Traffic Sequence Charts – From Visualization to Semantics

by

Werner Damm, Stephanie Kemper, Eike Möhlmann, Thomas Peikenkamp, Astrid Rakow
We present a formalism for scenario based specification via Snapshot Charts (SCs). Snapshot Charts (SCs) are a specification formalism within Traffic Sequence Charts (TSCs), where Snapshot Charts describe the continuous evolution and Live Sequence Charts (LSCs) are used to describe the communication protocols.

A snapshot can be thought of as a description of the current state. Like pages in a flip book, we can place snapshots in a sequel to tell a story about dynamic objects. Here, a single snapshot describes invariant properties of a sequel of time contiguous states that hold until the next snapshot.

In the following, we describe the syntax and semantics of Snapshot Charts (SCs).

Keywords:
formal specification language, highly autonomous driving, scenario catalog

1 Introduction

It is well-known that traditional approaches for type homologation fail for highly autonomous vehicles due to the impossibility of covering sufficiently many kilometers in field testing to achieve a statistically valid basis for building safety cases. This is due to the extreme variability of environmental contexts and the resulting complexity in, both, the perception- and trajectory planning systems of highly autonomous vehicles. The approach followed by the German automotive industry builds on scenario catalogs to capture for all perceivable traffic situations’ requirements on such systems jointly ensuring global safety objectives. Test drives are to be replaced to a significant extend by placing the vehicle under test in (semi-)virtual test environments exposing the vehicle to traffic situations covering all scenarios of
the scenario catalog, and monitoring compliance of the vehicle’s reaction to such scenarios. Such test environments will allow testing separately the perception components (along all stages covering preprocessed sensor data, sensor fusion, object identification algorithms) and the trajectory planning component (which involves exploring possible future evolutions of the currently perceived traffic situation to decide on the planned maneuver). Projects already running and pushing this approach are the PEGASUS project[^1] funded by the German Federal Ministry for Economic Affairs and Energy, involving all major German OEMs and Tier 1 companies, and the ENABLE-S3 project[^2] funded by the Joint Undertaking ECSEL, including, both, German and French automotive companies, and covering additionally other domains for building test environments for autonomous systems, such as maritime and rail. OFFIS participates in both these projects and is involved in the planning of follow-up projects pushing a fast implementation of this approach.

There are several challenges which must be addressed to make this approach viable:

(C1) Given the space of real world traffic situations, how can we achieve completeness of scenario catalogs, i.e. demonstrate with high confidence that all relevant real-world situations have been captured?

(C2) Given the remaining likelihood of experiencing failures in perception and interpretation after deployment, how can we establish a process learning from field incidents and accidents leading to updates of the scenario catalog avoiding re-occurrence of this incident in the field?

(C3) Given the complexity of real-world traffic situations, how can one at all achieve sufficiently concise specifications to make construction of scenario catalogs viable?

(C4) How can we assure, that the interpretation of scenarios and thus interpretation of test results is unambiguous across all test platforms?

All these challenges can only be addressed, if using a language for capturing scenarios, which is intuitively easy to understand, and, most prominently, which is equipped with a formal (declarative) semantics.

Challenge [C1] will be addressed by generalizing from data bases of observed traffic flows. A minimal requirement for checking for completeness is thus the need to formally define, whether a particular observed traffic behavior is already covered by the current scenario catalog or not. This requires the definition of a formal satisfaction relation. Moreover, as experienced in the play-out approach for Live Sequence Charts (LSCs)[^7], a formal semantics provides a basis for playing out the current scenario catalog, hence, generating traffic flows which in an expert can judge for unrealistic or missing real-life traffic flows.

Challenge [C2] requires a formal semantics to identify the gaps between the space of possible worlds described in the scenario catalog, and the concrete in-field incident or accident. Specifically, forthcoming regulations will require autonomously driving cars to record all those perceived environmental artifacts relevant to trajectory planning as well as the car’s trajectory control for a sufficiently long time-period. A formal semantics allows to check the failed scenario(s), offering a basis for refining the scenario specifications to cope with the observed failures in perception or interpretation of the real world.

[^1]: www.pegasusprojekt.de
[^2]: www.enable-s3.eu
Challenge C3 demands the use of a declarative specification language, where one single scenario specification stands for a possibly extremely large set of real world traffic situations, defined unambiguously through the satisfaction relation. Also, declarative specification languages allow for separation of concerns, such as focusing on particular kinds of critical situations in isolation, knowing that the car can only pass the test if all scenarios passed.

Finally, Challenge C4 can be addressed by automatically synthesizing monitors for compliance testing, using the standardized formal semantics.

This paper provides a formal semantics for the declarative visual specification language of traffic sequence charts (henceforth called TSCs), and thus meets a key industrial need. Not surprisingly, this comes with a number of scientific challenges outlined below, which we address by building on a number of previous publications, notably our previous work on introducing LSCs [1], and on automatic synthesis of driving strategies for autonomous vehicles [4].

Much as Message Sequence Charts [10] were lacking expressiveness and formal semantics, motivating the extension to Live Sequence Charts, the ongoing industrial pre-standardization effort for capturing scenarios, called OpenSCENARIO [17], falls significantly short in being able to address the above challenges. OpenSCENARIO allows describing what we called existential LSCs, i.e. give examples of desired behaviors, rather than being able to specify requirements on all behaviors, such as in what we called universal LSCs. TSCs “inherit” from LSCs the concepts related to distinguishing between possible and mandatory behaviors, the concepts of pre-charts which is key for characterizing those situations from when on all behaviors must comply to universal charts, and cold and hot conditions for distinguishing case-distinctions from failures. They go beyond LSCs in

- providing a visual specification language for describing first-order predicates on traffic situations,
- introducing a dedicated specification pattern reflecting the need to make current moves dependent on the future evolutions of traffic flows,
- coping with a priori unbounded number of traffic participants, such as [5],
- reflecting dynamic evolutions of traffic scenarios governed by complex vehicle dynamic models of the ego car, which depend on road surface conditions and dynamic models of other traffic participants.

In this paper, we focus on specifying requirements to be achieved by a single car, the ego car, without further assistance of cooperating vehicles. So we do not specify communications within TSCs for now and instead consider only TSCs that consist of so-called snapshot charts. For a paper motivating the language design of TSCs, we refer to [3].

Outline In the next section, we give a short overview of TSC notions. We give an introductory example in Section 3. In Section 4 we introduce the formal notions. An overview of the elements at a single snapshot is given in Section 5. Then we show how a snapshot is translated to a multi-sorted first-order formula in Section 6.

Snapshots are combined to build Snapshot Graphs (SGs). Snapshot Graphs, annotated with timing constraints, constitute Snapshot Charts (SCs). We introduce Snapshot Graphs and Snapshot Charts in Section 7. In Section 8 we explain how a corresponding real-time formula is derived by composing the snapshot formulas as obtained by translating the individual snapshots. In Section 9 we introduce TSC headers and derive a formula of the overall TSC. An implementation relation is defined in Section 10. Finally, we survey related work in Section 11.
2 Overview

In this section, we give an overview of the conceptual elements of a TSC specification.

2.1 Traffic Sequence Charts (TSCs)

TSCs are a visual formalism that combines two formalisms synergetically:

- Snapshot Charts (SCs) – introduced in this paper – focus on the visual specification of the continuous evolution at a scenario, while
- LSCs support the visual specification of communications between entities.

A TSC consists of a header, an SC and optionally an LSC part and a world model (cf. 1). In this paper we will present a special form of TSCs that do not have an LSC part. A TSC specification consists of a collection of TSCs that conjunctively constrain the behavior of the considered world model, which is discussed in more detail later in this section. The connection between visual symbols and objects of the world model is specified at the symbol dictionary.

2.2 Live Sequence Charts (LSCs)

LSCs [1] are an optional part of a TSC specification. They are used to specify communication protocols between the traffic participants. A detailed description can be found in [1] and we omit the details here since we did not include communication in this report.

2.3 Snapshot Charts (SCs)

SCs arrange snapshots within a directed acyclic graph. These snapshot graphs are additionally annotated by timing and synchronization constraints along a path and may be used to specify premise-consequence rules. A single snapshot encodes a multi-sorted first-order predicate on the world model. We can translate an SC to a first-order multi-sorted real-time formula, by composing the snapshot formulas —the formulas obtained by translating the individual snapshots— according to the graph’s structure and annotations.

2.4 Visualizations and the Spatial View

TSCs are a declarative specification formalism that allows expressing constraints within a visual language. We present a highway example in the next section on page 6. As different aspects of a scenario are important, TSCs allow to visualize these aspects in different visualization formalisms.

In the following, we discuss one of these visualization formalisms, namely, the spatial view. At the spatial view, the relative placement of symbols within a snapshot encodes the relative placement of represented objects. So for a symbol a dedicated anchor point is declared that corresponds to a dedicated anchor point of the represented object. The relative positioning of anchors within the $\mathbb{R} \times \mathbb{R}$ snapshot is transferred to relative positioning of the anchors of the represented objects within the world model for which also a $\mathbb{R} \times \mathbb{R}$ coordinate system is assumed. Sizes and containment relations can also be described at the spatial view. Such spatial visualizations can be helpful for the specification of collision freedom or a successful

\[3\] We restrict ourselves here to two dimensions. As outlook we sketch an example of three dimensions on p. 29
Parking maneuver. We imagine that also other visualizations are useful and can be easily integrated into TSCs, but these have not been designed yet.

### 2.5 World Model

The world model is a formal model, that specifies the assumed application domain.

A TSC is interpreted wrt. a world model. The world model defines classes of objects (cars, bikes, ...) with a set of attributes (position, size, velocity, ...) and the dynamics of moving objects. Each object belongs to one of finitely many classes. The number of objects might be unbounded.

TSCs restrict the overall possible behaviors of the world model objects (to behaviors satisfying the TSCs). The symbols and predicate annotations used in the snapshots refer to the objects and their states within the world model. That way a TSC specification defines the set of trajectories that arise from the world model and satisfy the predicates defined via (the conjunction of) TSCs.

