clouds for the two consecutive frames in Figure 8 where the drivable area in front of the ego is shown. (a) \( x-y \) plane bird-eye view of LiDAR point cloud at time \( t_0 \). (b) \( x-y \) plane bird-eye view of LiDAR point cloud at time \( t_1 \).
\[
\phi_{18} = \square \forall Id_1. \forall Id_2. (\neg E(Id_1) \land \neg E(Id_2) \land (Id_1 \neq Id_2) \\
\land (C(Id_1) = \text{Car}) \land (C(Id_2) = \text{Car}) \\
\land \text{LAT}(Id_1, \text{RM}) < -1 \land \text{LAT}(Id_2, \text{RM}) < -1 \\
\land \text{LAT}(Id_1, \text{RM}) > -6 \land \text{LAT}(Id_2, \text{RM}) > -6 \\
\land MD(Id_1) = MD(Id_2) \\
\land \sigma(V(U, CT, Id_1, Id_2)) = \sigma(V(U, CT, Id_1, Id_2)) \\
\Rightarrow \\
\neg (E(Id_1) \land \neg E(Id_2)))
\]  

(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 and 6 m from the left side of the ego car). We assume that the perception system returns 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 \(V(U, CT, Id_1, Id_2)\) requires a visible angle between the two cars from the point of view of the ego car (i.e., the center point of the \(U\)) 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 Figure 10 satisfies the formula in Eq. (20).

6. 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

\[
\phi = \square \forall Id_1 \land \exists \text{Id}_2. (\lor (\sigma(Id_1) \cup \sigma(Id_2))) \land \\
\lor (\sigma(Id_2) \cup \sigma(Id_1)) \land \text{Id}_1 = \text{Id}_2
\]

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

6.1. Algorithm-1

This algorithm represents the main dynamic programming body of the monitoring algorithm. It receives an STPL...
formula $\phi$, a data stream $\hat{p}$, and dynamic programming tables $M$ and $F$. The formula is divided into subformulas as in the example in Figure 11. That is, the formula is divided into subformulas based on the presence of freeze time operators ($\phi_i$) and spatial quantifiers ($\phi_j^i$). It is important to note that the nested subformulas have lower index, for example, $\phi_1$ is nested in $\phi_2$ as in Figure 11. Notice that when we have subindexes, that is, $\phi_{i,j}$, 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 four 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 LTL and the $S_4u$ terms. Lines 10–14 identify whether the subformula $\phi_{k,j}$ is a spatio-temporal operator and calls the respective function.

6.2. Algorithm-2

This algorithm computes the values of the DP table $M$ as defined in the temporal semantics of the STPL logic in Def. 4.2.2. For a given subformula, first, it determines the operator 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 $\phi_j$ 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 $(SO)^{|V_{id}|}$, where $SO$ 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 $\phi_j$. 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.

Figure 11. A parse tree for the formula in Eq. (14). Distinct regions represent subformulas $\phi_2$, $\phi_1$, $\phi_1$, and $\phi_2$. In each region, the subformulas subscript into the operators and predicates levels.
6.3. 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 \( F(n(\ldots)) \), if it has nested 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 the extended computation that merges these tables for different subformulas in Algorithm 2 by calling function UpdateIdTable at line 24.

6.4. 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, for example, 2D versus 3D, a number of 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).
Algorithm 3. Compute Function Expression

Input: \( \varphi, k, u, u_{01}, t, \bar{\rho} \)

procedure \textsc{ComputeFnExpr}(\( \varphi, k, u, u_{01}, t, \bar{\rho} \))
1: \( \varphi_k \equiv \text{Fn}(\Omega_1, \ldots, \Omega_m) \rightarrow r \) where
\( r \in \mathbb{R}, \quad k \in \{1, 2, \ldots, m\} \)
and \( \text{Fn} \) is a reserved function name
2: \( F_{u_{01}} \) is a list used to store the evaluated arguments of \( \text{Fn} \)
3: \( S_{id} \leftarrow (S_D)^{\text{variables}} \) where
\( S_D \) is the maximum number of objects in all the frames
and \( V_{id} \) is the set of the object variables in the scope of \( \varphi_k \)
4: \( S_{O}^{S_{id}} \equiv \) number of objects in the current frame \( u \)
5: \( S_{O}^{S_{id}} \equiv \) number of objects in the frame of the current variable \( t \)
6: \( F_{iD} \) is a map s.t.
   \( \forall I \in V_{id} \quad \{ F_{iD}[I] = \text{true} \mid \exists I \in V_{id} \in \varphi \}
   \) or
\( \{ F_{iD}[I] = \text{false} \mid I \in V_{id} \in \varphi \} \)
7: for \( i \leftarrow 1 \) to \( S_{id} \) do
   \( \triangleright \) iterates over combinatorial assignments of values to the IDs
8: \( \{ \text{Id} \in V_{id} \mid F_{iD}[\text{Id}] = \text{true} \} \)
9: to \( \{ \text{Id} \in V_{id} \mid F_{iD}[\text{Id}] = \text{false} \} \)
10: if \( 3 \text{Id} \in V_{id} \) and \( (F_{iD}[\text{Id}] = \text{true} \) and \( I \in S_{O}^{S_{id}} \))
   or
   \( (F_{iD}[\text{Id}] = \text{false} \) and \( I \in S_{O}^{S_{id}} \))
then
11: \( M[k, u_{01}], IT[t-1] \leftarrow 1 \)
12: else
13: for \( j \leftarrow 1 \) to \( n \) do
   \( \triangleright \) iterates over \( \text{Fn} \) arguments
14: if \( \Omega_j \) is a constant number or reserved word then
15: \( P[j] \text{val} \leftarrow \Omega_j \)
16: else if \( \sigma(\Omega_j) \equiv \text{Fn}(\ldots) \) then
17: \( P[j] \text{val} \leftarrow \text{ComputeFnExpr}(\Omega_j, k, u, u_{01}, t, \bar{\rho}) \)
18: else if \( \sigma(\Omega_j) = \sigma(\text{Id}) \) then
19: \( P[j].\text{region} \leftarrow \text{BoundingBox}(\text{Id}, F_{iD}, u, t) \)
20: else if \( \Omega_j \) is a spatial formula then
21: \( P[j].\text{region} \leftarrow \text{ComputeSpatioTemporal}(\Omega_j, u, t, \bar{\rho}) \)
22: end if
23: end for
24: \( M[k, u_{01}], IT[t-1] \leftarrow \text{Fn}(\text{P}) + r \)
25: end if
26: end for
27: \( S_{id} \leftarrow (S_D)^{\text{variables}} \)
where \( L \) is the maximum number of object variables that can be used
in the scope of any \( \text{G} \) formula
28: for \( i \leftarrow S_{id} - 1 \) do
   \( \triangleright \) iterates over the remaining of the ID Table
29: \( M[k, u_{01}], IT[t] \leftarrow \text{NaN} \)
30: end for
end procedure

Algorithm 4. Spatio-Temporal Monitor (with intervals)

Input: \( \theta, u', t, s, \bar{\rho} \)

procedure \textsc{ComputeSpatioTemporal}(\( \theta, u', t, s, \bar{\rho} \))
1: \( \{ \text{Id}_1, \ldots, \text{Id}_{\text{Id}_{\text{var}}} \} \rightarrow \text{i-th combinatorial assignment of } \{1, \ldots, S_D\}
\text{to object variables} \)
2: for \( u \leftarrow \lfloor s \rfloor \) down to \( u' \) do
   \( \triangleright \) iterates backward over time stamps
3: \( S_{O}^{\text{Id}} \equiv \) number of objects in the current frame \( u \)
4: \( S_{O}^{\text{Id}} \equiv \) number of the objects in the frame of the current variable \( t \)
5: for \( j \leftarrow \theta \) max down to \( \theta \) min do
   \( \triangleright \) iterates over subformulas
6: if \( \varphi_j \equiv \sigma(\text{Id}) \) then
7: if \( (F_{iD}[\text{Id}] = \text{true} \) and \( I \in S_{O}^{\text{Id}} \)) or
   \( (F_{iD}[\text{Id}] = \text{false} \) and \( I \in S_{O}^{\text{Id}} \))
then
8: \( \tilde{M}[j, u] \leftarrow \text{NaN} \)
   \( \triangleright \) mark as NaN
9: else
10: \( \tilde{M}[j, u] \leftarrow \text{BoundingBox}(\text{Id}, F_{iD}, u, t) \)
11: end if
12: else if \( \varphi_j \equiv \neg \varphi_m \) then
13: \( \tilde{M}[j, u] \leftarrow \text{Complement}(\tilde{M}[m, u]) \)
14: else if \( \varphi_j \equiv \varphi_m \land \varphi_n \) then
15: \( \tilde{M}[j, u] \leftarrow \text{Intersection}(\tilde{M}[m, u], \tilde{M}[n, u]) \)
16: else if \( \varphi_j \equiv \varphi_m \lor \varphi_n \) then
17: \( \tilde{M}[j, u] \leftarrow \text{Union}(\tilde{M}[m, u], \tilde{M}[n, u]) \)
18: else if \( \varphi_j \equiv I \varphi_m \) then
19: \( \tilde{M}[j, u] \leftarrow \text{Interior}(\tilde{M}[m, u]) \)
20: else if \( \varphi_j \equiv C \varphi_m \) then
21: \( \tilde{M}[j, u] \leftarrow \text{Closure}(\tilde{M}[m, u]) \)
22: else if \( \varphi_j \equiv \varphi_m \text{ U}_j \varphi_n \) then
23: if \( u = \lfloor s \rfloor - 1 \) and \( \theta \in \mathbb{I} \) then
24: \( \tilde{M}[j, u] \leftarrow \tilde{M}[m, u] \)
25: else if \( u = \lfloor s \rfloor - 1 \) and \( \theta \notin \mathbb{I} \) then
26: \( \tilde{M}[j, u] \leftarrow 1 \)
27: else if \( \mathbb{I} = [0, +\infty) \) then
28: \( \tilde{M}[j, u] \leftarrow \tilde{M}[m, u \cup \tilde{M}[j, u + 1]] \)
29: else
30: \( b_l \leftarrow \min(j + \mathbb{I}) \); \( b_u \leftarrow \max(j + \mathbb{I}) \)
31: \( \tilde{r}_{\text{min}} \leftarrow \text{interpolation}(\tilde{m}[m, j]) \)
32: \( \tilde{M}[j, u] \leftarrow 1 \)
33: for \( j' \leftarrow b_l \) to \( b_u \) do
   \( \triangleright \) iterates over frame indices
34: \( \tilde{M}[j', u] \leftarrow \tilde{M}[m, j'] \cup (\tilde{M}[m, j'] \cap \text{r}_{\text{min}}) \)
35: \( \text{r}_{\text{min}} \leftarrow \text{interpolation}(\tilde{m}[m, j']) \)
36: end for
37: if \( s \in \mathbb{I} = \{ +\infty \} \) then
38: \( \tilde{M}[j, u] \leftarrow \tilde{M}[m, u \cup \tilde{M}[j, u + 1]] \)
39: end if
40: end if
41: end if
42: end for
43: end for
44: return \( \tilde{\tilde{M}}[\theta \min, u'] \)
end procedure
6.5. Correctness and complexity analysis