Having a formal world model provides the basis for a wide range of automated analysis techniques, as such model-checking (e.g. analytically answer, whether a specified scenario is possible within the world model) and test-case generation (e.g. simulate trajectories for each scenario).

### 2.6 Symbol Dictionary

A TSC specification consists of a world model, symbol dictionary and a set of TSCs. The symbol dictionary links the visual symbols—e.g., —used in TSCs to the world model objects (objects of class Car) and it declares symbol properties like equivalences (i.e. "Are symbols different if they are rotated, scaled or of changed color?").

To link symbol positions to object positions, an anchor has to be defined for the symbol and the correspondence to the object’s anchor has to be declared, both in the symbol dictionary. The anchors are central for the definition of the spatial view (cf. Section 2.4).

The symbol dictionary also defines for a symbol modification (e.g., ) which object features are represented (the car is indicating to its left).
3 Example

To give an impression of the visual specification via TSCs, we sketch a collision avoidance maneuver. We consider two lanes of the same direction at a highway, further, we consider car objects and obstacle objects (objects of low or zero velocity, e.g., a construction site or a slowly moving vehicle). We do not discuss the underlying world model in more detail here and assume an appropriate world model to be given. Instead, we focus on the SC part of the specification.

We examine the scenarios that may arise when a car that drives in the right lane approaches an obstacle.

We first structure the space of possible scenarios arising in that situation. There are two basic scenarios: either the car stays in the right lane and collides with the obstacle, or it changes lane and avoids the collision. Figure 2 shows a TSC that specifies a collision scenario. TSCs are to be read from left to right. Figure 2 consists of a header followed by three snapshots (sn). Sn1 (black frame with gray hatching) is the empty snapshot and specifies that anything is allowed to happen before sn2. Sn2 specifies our initial situation: The car is in the right lane, distance \( \leq d_1 \) away from the obstacle (the black rectangle). The distance line is used to specify bounds on distances between objects. The third snapshot describes a collision between the car and the obstacle. The hatching of the lanes denotes that—for now—we do not constrain whether there are other objects (a so-called "don’t care"). Figure 3 specifies the collision-avoidance scenario. Again, the sequence of sn1, sn2 expresses that eventually sn2 is reached—the car is \( \leq d_1 \) away from the obstacle. Before the car gets closer to the obstacle than \( d_2 \), it starts changing lane (cf. sn3). The dashed somewhere-box surrounding the car indicates that the car may be anywhere within the box. The whole process of changing into the left lane is hence covered by sn3. Sn4 describes that the car has arrived in the left lane and drives past the obstacle. Finally, the last snapshot describes that the car has passed the obstacle. Note, we require snapshots (of a sequence) to continguously hold during a trajectory. Hence the somewhere-boxes in sn3 and sn4 are an important mean to write succinct specifications.

The headers in Figure 2 and Figure 3 declare that both TSCs are to be understood existentially (quantification mode = exists). That is, we specify that the scenarios of these figures exist. Existential TSCs allow cataloging observations of the real world. In contrast, the TSC of Figure 4 specifies the (desired) behavior of ego, the car under design, during a collision avoidance maneuver. It specifies that if ego gets into the situation of sn1—ego is closer than \( d_1 \) to an obstacle—and if the left lane will be free for a time duration greater than \( t \) (cf. sn2), then ego changes to the left lane and drives past the obstacle.
The TSC of Figure 4 uses a premise-consequence chart to express "if ego [...], then ego changes lane [...]". The dashed hexagon contains the premise. The consequence is specified via the snapshots right of the hexagon. Our premise consists of two parts: It specifies the initial situation via $sn_1$ (so the premise expresses "if ego is closer than $d_1$ to the obstacle") and via $sn_3$ the future. $sn_2$ adds to the premise "and if there will be no car on the left lane within a distance of $d_4$ behind ego up to $d_5$ in front of ego". Snapshot $sn_3$ is the True snapshot. It serves a technical matter here. The future ($sn_2; sn_3$) has to have the same time extend as the consequence. $sn_3$ expresses that arbitrary behaviour is allowed after the left lane has been free for $\delta$ time. For more details on premise-consequence charts we confer the reader to Section 7.3 and Section 8.2 . We use the nowhere-box, the black frame with diagonal lines, to denote that we rule out the existence of cars within the box. The extend of the nowhere-box is specified via the distance lines anchoring at ego and the box’s borders. The hourglass on top of $sn_2$ represents a constraint on the dwell-time of the snapshot $sn_3$ and specifies that the left lane will be free for a time duration greater than $t$. The consequence ($sn_4$ to $sn_7$) is much like $sn_3$ to $sn_5$ of Figure 3. The additional annotation of a $\otimes$ bar at its top highlights that the consequence ($sn_4$ to $sn_7$) and the future ($sn_2; sn_3$ abbreviated by $\otimes$) happen at the same time. The hourglass $\rho$ specifies that $sn_4; sn_5$ happen before $t$ time. That is, ego has to perform the lane change while the left lane is guaranteed to be free. Thus, the future snapshot $sn_2$ is concurrent to $sn_4; sn_5$ and ends some time later.

As the activation mode of the TSC of Figure 4 is always and the quantification mode is all, all trajectories have to satisfy the TSC and if at any time the premise matches ("ego is close to an obstacle and the left lane will be free"), the consequence has to hold ("ego changes to the left lane"). The TSC of Figure 4 specifies a very abstract lane change rule—chosen here for simplicity and ease of the example. A TSC for a concrete implementation has to rephrase the future part of the premise of Figure 4 ("the left lane will be free") in terms of sensor readings and on-board prediction so that a sufficiently free corridor is guaranteed.

4 Preliminaries

A TSC specification is a set of TSCs together with a world model over which the TSCs are interpreted according to the symbol dictionary. In this paper we explain the role of the world model, but focus on the SC part of a TSC and do not give a detailed discussion of the world model. The formal semantics of TSCs is given by a translation into a multi-sorted temporal formula of a logic $\mathcal{L}$. In the following we introduce the logics $\mathcal{L}$ and the formalism we consider in this paper to specify the world model.

A Multi-sorted Real Time Logic We consider a multi-sorted first-order real-time logic, which we simply call $\mathcal{L}$ in the sequel. We consider a signature $\Sigma = (\text{Var}, \Pi, \Upsilon, \Gamma, \sigma)$ to be given that comprises a set of variable symbols $\text{Var}$ and a set of predicate symbols $\Pi$, a set of function symbols $\Upsilon$, a set of type symbols $\Gamma$ and a function $\sigma$ that assigns types (sorts) to
variable, predicate and function symbols. We denote the set of type respecting ground terms as $T^\Sigma$.

Since TSCs talk about objects and their attributes at a certain time, we introduce the following distinction of variables, $\text{Var} = \text{Var}_{\text{Obj}} \cup \text{Var}_p$, where $\text{Var}_p$ is a set of so called plain variables and $\text{Var}_{\text{Obj}}$ is a set of object variables $o_i$.

The formulas of $\mathcal{L}$ are inductively defined by the grammar

$$\varphi \equiv \top | q(o_1, \ldots, o_n) | \exists w \varphi(w) | \neg \varphi | \varphi_1 \land \varphi_2 | \varphi_1 \cup \varphi_2 | \varphi_1 [t_1, t_2] \varphi_2 | \tau_1 \geq \tau_2$$

where $q \in \Pi$ is a predicate of arity greater or equal to zero, $o_i$ are object variables, $w$ is an object or plain variable, $t_1, t_2$ are plain variables, $\tau_1, \tau_2 \in T^\Sigma$, $\sigma(t_1) = \sigma(t_2)$ and $\preceq \in \{<, \leq, =, \geq, >\}$. We use the following common abbreviations: “$\varphi_1 \lor \varphi_2$” for “$\neg \varphi_1 \land \neg \varphi_2$”, “$\Diamond \varphi$” for “$\exists w \varphi$”, “$\Box \varphi$” for “$\forall \varphi$” and “$\forall w \varphi$” for “$\exists \varphi$”.

We assume now a structure $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ of $\Sigma$ to be given where the universe $\mathcal{U}$ is a non-empty set of concrete values and $\mathcal{I}$ is an interpretation of the symbols in $\Pi$, $\mathcal{Y}$, $\Gamma$ respecting the typing.

Further, we assume that the local state of each object is made up of its identity and list of attributes $\mathcal{A}(o) = \{a_1, \ldots, a_n\}$ with a fixed but arbitrary order of $a_i$’s. So $\sigma$ assigns to each object variable $o \in \text{Var}_{\text{Obj}}$ a type $T \in \Gamma$, such that a state of an object of type $T$ is a value in $\sigma(id) \times \sigma(a_1) \times \ldots \times \sigma(a_n)$. Given an object variable $o$, we denote $o.id$ to refer to the object’s identity and we use $o.a_i$ to refer to the object’s attribute $a_i$.

The semantics of $\mathcal{L}$ is defined at the end of this section, after the introduction of the semantic model over which we will interpret formulas of $\mathcal{L}$.

**World Model** The TSC semantics is given in terms of formulas of $\mathcal{L}$, that are interpreted on a world model WM.

We assume that the world model is composed of object models and that each object model is an instantiation of finitely many object classes. An object class defines a blue print for the objects via a list of attributes and their dynamics. At the instantiation of an object parameters of the blue print are fixed. We assume that there is a distinct object class representing the car under design, which we call $\text{ego}$. As minimal requirement on WM we require a global coordinate system in $\mathbb{R} \times \mathbb{R}$ (with an $x$ and a $y$ dimension). Usually objects of WM have at least a defined reference position, $pos \in \mathbb{R} \times \mathbb{R}$.

Although we allow infinitely many objects in the world model, we require that only finitely many are alive at any given time. To this end, we require that the TSC’s signature $\Sigma$ has a unary predicate $\text{alive}$ and that there is an appropriate interpretation of $\text{alive}(o)$ to distinguish whether object $o$ is currently alive or non-alive in WM. As in [1] the process of becoming alive or non-alive is assumed to be governed by rules that reflect plausibilities of the world model.