Let \( \hat{\rho} \) be an input signal of size \( |\hat{\rho}| \), \( \phi \) be an AAN STPL formula of size \( |\phi| \) (note that size of a formula is the summation of the number of spatial, temporal and spatio-temporal subformulas), \( |V| \) be the size of freeze time variables (or 1 if there is none), \( S_d \) 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 \( \phi \) is a TPTL formula, then it is known from Dokhanchi et al. (2016) that the upper bound time complexity for the variable-bound TPTL monitoring algorithm is \( O(|\phi| \times |\hat{\rho}|^2) \), which is polynomial. Additionally, the upper bound space complexity of the presented algorithm by Dokhanchi et al. (2016) is \( O(|\phi| \times |\hat{\rho}|) \). 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 precede the freeze time operator. The second addition is the production rule \( \Theta \). We call this extension of the TPTL language the Quantifiable TPTL (QTPTL). Next is the production rule \( T \) that produces purely spatio-temporal formulas \( (\Pi \mathbb{I} \mathbb{I} T \) 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 \( (S_{obj})^{S_d} \) 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 \( O((S_{obj})^{S_d} \times |V| \times |\phi| \times |\hat{\rho}|^2) \) and \( O((S_{obj})^{S_d} \times |\phi| \times |\hat{\rho}|) \), respectively.

Finally, we consider spatial and spatio-temporal subformulas denoted as \( \psi \), in addition to temporal ones denoted as \( \phi \), to do complexity analysis of the STPL algorithm. It is easy to see that the production rule \( T \) 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: 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, for example, Bournez et al. (1999). Assume that we can decompose our d-dimensional topological space (dimension is fixed to be 2D or 3D) into \( r^c \) cells, where \( r \) is the constant resolution along each axis. Construction of an image Quadtrees/Octrees 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 \( \varphi_s \), 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 \( \varphi_s \) 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.
- \( O(|\hat{\rho}| \times |\varphi_s| \times K \times c) \), if there are time/frame intervals in the formula and no frozen object variable is used in the formula. Here, \( c \) is defined as

\[
  c = \max_{0 \leq j \leq |I|, I \in T(\phi)} |j, max, J(j, I) |
\]

where \( T(\phi) \) contains all the timing constraints \( I \) which are either attached on the temporal operators in formula \( \phi \), or are timing constraints of the form \( t - x \sim n \) or \( F \sim x \sim n \) in the scope of a temporal operator in \( \phi \). For example, if \( \phi = 2[0.1, \infty) \), then \([0.1, \infty) \in T(\phi) \), or if \( \phi = x, t \in (\tau - x \leq 5.5 \Rightarrow \psi) \), then \([0, 5.5] \in T(\phi) \).

Intuitively, the function \( J \) returns all the samples that satisfy a constraint \( I \) from the set \( T(\phi) \) starting from a sample \( j \). Formally,

\[
  J(j, I) = \begin{cases} 
  t^{-1}((t(j)+K) \cap \tau I) & \text{if } \sup I = +\infty \\
  t^{-1}(t(j)+K) & \text{otherwise}
  \end{cases}
\]

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

- \( O(|\hat{\rho}| \times |\varphi_s| \times K \times |V|) \), if there is no time/frame intervals in the formula, but there are frozen object variables used in the formula.
- \( O(|\hat{\rho}| \times |\varphi_s| \times K \times c \times |V|) \), 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 \( O((S_{obj})^{S_d} \times |V| \times |\phi| \times |\rho|^2 \times K \times c) \) and \( O((S_{obj})^{S_d} \times |\phi| \times |\rho| \times K) \), 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: 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 $O(\sum_{i=0}^{\infty} (|\phi_i| + |\overline{\phi}|))$ 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.

### 6.6. 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 $\phi_s$ is bounded.

**6.6.1. Example.** The complexities of evaluating an arbitrary SPE formula $\phi_s$ (the formula is in conjunctive form) on a data stream $\overline{\rho}$ is

- $O(2^n 3^b)$ for time and space, if there is no complement operator in the formula, where $a, b \in \mathbb{N}$, and we have: $\text{argmax}(2^n 3^b | \max(|\phi_s| + |\overline{\phi}|) = 2a + 3b)$. For instance, $\phi_s := (T \sqcup T \sqcup T) \sqcap (T \sqcup T \sqcup T)$.  
- $O(4^{(|\phi_s|+1)/3})$ for time and space, if there are complement operators in the NNF formula. For instance, $\phi_s := \overline{T} \cap \overline{T} \cap \overline{T} \cap \overline{T} \cap \overline{T} \cap \overline{T} \cap \overline{T}$.

#### 6.6.2. Example. An arbitrary l-level nested spatial globally formula $\phi_l^g_s$, with a spatial term as its right-most subformula, is of $O(|\overline{\rho}|)$ time and $O(1)$ space complexity.

$$\phi_l^g_s := c_{1_l}^f (T \cap c_{2_l}^f (T \cap c_{3_l}^f (T \cap T)))$$

#### 6.6.3. Example. An arbitrary l-level nested spatial eventually formula $\phi_l^e_s$, with a spatial term as its right-most subformula, is of $O(|\overline{\rho}|^{l+1})$ time and space complexity.

$$\phi_l^e_s := c_{1_l}^f (T \sqcup c_{2_l}^f (T \sqcup c_{3_l}^f (T \sqcup T)))$$

#### 6.6.4. Example. An arbitrary l-level nested spatial eventually subformula $\phi_l^e_s$ followed by an arbitrary k-level nested spatial globally subformula $\phi_k^g_s$, with a spatial term as its right-most subformula, is of $O(|\overline{\rho}|^{l+1})$ time and space complexity.

$$\phi_l^e_s := c_{1_l}^f (T \sqcup c_{2_l}^f (T \sqcup c_{3_l}^f (T \sqcup T)))$$

#### 6.6.5. Example. A right-hand-side l-level nested spatial until formula $\phi_l^u_s$, where the left-hand-side of all the until operators are a single spatial term, is of $O(|\overline{\rho}|^{l+2})$.

$$\phi_l^u_s := T \sqcup c_{1_l}^f (T \sqcup c_{2_l}^f (T \sqcup T \sqcup T \sqcup T \sqcup T \sqcup T))$$

#### 6.6.6. Example. A left-hand-side l-level nested spatial release formula $\phi_l^r_s$, where the right-hand-side of all the until operators are a single spatial term, is of $O(|\overline{\rho}|^{l+2})$.

$$\phi_l^r_s := ((T \sqcup R_l^s T \sqcup T \sqcup T \sqcup T) \sqcap T \sqcap T) \sqcap T$$

### 7. 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 (Yu et al. (2020)). As indicated in the last column of Table 3, 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 3. We used a Windows 10 machine with Intel Core i7 CPU 8550U @ 1.8GHZ, 16 GB RAM, and Matlab R2020a. The STPL monitoring algorithm is implemented in C language and compiled for Matlab.

The formulas in Table 3 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., $(S_{ab})^{5\nu}$) 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 6.5.

In order to study the impact of the number of objects in each frame on the monitoring algorithm, we manipulated the BDD dataset to create artificial datasets with specific number of objects. The result is presented in Table 4. 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 \( O(\text{Obj}^2 \cdot \text{Id}) \), 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 past-time 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 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 8 for a brief discussion. We plan to pursue such a direction in the future.

### Table 3. Statistics on execution time for different formulas and data stream sizes.

| \( |\phi| \) | \( |\phi_s| \) | \( |\phi_t| \) | \( |\Id| \) | \( S_{\text{obj}} \) | \( S_{\text{id}} \) | 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 |
| 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-formed STPL Formula (14) with SPE operators |
| 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 |
| Quantifier-formed STPL Formula (15) with STE operators |
| 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-formed STPL Formula (16) with spatial terms |
| 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-formed STPL Formula (17) without spatial terms |
| 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 |

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.

### 8. Related works

In this section, we provide a detailed overview of related works. We primarily focus on comparing STPL with other...
Table 4. Statistics on execution time for the Formula (16) on an artificial perception data stream.

| $|\rho|$ | $|\varphi|$ | $|\psi|$ | $|V_i|$ | $S_{obj}$ | $S_d$ | 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 |

Table 5. Overview comparison of spatio-temporal formal languages.