We do not fix a certain formalism for the world model. We even allow the use of time discrete models. For our application context though, we prefer to think of hybrid automata. In the next paragraph, we present a notion of hybrid automata apt to be used as formal basis for our world model. So let us consider a WM that is given via the parallel composition of (finitely or infinitely many) hybrid automata $H_i$, that are instances of finitely many au-

---

4A higher dimensional coordinate system is possible, but for simplicity of the presentation, we restrict ourselves here to two dimensions.

5There may be exceptions if we do not want to talk about the positioning of certain objects.
tomaton classes \( C_i \), where an automata class is defined by the same dynamics law and list of variables. Intuitively, the automata instances represent objects within the world model.

The type \( \sigma(o) \) of an object variable \( o \) gets interpreted as an automaton class \( C_i \). The valuation \( \mu(o.a_j) \) at time \( t \) of attribute \( o.a_j \) is then the value of the variable \( a_j \) of \( H_i \) at time \( t \). So, for simplicity we do not distinguish between object attributes and variables of \( H_i \).

In order to represent the world as a compilation of hybrid automata, the relation of the automata with respect to each other have to be defined.

For instance, a hybrid automaton modelling a car defines a differential equation that determines the evolution of the car’s position. For the street the car is on, a moving car means a change of occupied positions. Likewise the car may define a differential equation modelling the temperature rise at its wheels at a emergency braking maneuver. The temperature rise at the car wheels influences the road surface temperature.

We model the interaction between objects via output and input variables.

So far, the requirements on a formal model of WM are very general in order to impose minimal restrictions. In the following we sketch a possible world model in more detail to illustrate how a world model could look like.

As a WM-instance for traffic scenarios, we assume that automata instances of one dedicated class correspond to the type of the car under design, which we call ego, so that we can specify requirements on ego. The models of the WM objects capture already known/fixed aspects of the application domain. So for instance, if we want to design the controller for an autonomous driving function, the model of ego reflects aspects of the car physics. Further we assume that objects of WM have sensors to perceive their surroundings (cf. Figure 5). A sensor of a car is modeled via an input variable of the hybrid automaton \( H_{\text{car}} \). Further, \( H_{\text{car}} \) controls its acceleration; the acceleration is an output of \( H_{\text{car}} \) and determines the evolution of the car’s position within the environment. So more generally (not limited to cars only), to model sensors that observe the object within a sensor orientation (e.g. up to 50m in front), we postulate that an object has sensor input variables. A topology automaton similar to [2] observes all objects and the environment and writes as output these sensor variables. It updates these variables so that the front sensor of an object is determined by the position of that object that is currently in front of it. We refer within TSC to objects and the environment, but not to the topology automaton.

![Figure 5: Sketch of a world model as parallel composition of hybrid automata.](image-url)
Hybrid Automata and their Runs In this section, we introduce an automata model that is apt to formalize world models in our application domain. We consider hybrid I/O automata (HIOA) that distinguish between input, output and local variables. An HIOA $H$ is a tuple $(M, \text{Var}^{\text{loc}}, \text{Var}^{\text{in}}, \text{Var}^{\text{out}}, R^{\text{dscr}}, R^{\text{nt}}, \Theta^{\text{in}}, \Theta^{\text{nt}})$ where

- $M$ is a finite set of modes,
- $\text{Var}^{\text{loc}}$, $\text{Var}^{\text{in}}$ and $\text{Var}^{\text{out}}$ are disjoint sets of local, input and output variables over $\mathbb{R}$.
- We denote $\text{Var}^{\text{loc}} \cup \text{Var}^{\text{in}} \cup \text{Var}^{\text{out}}$ as $\text{Var}$ and $\text{Var}^{\text{loc}} \cup \text{Var}^{\text{out}}$ as $\text{Var}^c$, the set of controlled variables. $X \in \mathbb{R}^{|\text{Var}|}$ denotes a state of $H$, where we assume a fixed but arbitrary order of $v \in \text{Var}$. Also we denote as $x_i$ the $i$-th coefficient of $X$.
- $\Theta^{\text{in}}$ is a predicate over $\text{Var}$ and a mode variable $v_{\ell \ell}$ which describes all combinations of initial states and modes.
- $\Theta^{\text{nt}}$ associates with each mode $m \in M$ a local invariant formula $\Theta^{\text{in}}(m)$.
- $R^{\text{dscr}}$ is the discrete transition relation with elements $(m, G, A, m')$ where $m, m' \in M$.
- $G$ is a first-order predicate over $\text{Var}$, and $A$ is a first-order predicate over $\text{Var} \cup \text{Var}^c$.
- $R^{\text{dscr}}$ holds decorated variants of variables in $\text{Var}^c$ and represents the value after the discrete update. $R^{\text{dscr}}$ consists of disjoint sets $R^d_U$, the urgent transitions, and $R^{\text{dscr}}_L$, the lazy transitions.
- $R^{\text{nt}}$ defines the continuous evolutions at each mode $m$ via the function $R^{\text{nt}}(m)$ that maps each $X \in \mathbb{R}^{\text{Var}}$ onto a closed subset of $\mathbb{R}^{\text{Var}^c}$, which is taken as the right-hand side of a differential inclusion.

Let $(\tau_i)$ be a time sequence, i.e., a sequence of monotonically increasing values of Time with $\tau_0 = 0$. A trajectory $(\pi_t)$ with switching times $(\tau_j)$ is a sequence of continuously differentiable functions $\pi_t : [\tau_i, \tau_{i+1}] \to \mathbb{R}_+$. $(\pi_t)'$ denotes the trajectory $(\pi_t)$ shifted by time $t$ so that $(\pi_t)'(t') = (\pi_t)(t' + t)$ and $r'(0) = 0$, and with $j$ the smallest index with $\tau_j > t$. $\forall i > 0 : \tau_i = \tau_{i+1} - t$. We define $\pi(t)$ as the $\pi_{\tau(t)}$, such that $\forall j > i : \tau_j > t$, i.e., $\pi(t)$ is the system state after all (possibly super-dense) switches that occur at time $t$.

A sequence $(\rho_t)$ is a run of $H$ with switching times $(\tau_i)$, where $\rho_t = [M_t, X^c_t, X^l_t]$, $M_t = \begin{bmatrix} X^c_t & \vdots \\ X^l_t & \vdots \end{bmatrix}$. $M_t : [\tau_i, \tau_{i+1}] \to M$, $X^c_t : [\tau_i, \tau_{i+1}] \to \mathbb{R}^{\text{Var}^c}$, and $X^l_t : [\tau_i, \tau_{i+1}] \to \mathbb{R}^{\text{Var}^i}$ are continuously differentiable functions and when it satisfies

- $(\rho_t)$ starts at an initial state, $\rho_0(0) \vdash \Phi^{\text{in}}$,
- mode changes at switching times only, $\forall i \in \mathbb{N} : \forall t \in [\tau_i, \tau_{i+1}) : M_t(t) = M_{\tau_i}$,
- the continuous evolution is governed by $R^{\text{nt}}$, $\forall i \in \mathbb{N} : \forall t \in (\tau_i, \tau_{i+1}) : (dx^c_t/dt(t), x^l_t(t)) \vdash R^{\text{nt}}(M_{\tau_i})$,
- invariants hold, $\forall i \in \mathbb{N} \forall t \in [\tau_i, \tau_{i+1}) : X^c(t) \vdash \Theta^{\text{in}}(M_{\tau_i})$,
- urgent discrete transitions are immediately executed, $\forall i \in \mathbb{N} : \forall t \in [\tau_i, \tau_{i+1}) : (\text{Var}_c(t), \phi, \text{A}, m') \in R^{\text{dscr}} \forall t \in [\tau_i, \tau_{i+1})$. we have that $X^c(t) \not\vdash \phi$, and
- at switching times either new values are assigned according to $R^{\text{dscr}}$ or input changes or the hybrid state is unchanged (stuttering), i.e. $\forall i \in \mathbb{N} : (M_{\tau_i} = \pi_{\tau_i} \wedge X^c_{\tau_i} = X^c_{\tau_i})$ and $\forall (m, \phi, A, m') \in R^{\text{dscr}} : (M_{\tau_i} = m \wedge X^c_{\tau_i} = m' \wedge X^l_{\tau_i} \vdash \phi) \wedge X^c_{\tau_i} = X^c_{\tau_i}).$

So the evolution of $\text{Var}^c$ is determined by $H$ itself, while values of $\text{Var}^i$ are assumed to be determined by the environment such that $\text{Var}^i$ is unconstrained (by $H$). The projection of a run of $H$ onto $\mathbb{R}^{\text{Var}^i}$ is called a trajectory of $H$.