| Language | Temporal foundation | $\exists \forall$ (data) | $\exists \forall$ (spatial) | Spatial foundation | Domain in practice | Applications |
|----------|---------------------|-------------------------|-------------------------|-----------------|-----------------|-------------|
| STPL     | AAN-TPTL            | ✓                       | ✓                       | $S^4_u$          | Sets in Euclidean spaces | Perception systems |
| TQTL     | AAN-TPTL            | ✓                       | ✓                       | Predicates       | Perception data | Vision based (2D) perception |
| SpRE     | FSM                 | ✓                       | ✓                       | $S^4_u$          | Sets in Euclidean spaces | Perception systems |
| STSL     | STL                 | ✓                       | ✓                       | $S^4_u$          | Distances in Euclidean spaces | System level requirements for CPS |
| SpaTeL   | STL                 | ✓                       | ✓                       | TSSL             | Quadrants      | Pattern recognition |
| SSTL     | STL                 | ✓                       | ✓                       | Graph-based modeling | Distances in Euclidean spaces | Pattern recognition |
| GSTL     | STL                 | ✓                       | ✓                       | Mereotopology    | Cubics         | Knowledge representation |

The $\exists \forall$ (data) column indicates whether the logic supports quantification over non-spatial data. The $\exists \forall$ (spatial column indicates whether the logic supports some form of spatial quantification, for example, emptiness operator (i.e., there exists a point), or directional operator (there exists a direction). FSM: Finite state machines.

spatio-temporal logics. In Table 5, 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, for example, 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 graph-based representation of discrete models of space. For the SSTL, undirected weighted graphs are used to model space, 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 $\rho_{[w_1,w_2]} \varphi$ and $\rho_1 S_{[w_1,w_2]} \varphi_2$ into STL (Maler and Nickovic (2004)), which are named bounded somewhere and bounded surround, respectively. The first operator requires $\varphi$ 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 $\varphi_1$, and for all the locations in the external boundary of $A$, they satisfy the formula $\varphi_2$. 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 quadrees 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, $\exists \varphi_1 \cup_k \varphi_2$ means that there exists a set of directions in $B$ by which the $i$-th label of a path $x^k$ satisfies the formula $\varphi_2$ and all the other labels on the path until $i$ satisfy the formula $\varphi_1$. 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 as follows:

- **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 quadrates 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 decidability reasons.

The works closest to ours stem from combining temporal logics with spatial logics (for a historical overview and a discussion on $\mathcal{S}_4$ see Kontchakov et al. (2007)). Gabelaia et al. (2005) combine LTL (Manna and Pnueli (1992)) with $\mathcal{S}_4$ to define the logic $\mathcal{PTL} \times \mathcal{S}_4$. They further define several fragments of $\mathcal{PTL} \times \mathcal{S}_4$ 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 $\mathcal{S}_4$. 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. 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, for example, 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 than STL. Note that further algorithmic improvements on AAN-TPTL are possible, for example, see Elgyutt et al. (2018); Ghorbel and Prabhu (2022). Nevertheless, TQTL cannot reason directly about properties of bounding boxes. For example, TQTL cannot reason about self-overlap of bounding boxes across time, that is, 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 $S_4$ 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.

9. Conclusions

In this paper, we presented 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 $S_4$, (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 been working toward developing a 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 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 Chakrabarti (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.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the NSF under grants CNS-2039087, CNS-2038666, IIP-1361926, and the NSF I/UCRC Center for Embedded Systems.

ORCID iD

Mohammad Hekmatnejad  𝐷 https://orcid.org/0000-0003-0110-0006

Notes

1. For a historical introduction to $S_4$, see Van Benthem and Bezhanishvili (2007) and Kontchakov et al. (2007).
2. https://www.cvlibs.net/datasets/kitti/
3. Training data: https://www.cvlibs.net/datasets/kitti/eval_tracking.php, and data format: https://github.com/JonathanLuiten/TrackEval/blob/master/docs/KITTI-format.txt
4. https://www.nuscenes.org/nuscenes?sceneId=scene-0274&frame=0&view=lidar
5. https://bdd-data.berkeley.edu/

References

Abbas H, O’Kelly ME, Rodionova A, et al. (2017) A driver’s license test for driverless vehicles. Mechanical Engineering 139: S13–S16.
Fremont DJ, Yue D, Dreossi T, et al. (2018) Scenic: Language-Based Scene Generation. arXiv:1809.09310.

Fremont DJ, Kim E, Punt YV, et al. (2020) Formal scenario-based testing of autonomous vehicles: from simulation to the real world. In: 23rd IEEE International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020.

Fuggitti F and Chakraborti T (2023) NL2TLD – a Python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. Proceedings of the AAAI Conference on Artificial Intelligence 37(13): 16428–16430.

Gabelata D, Kontchakov R, Kurucz A, et al. (2005) Combining spatial and temporal logics: expressiveness vs. complexity. Journal of Artificial Intelligence Research 23: 167–243.

Geiger A, Lenz P, Stiller C, et al. (2013a) Vision meets robotics: the kitti dataset. The International Journal of Robotics Research 32(11): 1231–1237.

Geiger A, Lenz P, Stiller C, et al. (2013b) Vision meets robotics: the KITTI dataset. The International Journal of Robotics Research 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), Milan, 4–6 May 2022.

Gladisch C, Heinz T, Heinzemann C, et al. (2019) Experience paper: search-based testing in automated driving control applications. In: 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, 11–15 November 2019.

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, Los Angeles, CA, 15–17 December 2014.

Gordon D, Farhadi A and Fox D (2018) 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, et al. (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, Seattle, WA, 14–16 April 2015.