\*HIOAs were introduced by Lynch et al. in [13]. The original definition additionally defines local, input, and output actions. These are omitted here since we do not yet specify communication. However, we plan to integrate LSCs to specify communications.
In case two HIOA $H_i, i \in \{1, 2\}$, share only input variables or read the other’s output variables, we define the composition of the two. The parallel composition of $H_1$ and $H_2$, $H_1 \parallel H_2 = H$, is given by

- $\Sigma = \Sigma_1 \times \Sigma_2,$
- $Var^{out} = Var^{1}_{out} \cup Var^{2}_{out}$ and $Var^{in} = (Var^{1}_{in} \cup Var^{2}_{in}) - Var^{out},$
- $\Phi^{loc} = Var^{1}_{loc} \cup \Phi^{2}_{loc} \cup ((Var^{1}_{in} \cup Var^{2}_{in}) \cap Var^{out}),$
- $R^{nt}(m_1, m_2) = R^{nt}_1(m_1) \land R^{nt}_2(m_2),$
- $R^{\Sigma}$ that consists of transitions
  - (a) $(m_1, m_2, \Phi_1, A_1, (m'_1, m'_2))$ for each $(m_1, \Phi_1, A_1, m'_1) \in R^{\Sigma},$ and
  - (b) transitions of the form (a) with the role of $H_1$ and $H_2$ interchanged,
- $\Phi^{init}$ that is defined analogously to $\Phi^{init},$
- $\Phi^{init} = \Phi^{init}_1 \land \Phi^{init}_2$ and $\Theta((m_1, m_2)) = \Theta_1(m_1) \land \Theta_2(m_2).$

Note that input variables of $H_i$ become local variables, if they are driven by outputs of $H_j$, while output variables stay outputs. We denote the composition of infinitely many hybrid automata $H_i, (\ldots((H_1 \parallel H_2)\parallel H_3)) \ldots$, as $\parallel_{\infty} H.$

**Semantics of $\mathbb{L}$** The $\models$ relation is defined inductively over the structure of the formula. To this end, we consider (i) a structure $M$ of $\Sigma$ that interprets $\Sigma$’s symbols on WM and (ii) an infinite trajectory, $(\pi_i)$, of $WM$, and (iii) a valuation $\mu$ to be given that assigns values to the free variables of a formula $\varphi.$ We write $\{o \mapsto \langle \pi_i \rangle\}_{Var^i}$ to denote that the value of $o$ at time $t$ is determined by the value of $(\pi_i)(t)_{Var^i}.$ Since the valuation of object variables is time dependent, the values assigned for object variables need to be time shifted analogously to the trajectory at the definition of the temporal operator $U.$

- Given $\varphi = \Sigma u a e, then \langle \pi_i \rangle, \mu \models \varphi$
- Given $\varphi = p$ or $\varphi = q(v_1, \ldots, v_n)$ or $\varphi = t_1 \cancel{=} t_2,$ then
  - $(\pi_i), \mu \models \varphi$ if $\pi(0) \models \varphi[\mu].$
  - $(\pi_i), \mu \models \neg \varphi$ if not $(\pi_i), \mu \models \varphi.$
  - $(\pi_i), \mu \models \varphi_1 \land \varphi_2$ if $(\pi_i), \mu \models \varphi_1$ and $(\pi_i), \mu \models \varphi_2.$
  - $(\pi_i), \mu \models \exists t \models (t) \text{ iff for some } val \text{ in } Time \text{ it holds that } (\pi_i), \mu' \models \varphi \text{ with } \mu' := \mu \cup \{t \mapsto val\}.$
  - $(\pi_i), \mu \models \exists o \varphi(o) \text{ iff for some } H_i \text{ of automaton class } C \text{ holds that } (\pi_i), \mu' \models \varphi \text{ with } \mu' := \mu \cup \{o \mapsto (\pi_i)_{Var^i}\} \text{ and } I(\sigma(o)) = C.$
  - $(\pi_i), \mu \models \varphi_1 \cup \varphi_2 \text{ iff for some } \epsilon \in [\mu(t_1), \mu(t_2)], (\pi_i)^\varsigma, \mu' \models \varphi_2 \text{ and } (\pi_i)^\varsigma, \mu'' \models \varphi_1 \text{ for all } u \in (0, \tau).$

The order of precedence is $\{\neg, [\square, [\diamondsuit], [\land], [\lor, \rightarrow, \Rightarrow], \{U\}.$

Let $\varphi$ be a closed formula of $\mathbb{L}$ and $WM$ be a HIOA that is interpreted as parallel composition of (possibly infinitely many) hybrid automata. $WM \models A \varphi$ iff all runs of $WM$ satisfy $\varphi$ and $WM \models E \varphi$ iff some run of $WM$ satisfies $\varphi.$

### 5 Snapshots and their Visualization

In the following, we present our concept for a spatial view. We start with an overview on the concepts used in a spatial view. We then discuss the introduced elements in more detail in Section 5.1 to Section 5.6. Their formal semantics will be given in Section 6 by a translation into a first-order formula. In Section 6.2 we introduce the symbol dictionary, where symbols are declared and their link to the world model (also in terms of predicates of $\mathbb{L}$) is specified.
The spatial view certainly represents an important aspect of traffic maneuvers. Properties like collision freedom, distances within a platoon or successful parking maneuvers require reasoning about the spatial dimension.

To start with, a snapshot collects constraints that hold conjunctively. Hence the empty snapshot denotes True (cf. Section 5.1). To fill a snapshot with life, we place visual symbols representing objects of the world model within a snapshot frame.

**Snapshot Spatial View.** The placement of symbols only describes the relative positioning of the respective objects. It does not give any indication about the absolute distances. With other words, we derive from the symbol’s anchor placement that the respective object(anchor)’s x/y positioning are less, equal or greater in comparison to an other object(anchor’s position). If absolute distances need to be specified, this can be done by distance lines, i.e., lines that describe their distance relation cf. Section 5.5. By a snapshot we describe a traffic situation. Therefore,

1. we specify which objects we require to be present or absent.
   We don’t care whether other objects are present or absent. Both is okay if not explicitly stated. In general, placing a symbol within a snapshot frame, means we require such an object to be present. The nowhere-box allows us to rule out that certain objects are present in that snapshot. An illustrating example is given in Figure 6.

   ![Figure 6: Requiring presence and absence.](image)

   There is a car. All situations match this snapshot, where at least one car is present. There may be other objects as well.

   There is no car within the red box. The snapshot uses the n-box to express the requirement "There is an area of 0.5km x 1km without a car".

2. we annotate that objects are in a certain state (have certain attribute values).
   We may use an appropriately modified object symbol (a car with highlighted indicators). We may also annotate it with an appropriate predicate (cf. Figure 7).

3. we specify the placement of objects.
   By placing the object symbols within a snapshot frame, we specify the relative placement of the represented objects (cf. Figure 8).

   For our spatial view, we determine the x,y- position, \(x, y \in \mathbb{R}\) of symbols within a snapshot with respect to a two dimensional coordinate system. The x positions increase from left to right, and y positions from bottom to top. We also assume a coordinate system in \(\mathbb{R} \times \mathbb{R}\) for the world model.

   Further, any symbol \(s\) has anchors, that are dedicated points of the symbol. Also any object of the world model has at least one anchor \(pos \in \mathbb{R} \times \mathbb{R}\). Additional anchors

---

7Note that since we are considering x and y placement separately, the objects may actually be apart in x direction while they still have the same y-coordinate.
can be defined for an object within the specification of the world model. The anchors of a symbol are linked to the anchors of the objects by the symbol dictionary. Only the position of anchors is semantically relevant: The relative placement of symbols in terms of anchors is transferred to the placement of object anchors.

At the spatial view, a (total) order of object positions is derived from the placement of object symbols within a snapshot unless we use somewhere-boxes (s-boxes), cf. Figure 9, to specify that the object may be somewhere within the box or nowhere-boxes (n-boxes) to specify that the object may be nowhere within the box.

Absolute distances can be specified via distance lines (cf. Figure 10).

a) If we place an object symbol \( s \) within a snapshot next to another symbol \( s' \), then we thereby specify the relative placement of the respective objects (cf. Figure 8); with \( s . pos < s' . pos \) we specify that \( o . pos < o' . pos \), where \( s \) represents object \( o \) and \( s' \) object \( o' \). A symbol \( s \) represents at least a distinguished position, \( s . pos \) of the respective object \( o \), that is \( o . pos \). The symbol may also represent the standard anchors \( x, x', y \) and \( y', \) representing the maximal and minimal \( x \) and \( y \) positions covered by the object. The relative placement of symbol( anchor)s is transferred to hold for the respective objects.

b) If we place an object symbol within a somewhere-box, it means, that the symbol may be anywhere within the box. That way, the spatial order among objects within the snapshot does not need to be totally defined. An illustrating example is given in Figure 9.

c) We may define distances between objects.
We annotate distances between objects via distance lines, that span from anchors (cf. Figure 10).

d) We may declare distinguished positions via a position pins. A position pin labels a dedicated position, so that a relative topology can be defined referring to the pin. Position pins have a global scope within the TSC. An example is given in Figure 11.

4. We may annotate relations between objects. To do so, we connect objects via an arrow and label the arrow with a predicate of our signature \( \Sigma \). The predicate on the respective objects specifies the relation of them (e.g. cf. Figure 12).

5. We may negate a snapshot (cf. Figure 13). Similarly to the n-box we cross-out the snapshot via inscribing dashed diagonals. We preferably use a red frame and red dashed diagonals. The negation of a snapshot simply means the negation of its content.

Example 1. Figure 13 means that there is no car not indicating to its right.

5.1 True Snapshot and False Snapshot

An empty snapshot is visualized as a hatched canvas surrounded by a rectangular thin black frame (cf. Figure 14). The empty snapshot is semantically equivalent to True—the identity element of conjunction. It encodes that we do not constrain the world model’s behavior. To underline that we don’t know anything about the state of the world model—since anything is allowed—we may hatch the snapshot canvas. Please keep in mind, that hatching only highlights the lack of knowledge. Hatching does not change the meaning of the snapshot.
There is a car indicating right. The snapshot expresses the constraint via an appropriate visual object symbol.

Figure 7: Constraining the state of an object.

Figure 8: Relative placement of objects. There are three lanes next to each other. In the top lane there is a car. In the bottom lane there are a car and a bike. A total order of object positions is implied: The x position of the car in the bottom lane is less than the x position of the car in the top lane and the x position of the car in the top lane is less than the x position of the bike. Likewise relations of y positions are implied.

Figure 9: Relative placement of objects with s-box. There are three lanes next to each other. In the top lane there is a car. In the bottom lane there are a car and a bike. The x position of the car in the bottom lane is less than the x position of the bike. The x position of the car in the top lane is at least equal to the x position of the car in the bottom lane and at most equal to the x position of the bike.

Figure 10: A car more than 100m behind a bike. The maximal x position of the car object is more than 100m apart from the minimal x position of the bike.
Figure 11: Position pin within a SC: A position pin declares a distinct position—globally with the SC. The SC describes that first there is a free space of more than 5m×2.3m at the position pin, then eventually (eventually is expressed via the second snapshot.) there is a car somewhere within the formally free space.

Figure 12: The leader-follower relation of a car platoon.

Our visualization of False—every behavior of the world model is ruled out—is analogously visualized by a hatched canvas within a frame and a cross (connecting diagonally opposite corners)—to symbolize the negation of True. We prefer to use a red frame and a red cross. The true snapshot is useful for instance when listing scenarios. Usually arbitrary behavior is allowed before an observed scenario, as, e.g., in Figure 2 and Figure 3 on page 6. The false snapshot allows to rule out certain behavior of the world model. An example is given in Figure 60 on page 47.

5.2 Object Symbol

An object symbol (like 🚗) represents an object of the world model (here, an instance of class Car). Semantically, the object symbol specifies constrains on the world model objects—such as "the object is an instance of class Car", "the object is an instance of class Car with speed 200 km/h" or "the object is an instance of class Pedestrian". Any object that satisfies the constraints associated to the object symbol is a valid match. Given a world model with unboundedly many objects, a single snapshot like in Figure 7 that says "there is a car indicating to its right", represents infinitely many situations, for instance "there is a car indicating to its right and there are three car not indicating" or "there is a car indicating to its right and there is a cow", ....

The situation depicted via a snapshot (visualization) visualizes a conjunction of constraints on the world model. We have already seen visualizations of cars, bikes and also lanes. So the world model defines dynamic objects like cars and also infrastructure objects like lanes, traffic lights or parking lots.

Example 2. We can for instance visualize the infrastructure of having a road with three lanes as in Figure 15 or an intersection as in Figure 16.

---

*We used the True snapshot to express eventually in Section 3.
Figure 13: A simple negated snapshot.

True, i.e., the empty snapshot

False, i.e., a negated empty snapshot

Also the visual representation of False

Figure 14: The empty snapshot encodes True. The negated empty snapshot encodes False.

Figure 15: Three lanes

Figure 16: An intersection

Along with a world model defining objects we allow the logics to refer to types.

5.2.1 Visualization of Objects

An object symbol represents a single object of the world model. The object symbol expresses constraints on the object that narrow down the choice of matching objects within the world model.

5.2.2 Class Instances – Visualization of Constraints of an Object Type

For the following, let the type of an object be its automaton class within the world model WM. We have a more general concept in mind, that allows using types abstractly defined via ontologies, but this is out of the scope of this paper.

To visualize the type of an object, we advocate to use intuitive graphical symbols. The degree of abstractness of the symbol is critical though. Since symbols like ♦, ♣, ♢ and ♠ are intuitively understood as representations of a car, bike or truck, the represented object is usually associated with list of associated attribute values and properties.
Figure 17: Can the car/bike/general vehicle circumvent the obstacle without colliding with the opposing traffic?

Left indicators are lit. Right indicators are lit. No indicator is lit.

Figure 18: Visualization for the state of left and right turn indicators

For example, most people would agree with the following statements "A bike fits into a lane.", "A car is wider than a bike.", "A truck is wider than a car.", "A truck may be too wide to fit into a lane but a truck fits in two lanes.".

To avoid that users wrongly associate properties with (seemingly intuitive) symbols, we advocate the use of sufficiently abstract graphical symbols like \( \bigotimes \), if it is in question whether the properties associated with the symbol match the properties of the objects represented by the symbol.

Example 3. To illustrate how the choice of symbols primes our perception of a situation, let us consider the three snapshots of Figure 17.

Each snapshot specifies a situation where two cars/bikes/general vehicles are involved and an obstacle (black rectangle). At the first snapshot circumvention of the obstacle does not seem an option, while in the middle snapshot the expectation usually is, that the bike facing the obstacle may be able to change into the other lane, if the opposing bike cooperates, as both bikes can be side by side in one lane. Maybe less expectations on the possible future evolution are triggered by the right most snapshot, since the chosen symbol does not trigger an association with a real world object.

5.2.3 Features – Visualization of Constraints on the Object’s State

To visualize the state of an object, we usually modify the object symbol. The visualization of an object with features (like setting turn indicators or brake lights) is usually derived from the object symbol (without features), e.g., a car operating its turn indicators is visualized as a car (symbol), that additionally got indicator lights, cf. Figure 18.

As a rule of thumb, we draw colored indicator lights to denote that the lights are switched on. To denote that the lights are turned off, we draw a red cross on top of the "switched on"-symbol. If we do not explicitly specify the state of a light, then its value is unconstrained, i.e., "don’t care" (cf. Figure 18).

As a general rule, we specify absence by negation of presence and if we do not specify the state of a certain feature, we consider it as not relevant, i.e., "don’t care". Constraints on

\(^9\) people who read or write TSC specifications
an object can also be expressed as a predicate that labels the respective object symbol (cf. Figure 7).

5.2.4 Relative Placement – Positions and Anchors of a Symbol

We distinguish between symbols, which visually represent objects, and visualized relations. Symbols are either object symbols, s- and n-boxes. Visualized relations are predicate arrows and distance lines. The arrangement of symbols within a snapshot is transferred as constraints on the arrangement of the represented objects.

At the spatial view a visual symbol \( s \) represents a world model object \( o \) that covers at least one position in the world model’s space, its anchor \( o.\text{pos} \). A symbol \( s \) and the represented world model object \( o \) correspond at least at this distinct position, \( s.\text{pos} \) represents \( o.\text{pos} \). We require that any symbol and any object declares at least the anchor \( \text{pos} \). Additionally, we often use the \( \overline{x}, x, y, \overline{y} \) to represent the minimal/maximal covered x/y position of the object or symbol. Anchors are declared in the symbol dictionary.

The relative placement of the symbols’ anchors—e.g. constraints like \( s.\text{pos} < s'.\text{pos} \)—is translated into a corresponding relative placement of the world model objects’ anchors—\( o.\text{pos} < o'.\text{pos} \).

If an object symbol \( s \) represents a world model object \( o \) of an abstract type—like vehicles—the symbol may represent just a position. Our formalism allows to not specify the size of an object, so that for instance constraints on the size can be determined later on via an analysis of the modelled scenarios.

For more concrete objects usually certain spatial constraints are assumed and intuitive. So if a symbol represents more concretely a supermini car, then certain spatial constrains may be represented by the car symbol. We expect that usually an upper and lower bound of an object can be determined even for an object of an abstract type vehicle. An object symbol—additionally to representing the distinguished position—may also represent the standard anchors \( \overline{x}, x, \overline{y} \) and \( y \). If a symbol is declared to represent these anchors, then the symbol represents a spatial region. For such symbols the relative placement is derived wrt their anchors.

Example 4. [Figure 19] shows a snapshot with a car on a road. That the car is actually in the road follows from the topological relations of the anchors that is specified by the spatial arrangement of symbols.

![Figure 19: Car in a one lane road.](image)

The lane symbol and the car symbol have standard anchors, as declared in the symbol dictionary (cf. [Figure 20]).

Note, that in [Figure 19] the lane’s \( \overline{x} \) and \( x \) are outside of the snapshot.

We derive the following list of constraints on the topology of the lane and the car object from the snapshot: The car’s \( \overline{y} \) is less than the lane’s \( \overline{y} \), its \( y \) is greater than the lane’s \( y \), its \( \overline{x} \) is less than the lanes \( \overline{x} \) and its \( x \) is greater than the lane’s \( \overline{x} \).
5.3 Somewhere-box and Nowhere-box

Object symbols placed within the s-box represent objects that are somewhere within the area represented by the s-box. With other words, we can create a semantically equivalent snapshot but with a different graphical representation, if we move the box’s content within the box (cf. Figure 21). We change the semantics though, if we change the relative placement
1 among object symbols within the s-box and
2 among objects outside of the s-box and
3 of objects outside of the s-box in the s-box.
For examples see Figure 22

Figure 21: Semantically equivalent but (graphically) different snapshots, as the object symbols in the top lane changed positions. Both snapshots express: There are three lanes next to each other. In the top lane there is a bike and then a car. In the bottom lane there are a car and then a bike. The s-box is beside the car in the bottom lane and behind the bike in the bottom lane. Hence the car and the bike in the top lane are behind (i.e., their x positions are less than that of) the bike in the bottom lane. The car in the bottom lane may be side by side to the car or bike in the top lane.

Note that s-boxes may overlap as e.g. in Figure 23

While the s-box expresses that the contained objects with their relative arrangement are somewhere within the box, the n-box expresses that inside of it, nowhere are the contained objects with their relative arrangement. Likewise, the content of the n-box can be moved within the n-box without changing the semantics.
Figure 22: The somewhere-box and relative placement. The presented snapshots are not equivalent to each other since the relative order of object symbols has been changed (see item 1 for sn₁, item 2 for sn₂, for sn₃ item 3 on p. 19).

Figure 23: Overlapping s-boxes. There may be two bikes next to the car or there may be just one bike where the two boxes overlap.

In the bottom lane there is nowhere a bike and then a car, but note that e.g. a car and then a bike is allowed. In the middle lane there is a car. In the top lane, there is a bike and right in front of it there is no car. This bike can be next to the car in the middle lane or in front of it.

Figure 24: Nowhere-box and somewhere-box
5.4 Position Pins

We use position pins to label distinct positions. The position pin introduces a name for a distinct position that can be used within a SC to define constraints on the spatial relation between the pin and other objects. We have already seen an example in Figure 11 on page 15.

Formally, a position pin does not represent objects of the world model but refers to the coordinates of the world model’s coordinate system. But we may think of position pins as symbols representing a world model object that is not moving. There are two reasons why we introduce position pins as part of the TSC core language: (1) we ensure this way that a position pin has no dynamics assigned to it and (2) there usually is no real world object like a position pin, its an artificial means to describe spatial constraints.

5.5 Distance Lines

We use distances lines to specify x and y distances between objects. Distance lines are labeled with a constraint on the respective distance. Distance lines are a shorthand notation for a special kind of predicate arrows, i.e., instead of using distance lines one can equivalently annotate predicates constraining the distance between the object’s reference points. Distance lines can be used quite intuitively in many situations and they provide a succinct annotation of distance constraints – as illustrated in Figure 10 on page 14 and reproduced here for the readers convenience in Figure 25. Figure 25 specifies the distance between the car and the bike (actually: the distance between $x$ of the car and $x$ of the bike) to be greater than 100m,

![Figure 25: A car more than 100m behind a bike. The car's front is more than 100m apart from the rear of the bike.](image)

Conceptually distance lines are abbreviations to annotate distance constraints. Distance lines come in two flavors $\leftrightarrow$ and $\rightarrow$

1. A distance line $\leftrightarrow$ is either a horizontal or vertical line and refers to the distance between the respective objects’ bounding box, that is defined by the standard anchors $x$, $\bar{x}$, $y$, $\bar{y}$, representing the minimal and maximal x/y positions. The distance line heads are bars. If necessary, these bars are extended to connect the respective object symbols (cf. Figure 26).