Havelund K, Peled D and Ulus3 D (2020) First-order Temporal Logic Monitoring with Bdds 56.

He K, Lahijanian M, Kavraki LE, et al. (2015) Through manipulation planning with temporal logic specifications. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, 26–30 May 2015.

He K, Lahijanian M, Kavraki LE, et al. (2018) Automated abstraction of manipulation domains for cost-based reactive synthesis. IEEE Robotics and Automation Letters 4(2): 285–292.

Hekmatnejad M, Yaghoubi S, Dokhanchi A, et al. (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), San Diego, CA, 9–11 October 2019.

Huth M and Ryan M (2004) Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge, UK: Cambridge University Press.

Kim E, Shenoy J, Junges S, et al. (2022) Querying labelled data with scenario programs for sim-to-real validation. In: ACM/IEEE 13th International Conference on Cyber-Physical Systems (IC CPS), Milan, Italy, 4–6 May 2022.

Kontchakov R, Kurucz A, Wolter F, et al. (2007) Handbook of Spatial Logics: Spatial Logic + Temporal Logic = ? Springer, 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), Anchorage, 3–8 May 2010.

Lee TB (2018) Report: software bug led to death in Uber’s self-driving crash. Ars Technica.

Li T, Liu J, Sun H, et al. (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), Shanghai, China, 4–6 September 2013.

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, Springer, 6664, 42–56.

Ma M, Bartocci E, Lifland E, et al. (2020) SaSTL: spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: ACM/IEEE 11th International Conference on Cyber-Physical Systems (IC CPS), Sydney, Australia, 22–24 April 2020.

Maler O and Nickovic D (2004) Monitoring temporal properties of continuous signals. Proceedings of FORMATS-FTRTFT, LNCS 3253: 152–166.

Mallick S, Ghosal S, Balakrishnan A, et al. (2023) Safety monitoring for pedestrian detection in adverse conditions. In: 23rd International Conference on Runtime Verification, Thessaloniki, Greece, 3–6 October 2023.

Manna Z and Pruefi A (1992) The Temporal Logic of Reactive and Concurrent Systems — Specification. New York, NY: 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, et al. (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. Available at: https://www.nuscenes.org/nuscenes (Accessed 14 November 2020)

Nenzi L, Bortolussi L, Ciancia V, et al. (2015) Qualitative and quantitative monitoring of spatio-temporal properties. In: Runtime Verification. Cham, Switzerland: Springer, 37–53.

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), London, UK, 29 May–2 June 2023.

Perugini S (2021) Programming Languages: Concepts and Implementation. Burlington, MA: Jones & Bartlett Learning.
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 10.1. STPL AAN Syntax for Discrete-Time Signal. Let \( V_r \) and \( V_s \) be sets of time variables and object variables, respectively. Let \( x \) be a vector of time variables, that is, \( x = [x_0, \ldots, x_{n-1}]^T \), and \( id \) be a vector of object variables, that is, \( id = [id_0, \ldots, id_{m-1}]^T \), and \( T \) 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:

\[
\Phi_{ij} := \exists id_i \land x_i \Phi_i | x_i \Phi_i' | \exists id_i \Phi_{ij}'
\]
\[
\begin{align*}
T \mid \neg \Phi_{ij} | \Phi_{ij} \lor \Phi_{ij}' | \lor \Phi_{ij} \land \land \Phi_{ij} | \\
\bigcirc \Phi_{ij} | \Phi_{ij} \check{U}_T \Phi_{ij} | \Phi_{ij} \check{U}_T \Phi_{ij}'
\end{align*}
\]
\[
\Phi_i' := \tau - x_i > t \mid x_i > n | \Phi_{i+1,j} | \\
\begin{align*}
T \mid \neg \Phi_i' | \Phi_i' \lor \Phi_i' | \lor \Phi_i' \land \land \Phi_i' | \\
\bigcirc \Phi_i' | \Phi_i' \check{U}_T \Phi_i' | \Phi_i' \check{U}_T \Phi_i'
\end{align*}
\]
\[
\Phi_{ij}' := C(A_{ij}) = c \mid C(A_{ij}) = C(A_{ij}) \mid P(A_{ij}) \geq r
\]
\[
\begin{align*}
T \mid \neg \Phi_{ij}' | \Phi_{ij}' \lor \Phi_{ij}' | \lor \Phi_{ij}' \land \land \Phi_{ij}' | \\
\bigcirc \Phi_{ij}' | \Phi_{ij}' \check{U}_T \Phi_{ij}' | \Phi_{ij}' \check{U}_T \Phi_{ij}'
\end{align*}
\]

\[
\begin{align*}
\Box_{ij} | \Theta_{ij} | \Pi_{ij}
\end{align*}
\]

\[
\Lambda_{ij} := id_j | id_{j+1} | \ldots | id_l
\]

\[
\begin{align*}
\Omega_{ij} := \sigma(A_{ij}) \lor [Z] U | \Gamma_{ij} | \Omega_{ij} \land \Omega_{ij} | I \Omega_{ij} | \\
\Omega_{ij} \cup T \Omega_{ij} | \check{\Omega}_{ij} | \check{\Omega}_{ij} \cup T \Omega_{ij} | \check{\Omega}_{ij} \land \Omega_{ij} | \\
\Omega_{ij} \check{U}_T \Omega_{ij} | \check{\Omega}_{ij} \check{U}_T \Omega_{ij} | \check{\Omega}_{ij} \check{U}_T \Omega_{ij} | \check{\Omega}_{ij} \check{U}_T \Omega_{ij}
\end{align*}
\]

\[
\begin{align*}
\Omega_{ij} := Yurtsever E, Lambert J, Carballo A, et al. (2020) A survey of autonomous driving: common practices and emerging technologies. IEEE Access 8: 58443–58469.
\end{align*}
\]