Object symbols that are connected via distance lines are required to represent objects that have a spatial extension (for instance, we can image that wind is modelled as an object without a spatial extension). Any object with a spatial extension has x and y extrema (represented via $\bar{x}, \bar{x}, \bar{y}, \bar{y}$), maximum and minimum of x/y values of coordinates it covers. The bounding box is a concept to visualize these extrema. The bounding box is a rectangle whose sides go through the extrema of the (contained) object, i.e., the left side of the bounding box is the vertical line passing through $\bar{x}$, the right side

\[10\] We use simple and clear shapes here to stress the difference between (i) distance line referring to anchors and (ii) distance lines referring to bounding boxes. The same concepts are applied when traffic symbols are used.
is the vertical line through $\bar{x}$, the bottom line is a horizontal line through $y$ and the top line passes through the $\bar{y}$.

Figure 26: | $| $ Distance lines. The distance lines without header annotations imply constraints on the extrema of coordinates. In the right snapshot the bounding boxes are depicted, to visualize the extrema, as the borders of a bounding box run through the extrema.

Connecting symbols $s_1, s_2$ (or different sides of the same symbol) via a distance line labeled with, e.g., $\bowtie d$, means the constraint "$|e_1 - e_2| \bowtie d$", where $\bowtie \in \{\leq, \geq, <, >, =\}$, $d \in \mathbb{R}$ and $e_1, e_2$ are either both x anchors or both y anchors. A horizontal line at the right side of a symbol $\delta$ (i.e., the right side of the bounding box) anchors at $\delta \bar{x}$, at its left side $\delta . x$. A vertical line at the top anchors at $\delta \bar{y}$ and a vertical line at an object’s bottom anchors at $\delta . y$.

2. Additionally, we allow $\leftrightarrow$ distance lines to refer to certain anchors of the respective objects. These anchors represent distinguished coordinates of the respective world model objects. The distance line implies a constraint on the Euclidean distance between the two coordinates.

The use of distance lines with header references is illustrated in Figure 27.

Figure 27: Distance lines with anchor references at their headers. $\text{corner1}$ is an anchor of the star symbol. Likewise $\text{center}$ is an anchor of the box symbol. In the symbol dictionary the symbols’ anchors are mapped to objects’ anchors.

5.6 Predicate Arrows

We can annotate general predicates on objects via predicate arrows. An arrow is drawn between the involved objects to which the predicate refers. An arrow’s line is labelled with the respective predicate. We have already seen examples of such predicate annotations in Figure 7 on p. 14 and Figure 12 on page 15 again reproduced in Figure 28 for the readers’ convenience.

We have already seen the left snapshot in Figure 7 on p. 14 where we also presented a visual representation of the predicate via a car symbol with lit left indicators. Visualization often seems as a very succinct way to represent predicates. But sometimes it is hard to find an intuitive and exact visual symbol. Also, it can be tedious to learn a catalog of symbols,
Figure 28: Predicate annotations in snapshots. $S_{n_1}$ and $S_{n_2}$ show how unary or binary predicates can be annotated as text labels of arrows connecting the involved objects. $S_{n_2}$ and $s_{n_3}$ illustrate a visual representation of the follower-leader predicate (with discussable intuitiveness).

especially if the symbols are rarely used. So, having the option to visualize or express the constraint textual seems an elegant option. For instance, since the visualization and the predicate are semantically equivalent, a TSC application program could easily display the textual predicates, when e.g., the mouse cursor hovers over the respective symbol.

We only allow binary predicates between a symbol occurrence $s$ and an occurrence $s'$ that is in the same frame or an upper frame. So in Figure 29a we can for instance connect the car and the bike in the top lane, since the bike is in an upper frame of the car. That way we can for instance express that there is no car slower than the bike in front of the bike. Likewise we can connect the bike in bottom lane to the car in the middle lane.

For now we only allow this restricted use of predicate arrows. A more general scheme is possible. We plan to evaluate the practical relevance and intuitiveness first in order to decide which kind of extension we want to implement.

A first simple step towards a more general scheme is to allow predicate arrows (from any frame) to all positive symbol occurrences, that is, all those occurrences not directly and not transitively within a nowhere box.

Figure 29b illustrates that connecting to a symbol within a nowhere-box may be problematic. Assume we connect the two white cars, as illustrated by the orange arrow, stating the upper car has the same brand as the lower car. So at the top lane there is somewhere a bike and in front of it there is nowhere a car, which has the same brand as—and here is the problematic bit—the car that is nowhere in front of the bike at the lower lane. So the predicate refers to an object which is required not to exist. Hence, if we want to allow even more general predicate arrows we have to take care of existence of objects and hence the negation levels.

Figure 29: Predicate arrows between symbols of different frames.
5.7 Conclusion

In this section, we introduced the basic elements within a snapshot. The focus is on visualizing the spatial view of traffic scenarios. An optimization criteria for the presented visualization is that the specifications are printable. Nevertheless, an editor to specify and display the specification seems the best way to deal with the high complexity of the specification. Then different aspects can be displayed in combination or switched off to reduce the complexity for the current analysis.

6 From Snapshots to First-Order Logic

By now, we have introduced elements of visual syntax of the spatial view of a single snapshot. We can place object symbols, n-boxes, s-boxes and position pins within a snapshot. We can annotate distance lines and predicate arrows. In the following, we will explain, how a snapshot of a spatial view can be translated into a formula of our first-order logic $\mathcal{L}$ that in turn is interpreted on the world model WM.

6.1 Overview

In the following, we give a short overview of the translation process. We introduce the basic translation scheme and give an illustrative example. In the following subsections, we will detailedly explain how a snapshot of the spatial view is translated into a first-order formula. At the translation, we distinguish between symbols $s$ and symbol occurrences $s$. A symbol like $\text{car}$ can occur at the same snapshot several times (e.g. three times in the second and third snapshot of Figure 28).

At the translation process, object symbol occurrences are mapped to object variables and predicates are derived from the visual snapshot that constrain the possible values of these variables. A variable $o_{\text{car}}$ corresponding to $\text{car}$ can only take on values of class Car and additionally a predicate like $\text{left}(o_{\text{car}})$ is derived. The relative placement of symbols is transferred to placement constraints referring to the object variables. Also distance lines and predicate arrows are translated to predicates referring to these variables.

Somewhere- (and nowhere-)boxes roughly express "within my bounds somewhere (resp. nowhere) there is my content". To deal with somewhere- and nowhere-boxes in our translation scheme, we introduce the notion of frame. Boxes can be nested; if for instance we place at snapshot a s-box and within this s-box a bike symbol and in front of it a n-box symbol and within the n-box a car symbol, as at the top of Figure 30 (1), we specify "somewhere is a bike and in front of it nowhere is a car"). So the meaning of the symbol occurrences also depends on their container, their frame. A snapshot $sn$ spans a frame, $f_{sn}$. Also each box symbol occurrence $b$ spans a frame, $f_{b}$. Further, we define that a symbol occurrence $s$ belongs to frame $f$, if $s$ is within the symbol occurrence $b$ spanning $f$, $f_{b} = f$, and if there is another symbol $b'$ spanning $f_{b'}$ that contains $s$ then it also contains $b$, i.e., $s$ is directly placed in $f$. At our above example, the s-box belongs to the snapshot’s frame while the bike and the n-box belong to the s-box’s frame. The car belongs to the n-box’s frame.

For the translation, the set $S_{f}$ denotes the set of all symbol occurrences (i.e., object and box symbols) that belong to frame $f$. We assume that each symbol occurrence $s \in S_{f}$ has an identity $s.id$ and a position in $\mathbb{R} \times \mathbb{R}$ within the snapshot for each of its anchors. We
write $s.a.x$ and $s.a.y$ for the x and y coordinate of the position of anchor $a$. We write $s.a$ to denote the occurring symbol. Further, we consider the set $A_f$ of occurrences of predicate arrows (cf. item 4) that connect a symbol occurrence $s \in S_f$ to a symbol occurrence $s'$ in a frame (transitively) containing $f$. Also, $a \in A_f$ has an identity and we denote its source symbol occurrence as $a.s$ and its target symbol occurrence as $a.s'$. $L_f$ is the set of all distance lines that connect symbol occurrences in $S_f$ with symbol occurrences at a frame transitively containing $f$. Additionally, we denote by $S_f^{WB}$ the set of n-box occurrences at frame $f$, $S_f^{SB}$ denotes the set of s-box occurrences at $f$, $S_f^B$ denotes the set $S_f^{SB} \cup S_f^{SB}$, $S_f^B$ is the set $S_f \setminus S_f^B$ and $S_f^{PP}$ is the set of position pin occurrences at frame $f$.

To translate a snapshot (cf. Algorithm 2), we recursively translate the snapshot frame. To translate a frame $f$ we

(i) derive constraints that capture the meaning of the symbols $s$ with occurrence $s$ at $f$, $s \in S_f$. If $s.a$ is a box, we use as constraint the formula of the content of $s$ here.

(ii) We reflect the relative placement of symbols of $f$,

(iii) encode that the symbols of $f$ are within $f$, and

(iv) reflect predicate arrows $a \in A_f$ between (a) symbols of $f$ and (b) symbols at $f$ or upper frames.

For an example let us consider the first snapshot at Figure 30. We highlight the currently translated elements by a white background and grey-off the remainder.

### 6.2 Symbol Dictionary

The $s$-dictionary defines the interpretation of visual symbols in terms of the signature $\Sigma$ and formulas of $L$. These definitions are used at the translation of a snapshot to a formula of $L$.

(i) For an object symbol $s$ the $s$-dictionary specifies $\text{type}_{s\text{dict}}(s)$, a type $T \in \Gamma$. Occurrences $s$ of $s$ get translated to object variables $o_s$ of the type declared at the $s$-dictionary, $\text{type}_{s\text{dict}}(s)$.

(ii) For all modifications $s'$ of an object symbol $s$ (cf. item 2, p. 12), it specifies a unary predicate $O_{s'}(o_{s.id})$ of $L$, that encodes the constraints visualized via the modification of $s$.

So given we have a car symbol—as example of an object symbol—we can declare at the symbol dictionary that a "car is indicating" is represented by a car symbol modification, e.g., the car symbol with indication lights in yellow. The fact that the car is indicating is translated as a constraint on the car object, like, e.g., $\text{indicating}(o_{\text{car}})$.

(iii) Further, the $s$-dictionary specifies spatial characteristics.

For the spatial view we require that each symbol $s$ has at least an anchor that is a dedicated position within the symbol. A symbol’s anchors are declared at the $s$-dictionary, as, e.g., in Figure 31. Also at the $s$-dictionary the symbol’s anchors are linked to the anchors of objects that are represented by the symbol. By default, $s.w$ is translated to $o_{s.id}.w$ where $w \in \{\text{pos}, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}\}$ and $s$ is the translated occurrence of $s$. The anchors $x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}$ represent a bounding box by being interpreted as minimal and maximal positions at the x/y dimension.
We present here a prose sketch of the translation process. The visual elements currently being translated are highlighted.

The top-level frame translates to “There are three lanes next to each other. At the top lane is a s-box swb and within it, \( \varphi_{swb} \) holds. At the middle lane is a car. At the bottom lane is a n-box nwb and within it \( \neg \varphi_{nwb} \) holds. On the x-axis, first comes the n-box’s back, then the s-box’s back, then the car’s back, then the car’s front, then the s-box’s front, then the n-box’s front.”

The content of the s-box swb translates to \( \varphi_{swb} = \text{“There is a bike and at its front (anchor) is an n-box, within which } \neg \varphi_{nwb} \text{ holds.”} \)

The content of the n-box nwb at the top lane translates to \( \varphi'_{nwb} = \text{“There is a car faster than the car at the middle lane.”} \)

The content of the bottom n-box translates to \( \varphi_{nwb} := \text{“There is a bike and in front of it a car.”} \)

Put together:

There are three lanes next to each other. At the top lane somewhere within area A1 is a bike then directly in front of it nowhere is a car faster than the car at the middle lane. At the middle lane is a car. At the bottom lane nowhere is first a bike and then a car. We also derive the following order of anchors on the x-axis: Area A1 is included by the x-positions of area A3, the bottom n-box. The x-position of the car is included in the x-positions of A1.

Figure 30: Translating a Snapshot with Nesting.
6.3 Symbol Placement within a Snapshot

Given a single snapshot of a spatial view as introduced at Section 5. Recall that a snapshot is composed of nested frames. Let $S_f$ be the set of all symbols—object symbols, nowhere-boxes, somewhere-boxes, pins—of a snapshot’s frame $f$.

6.3.1 Presence of Symbols

In the following we define what the presence of symbols means.

**Object Symbol.** The occurrence of an object symbol $s \in S_f^O$ within a snapshot $sn$ translates to a subformula that states that a corresponding object exists.

$$E_s = \exists o_{s, id} \in \text{type}_{idict(s, \ldots)} : \text{alive}(o_{s, id})$$

where $\text{alive}(o_{s, id})$ expresses that the quantified object is actually alive (cf. Section 4, page 8). $E_s$ introduces for each object symbol occurrence $s$ a corresponding object variable $o_{s, id}$.

The variable $o_{s, id}$ is further constrained by the snapshot: How these constraints are encoded into the formula will be explained in the following.

**Box Symbol.** The occurrence of a box symbol $s \in S_f^B$ within a snapshot $sn$ translates to a subformula that binds variables $x_{s, id}, y_{s, id}, \overline{x}_{s, id}, \overline{y}_{s, id}$, that represent the boxes right, left, bottom and top side.

$$E_s = \exists x_{s, id}, \overline{x}_{s, id}, y_{s, id}, \overline{y}_{s, id} \in \mathbb{R} : x_{s, id} \leq \overline{x}_{s, id} \land y_{s, id} \leq \overline{y}_{s, id}$$

**Position Pin Symbol.** For the convenience of the reader, we reproduced the position pin example of Figure 11, p. 15 in Figure 32.

The occurrence of a position pin symbol $s \in S_f^{PP}$ within a snapshot $sn$ means a quantification of two distinct position variables, $x_{s, id}, y_{s, id}$. The scope of this quantification covers the complete SC. So a position pin variable introduces quantified variables $x_{s, id}, y_{s, id}$ that represent x and y coordinate labeled via the position pin.

$$E_s = \exists x_{s, id} \exists y_{s, id}$$
Figure 32: Position pin within a SC: A position pin declares a distinct position—globally with the SC. The SC describes that first there is a free space of more than 5m×2.3m at the position pin, then eventually (eventually is expressed via the second snapshot.) there is a car somewhere within the formally free space.

Within the overall formula that represents the SC, this snipped is prefix, so that the scope of quantification covers the formula derived for the SC. Therefore position pins are not quantified at snapshot (cf. Algorithm 2 and Section 9.5).

Example 5. Roughly, the SC of Figure 33 expresses that there is a position p so that first there is a car approaching it and then passing the position p on the bottom lane (snapshots one to three). Then there is no car at the bottom lane below p. p (snapshot 4). Finally the dark grey car is again approaching p.

The derived formula is prefixed by $E_s$ to express “there is a $x$ position of p and a $y$ position of p” about which we talk in the following snapshots—that is, the following constraints refer to the $x_{s,id}, y_{s,d}$ variables representing the distinct position fixed via the position pin.

6.3.2 Spatial Relations between Symbols

The relative placement between two symbol occurrences $s, s' \in S_f$ of the same frame implies constraints on the topology of the represented objects. By looking at a snapshot like the one given at Figure 35 we can say “the bike is in front of the car”. To capture this semantic dimension of the visualization as constraint in a formula, we order the position of objects just like the positions of symbol occurrences are ordered—we reflect only anchor positions.

The spatial arrangement can be expressed via “car.$x$ < bike.$x$”.

As stated in Section 4 we assume a two dimensional coordinate system on the world model.
The general approach is also applicable to higher dimensions. If we want to specify for instance scenarios in air space, we could come up with a TSC like in Figure 35, where we specify that a plane should be surrounded by a free space in three dimensions. We assume that the world model then has a three-dimensional coordinate system and that NB1 is a three-dimensional nowhere box.

Figure 35: A 3-dimensional specification. The TSC specifies that if there is a plane, it is surrounded by a free three-dimensional space.

The TSC of Figure 35 gives just an impression on how TSC could be used. For a more comprehensible introduction of the main concept we limit the scope of this paper to world models with two dimensions only.

At the translation of the relative placement we distinguish two kinds of anchors

(i) two dimensional anchors. They are also called point anchors as they represent points in the global coordinate system $\mathbb{R} \times \mathbb{R}$ ("positions").

(ii) one dimensional anchors. They are also called line anchors. Standard line anchors are $\overline{x}$, $\underline{x}$, $\overline{y}$, $\underline{y}$. Together they represent the bounding box of an object.

Then we encode the relative placement by building predicates that encode a comparison of anchors, at all dimensions given.

At the symbol dictionary it is defined which point of the symbol is an anchor—or where the bounding box $\overline{y}, \overline{y}, \underline{x}, \underline{x}$ of the symbol is—and it defines a corresponding point—or bounding box—for the represented objects. For example, the center of the car symbol corresponds to the center of the represented car object. This is illustrated in Figure 36.

As outlined at Section 5.5, we assume an order in both dimensions (the ‘greater’ order on $\mathbb{R}$). The line anchors $\overline{x}, \underline{x}, \overline{y}, \underline{y}$ constitute a bounding box, i.e., they pass through the object points having minimal and maximal $x$ and $y$ position, respectively. For example, for our car symbol as in Figure 37 the symbol’s $\overline{x}$ represents the maximum $x$ coordinate of the car (a point at its bumper). Figure 37 illustrates the bounding box for our car symbol.

In the following, we treat s- and n-boxes basically like object symbols. Since these boxes are not really objects, we make the distinction between them more prominent by not using $\overline{x}, \underline{x}, \overline{y}, \underline{y}$ but introducing new names to represent the respective box. So for a box $b$ we talk about $\text{max}_{x,b}$ and not about $\overline{x}$, we talk about $\text{min}_{x,b}$ instead of $\underline{x}$, we talk about $\text{max}_{y,b}$ instead of $\overline{y}$ and $\text{min}_{y,b}$ anchor instead of $\overline{y}$ and $\underline{y}$.

We derive constraints, which describe the relative placement, by comparing the positions of anchors of all symbol occurrences, as illustrated in Ex. 6.

**Example 6.** Consider the snapshot in Fig. 38. Let’s say the star symbol’s anchor corner1 has position $(1.1, 1)$ within the snapshot, while the squares anchor center has position
The symbol dictionary declares a position within the symbol to represent an anchor. This anchor is mapped to the variable \( \text{pos} \) of the hybrid automaton modelling the car object within the world model. In the example the variable \( \text{pos} \) models the center of a car as sketched in the photo.

![Car symbol within its bounding box. The points having the minimum and maximum x and y are within the red ellipses.](image)

Figure 36: Mapping of symbol anchor to an anchor of a world model objects.

Figure 37: Car symbol within its bounding box. The points having the minimum and maximum x and y are within the red ellipses.
We hence derive as a predicate on world model objects star and square that 'star.corner1.x ≤ square.center.x' and 'star.corner1.y ≤ square.center.y'. For convenience we explicitly represent the two anchors in Fig. 38. Anchors are declared in the symbol dictionary and are implicitly present at the symbols. **\( \Box \)**

**Figure 38: Distance lines with anchor references at their headers.**

*corner1* is an anchor of the star symbol. Likewise *center* is an anchor of the box symbol. At the symbol dictionary the symbols’ anchors are mapped to objects’ anchors.