Qin B, Chong ZJ, Soh SH, et al. (2016) A spatial-temporal approach for moving object recognition with 2d lidar. In: Experimental Robotics. Berlin, Germany: Springer, 807–820.

Richter SR, Hayder Z and Koltnv V (2017) Playing for benchmarks. IEEE International Conference on Computer Vision ICCV: 2232–2241.

Rizaldi A, Keinholz J, Huber M, et al. (2017) Formalising and monitoring traffic rules for autonomous vehicles in isabelle/hol. In: Integrated Formal Methods. Cham, Switzerland: Springer, 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.

Shneider 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. Boston: Thompson 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, Dublin, Ireland, 1–4 August 2014.

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), Macao, 3–8 November 2019.

Templeton B (2020) Tesla in Taiwan crashes directly into overturned truck, ignores pedestrian, with autopilot on.

Tuncali CE, Fainekos G, Prokhorov D, et al. (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. 261–298.

Wu B, Iandola F, Jin PH, et al. (2017) Squeezenet: 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, Honolulu, 21–26 July 2017.

Yadav P and Curry E (2019) Videcp: complex event processing framework to detect spatiotemporal patterns in video streams. In: IEEE International Conference on Big Data, Los Angeles, CA, 9–12 December 2019.

Yu F, Chen H, Wang X, et al. (2020) Bdd100k: a diverse driving dataset for heterogeneous multitask learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, 13–19 June 2020.

Yurtssever E, Lambert J, Carballo A, et al. (2020) A survey of autonomous driving: common practices and emerging technologies. IEEE Access 8: 58443–58469.
\[ \Pi_{ij} := \text{Area}(\Omega_{ij}) \geq r \mid \text{Area}(\Omega_{ij}) \geq r \times \text{Area}(\Omega_{ij}) \]

\[ \Theta_{ij} := \text{Dist}(\Lambda_{ij}, \text{CRT}, \Lambda_{ij}, \text{CRT}) \geq r \mid \text{Lat}(\Lambda_{ij}, \text{CRT}) \geq r \mid \text{Lon}(\Lambda_{ij}, \text{CRT}) \geq r \mid \text{Lon}(\Lambda_{ij}, \text{CRT}) \geq r \times \text{Lat}(\Lambda_{ij}, \text{CRT}) \mid \text{Lon}(\Lambda_{ij}, \text{CRT}) \geq r \times \text{Lon}(\Lambda_{ij}, \text{CRT}) \mid \text{Area}(\Lambda_{ij}) \geq r \times \text{Area}(\Lambda_{ij}) \]

\[ \text{CRT} := \text{LM} \mid \text{RM} \mid \text{TM} \mid \text{BM} \mid \text{CT} \]

where \( i \geq 0 \), 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 \( F = x > n \), respectively. The freeze time quantifier \( x.\Phi \) assigns the current frame \( i \) to time variable \( x \) before processing the subformula \( \Phi \). The Existential quantifier is denoted as \( \exists \), and the Universal quantifier is denoted as \( \forall \). The following syntactic equivalences hold for the STPL formulas \( \psi \) and \( \Phi \) using syntactic manipulation. \( \forall \{id\}@x.\Phi \equiv (\exists \{id\}@x.\Phi) \), \( \exists \Phi \equiv (\exists \Phi \land \psi) \land \neg \exists \Phi \), \( \psi \land \neg \exists \Phi \equiv (\exists \Phi \land \psi) \land \neg \exists \Phi \).

For parsing a formula using the above grammar, there are two production rules \( \Phi^i \) and \( \Phi^f \), in which we can use the initial production rule after increasing the index \( i=0,i+1,i+2 \). 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 \( i \) 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 \( \phi = \exists id_0. (\exists id_1. (\exists id_2. (\phi_1 \lor \phi_2)) \lor \phi_3) \), we have \( i = 2 \) and \( j = 0 \) while parsing the subformula \( \phi_1 \), whereas in \( \phi_2 \), we have \( i = 0 \) and \( j = 0 \). Thus any function in \( \phi_1 \) with object variables in it will use the production rule \( \Lambda_{2,0} \). Thus, the allowed object variables in \( \phi_1 \) are \( id_0, id_1 \) and \( id_2 \). However, while parsing the subformula \( \phi_2 \), we use \( \Lambda_{0,0} \) in which the only allowed object variable is \( id_0 \).

### B Appendix: Complexity Analysis of STE formulas

Here the assumption is that we use the linked-list data structure to represent a spatial term \( T \) as a union of a finite number of unique subsets. We can compute \( V(t_1, U_{t_2}^T t_2, D, t, t, t, \tau, \epsilon, \zeta) \) recursively as \( V(t_1, D, t, t, t, \tau, \epsilon, \zeta) \cup V(t_2, D, t, t, t, \tau, \epsilon, \zeta) \cap V((t_1, U_{t_2}^T t_2), D, t+1, t, t, \tau, \epsilon, \zeta) \). For each until formula in each row of Table 6 (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 \( t_1 \cup t_2 \) is equal to the total number of boxes in the two spatial terms (i.e., \( |t_1| \times |t_2| \)). Additionally, the maximum number of bounding boxes that can be computed by \( t_1 \cap t_2 \) is equal to the product of number of boxes in the two spatial terms (i.e., \( |t_1| \times |t_2| \)). Consequently, the maximum number of bounding boxes that can be produced at each time step \( t \) for the above until formula is \( |V(t_1, U_{t_2}^T t_2, D, t, t, t, \tau, \epsilon, \zeta) \times |V((t_1, U_{t_2}^T t_2), D, t+1, t, t, \tau, \epsilon, \zeta) \)|.

### B.1 Formulas with Exponential Time/Space Complexities

As it is stated in the first and second rows in Table 6, the number of needed operations grow as in arithmetic sequences. 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

| \( l \) | \( t_n \) | \( t_{n-1} \) | \( t_{n-2} \) | \( t_{n-3} \) | \( t_{n-4} \) | ... |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | (1 + 1) = 2 | (1 + 1 + 1) = 3 | (1 + 1 + 1 + 1 + 1) = 4 | (1 + 1 + 1 + 1 + 1 + 1 + 1) = 5 |
| 2 | 2 | (1 + 1) = 4 | (1 + 1 + 1 + 1) = 6 | (1 + 1 + 1 + 1) = 8 | (1 + 1 + 1 + 1) = 10 |
| 3 | 3 | (1 + 1) = 8 | (1 + 1 + 1 + 1) = 9 | (1 + 1 + 1 + 1) = 6 | (1 + 1 + 1 + 1) = 4 |

At \( l = 0 \) we have \( \tau = \sigma(id) \); at \( l = 1 \) we have \( \tau = U_{t_2}^T r, \) and \( \tau = t_1 r \); at \( l = 2 \) we have \( (\tau U_{t_2}^T r, \tau U_{t_2}^T r) \); \( \tau U_{t_2}^T r \); at \( l = 3 \) we have \( ((\tau U_{t_2}^T r) \tau U_{t_2}^T r, \tau U_{t_2}^T r) \).
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 \( (\tau \ U^2_2 \ t) \ U^2_2 \ (\tau \ U^2_2 \ t) \).

\[
\sum_{t=1}^{n} f_2(t) = 1 + 2(1 + 1) + 3(1 + 2(1 + 1)) + 4(1 + 3(1 + 2(1 + 1))) + 5(1 + 4(1 + 3(1 + 2(1 + 1)))) + \ldots + n(1 + (n - 1)(1 + (n - 2)(\ldots + 2(1 + 1))))) \ldots
\]

(21)

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

\[
\sum_{t=1}^{n} f_2(t) < 1 + 3! + 4! + 5! + 6! + \ldots + (n + 1)! \quad (22)
\]

where \( n \geq 2 \). Therefore, we have

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

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 \( 2^{(n+1)} < a(1 + 2^n) \) for \( a \geq 2 \) to derive the following inequality from Eq. (21)

\[
\sum_{t=1}^{n} f_2(t) > 1 + 2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^{(n-1)} \quad (24)
\]

where \( n \geq 2 \). Therefore, we have

\[
\sum_{t=1}^{n} f_2(t) > 2^n \quad (25)
\]

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

### B.2 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 \( a^{r+1} < a'(l + b) \) for \( b > a \) in each summation of the elements of rows to derive the below inequality

\[
\sum_{t=1}^{n} f_0(t) + \sum_{t=1}^{n} f_1(t) + \ldots + \sum_{t=1}^{n} f_l(t) >
\]

\[
n + \frac{n(n + 1)}{2} + \frac{2^n - 1}{2 - 1} + \frac{3^n - 1}{3 - 1} + \ldots + \frac{p^n - 1}{l - 1} >
\]

\[
n + \frac{n(n + 1)}{2} + 2^{(n-1)} + 3^{(n-1)} + \ldots + l^{(n-1)}
\]

\[
- \left( 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{l-1} \right)
\]

\[
> 2^{(n-1)} + 3^{(n-1)} + \ldots + l^{(n-1)}
\]

where \( n \geq 2 \). Therefore, we have

\[
\sum_{t=1}^{n} f_0(t) + \sum_{t=1}^{n} f_1(t) + \ldots + \sum_{t=1}^{n} f_l(t) > f^{(n-1)} \quad (26)
\]

This concludes the complexity of the algorithm to be

\[\Omega(|\varphi_x|^{(l-1)})\].