Quite naturally, we only compare the anchor coordinates of the same dimension, as illustrated in Figure 39. Note that in Figure 39 not all constraints are annotated but only some exemplary. Missing is for instance circle.center.y < star.center.x.

We denote one coordinate of a point anchor \( \alpha \) as \( \alpha.x \). Likewise \( \alpha.y \) denotes the other coordinate of \( \alpha \). Note, that the line anchors \( \bar{y}, \bar{y} \) have only \( y \) coordinates and \( \bar{x}, \bar{x} \) have only \( x \) coordinates. For brevity we identify \( \bar{x}.x \) with \( \bar{x} \) and do analogously so for \( \bar{x}.y, \bar{y} \).

**Algorithm 1** builds a predicate that encodes constraints on the relative placements of objects \( o_{s.id}, o_{s'.id} \) according to the placement of two symbol occurrences \( s, s' \). Therefore, it iterates over all possible constraints between the two symbol occurrences. In case a constraint between the symbol occurrences holds, a conjunct encodes a corresponding constraint on the object variables. Therefore the anchor coordinates of the symbol are mapped according to the symbol dictionary onto object attributes.

Note, that the translation scheme **Algorithm 1** certainly does not generate the minimal set of constraints, as this is currently not our priority. Further note that we do not evaluate \( \bowtie \) being \( = \). Equality results from the conjunction of \( \leq \) and \( \geq \) constraints.

**Figure 39: Some constraints that reflect the relative positioning.** We assume that circle.center, star.anchor, star.corner, square.center are anchors that are declared at the symbol dictionary.
Algorithm 1: topologyBetween builds a predicate reflecting relative placement of symbol occurrences \(s,s'\) within snapshot \(sn\)

1 Function topologyBetween
2 input : \(s, s'\) symbol occurrences within snapshot \(sn\)
3 output: \(T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}})\), predicate encoding relative placement
4 \(T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}}) := \text{True};\)
5 foreach \(\bowtie \in \{<, \leq, \geq, >\} \) do
6 foreach \((s..x, s'..x) \in A_x(s.s) \times A_x(s'.s)\) do
7 if \(s..x \bowtie s'..x\) then
8 \(T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}}) := T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}}) \land \\
9 o_{s,\text{id}}.sdic(s..x) \bowtie o_{s',\text{id}}.sdic(s'..x);\)
10 foreach \((s..y, s'..y) \in A_y(s.s) \times A_y(s'.s)\) do
11 if \(s..y \bowtie s..y\) then
12 \(T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}}) := T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}}) \land \\
13 o_{s,\text{id}}.sdic(s..y) \bowtie o_{s',\text{id}}.sdic(s'..y);\)
14 return \(T_{s,s'}(o_{s,\text{id}}, o_{s',\text{id}});\)

6.4 Presence of Predicate Arrows and Distance Lines

Predicate Arrows. Predicate arrows are either unary or binary, that is they relate either to only one symbol/object or they relate two symbols/objects.

- Binary predicate arrows:
  - Binary predicate arrows connect two symbol occurrences \(s, s'\), where w.l.o.g. \(s\) is of a frame \(f\) and \(s'\) of a frame transitively contained in \(f\).
  - The line is annotated with a binary predicate \text{label-predicate}.
  - The predicate arrow is then translated to \(P_l(o_{s,\text{id}}, o_{s',\text{id}}) = \text{label-predicate}(o_{s,\text{id}}, o_{s',\text{id}}).\)

- Unary predicate arrows:
  - If a unary predicate arrow \(l\) connects a label \text{label-predicate} to symbol \(s\), then the predicate arrow is translated to \(P_l(o_{s,\text{id}}) = \text{label-predicate}(o_{s,\text{id}}).\)

Distance Lines. Distance lines are examples how specialized symbols might be used in TSCs. They are a short cut annotation for predicate arrows.

The translation of distance lines distinguishes two flavors (cf. Section 5.5). Both require that a predicate labels the distance line. This predicate expresses a constraint on the distance (cf. Figure 40).

1. A distance line \(\rightarrow\) is either a horizontal or vertical line orthogonally headed by bars and refers to the distance between the sides of the respective objects’ bounding box defined by the anchors \(x, \bar{x}, y, \bar{y}\) (or of \(n\)-or \(s\)-boxes). Object symbols that are connected via distance lines are required to define these anchors. Since the symbols may be at...
the same time horizontally and vertically be apart, we extend the bars appropriately
to connect the two related object.
Consider the following cases: A horizontal distance line is drawn between two x line
anchors or a vertical distance line is drawn between two y line anchors. Then the
distance line is translated to
\[ D_{s,s'}(o,o') = \text{label-predicate}(|o.a - o'.a'|) \]
where 
\( \text{label-predicate} \) is a predicate that labels the distance line and constrains the
distance \( d = |o.a - o'.a'| \). As a short hand, we allow to omit the expression \( |o.a - o'.a'| \)
within \( \text{label-predicate} \).

2. Additionally, distance lines \( \rightarrow \) may refer to point anchors of the connected objects
or to position pins. The distance line is translated to a constraint on the Euclidean
distance between the two points.
Consider two symbol occurrences \( s \) and \( s' \). Consider the case that there is a distance
line between \( s \) and \( s' \) with heads \( h \) and \( h' \), respectively. Say (w.o.l.o.g) the distance line’s head \( h \) is not annotated with an anchor name, then the distance line head refers
to the standard anchor \( a = s.s.pos \). Say (w.o.l.o.g.) the distance line’s head \( h' \) is
annotated with anchor name \( a' \), then \( h' \) refers to that name as \( a' \). In both cases the
symbol dictionary has to have an according symbol declaration and mapping defined.
The distance line is labeled with predicate \( \text{label-predicate} \), then the distance line is translated to
\[ D_{s,s'}(o,o') = \text{label-predicate}(\sqrt{(o.a.x - o'.a'.x)^2 + (o.a.y - o'.a'.y)^2}) \]

[Figure 40] illustrates the translation of \( \rightarrow \) distance lines and highlights the assumed order on
coordinates at a dimension. \( star, square \) and \( rhombus \) are object variables that correspond
to the respective symbols.

---

**Figure 40:** Distance lines without anchor references at their heads.

\[ |star.x - square.x| < 2\text{cm} \land |square.x - square.x| < 1.4\text{cm} \land |square.x - rhombus.x| < \]
\[ 2\text{cm} \]

---

### Figure 41:

[Figure 41] illustrates the translation of \( \rightarrow \) distance lines.

---

**Figure 41:** Distance lines with anchor references at their heads.

\[ \sqrt{(star.corner1.x - square.center.x)^2 + (star.corner1.y - square.center.y)^2} \geq 2\text{cm} \]
6.5 Object Symbol

An object symbol $s \, \sigma$ (e.g., $\bigotimes$) represents a predicate $\mathbb{O}_{s \, \sigma}(o)$ that encodes the constraints on the world model object that are visualized by the symbol $s \, \sigma$ at the symbol occurrence $s$. The mapping from symbol $\sigma$ to predicate $\mathbb{O}_{s \, \sigma}(o)$ is defined at a symbol dictionary (cf. Fig. 31 at p. 27).

Given an object symbol $\sigma$ and entries at the s-dictionary where a symbol feature $f$ is defined to be translated to constraint $F_f(o)$. Let $\bigwedge_{f \, \sigma} F_f(o)$ denote the conjunction of all constraints $F_f$ corresponding to a feature $f$ present at symbol occurrence $s$. A symbol occurrence $s$ induces the predicate

$$\mathbb{O}_{s \, \sigma}(o) = \bigwedge_{f \, \sigma} F_f(o),$$

that describes constraints on the object expressed by the modifications of the visual symbol.

The visual symbol also determines the type of the object variable. To this end, the s-dictionary specifies that $\sigma$ represents objects of type $\text{type}_{\text{sdic}}(\sigma)$. This declaration is reflected as range of the quantifier at Section 6.3.1 on page 27.

Figure 42 illustrates how a single visual object symbol gets translated. Here, we assume that our world model defines a class $\text{Car}$ and we assume that the symbol dictionary declares that $\text{type}_{\text{sdic}}(\bigotimes)$ is $\text{Car}$ and it defines that the symbol modification with right light indicators is translated to $\text{indicateRight}(\cdot)$.

Figure 42: Translation of an object symbol with features. $\exists o \in \text{Car} : \text{alive}(o) \land \text{indicateRight}(o)$

6.6 Somewhere-box and Nowhere-box

We already sketched the translation scheme from the visualization to a first order formula at the start of this section on page 25. This scheme progresses recursively following the nesting of somewhere- and nowhere-boxes.

The somewhere-box expresses that the contained objects (with their relative placement) are somewhere within its boundaries. The nowhere-box denotes that its content is nowhere within the nowhere-box area.

As the objects are somewhere within the somewhere-box’s boundaries (nowhere within the nowhere-box’s boundaries), the spatial arrangement of symbols within the box does not imply spatial constraints referring to objects outside of the box. This is illustrated at Figure 43 and has already been discussed at item 3 at page 12.

We already introduced the notion of frame at page 24. Each snapshot and each box spans a frame. To reflect the relative placement of symbols at snapshots with s- and n-boxes, we derive constraints on the placement only

P1 for objects (including boxes) at the same frame
Figure 43: Relative placement of objects and the somewhere-box.
The car at the top lane can be anywhere within the box. It can, e.g., be next to the car at the bottom
lane or it can be next to the bike.

P2 for objects within a box(-frame) and the box.
This expresses that the box’s content is contained in the box.

By transitivity, spatial relations between objects inside and outside of a box are implied:
There are spatial constraints between the box and objects outside of a box’s and there are
constraints, which in turn relate the box to its contained objects.

So, to deal with boxes within the translation scheme, we derive T-predicates not only for
symbol occurrences \( s, s' \) of the same frame (P1) as we did at Section 6.3.2, we now additionally derive T-
predicates for symbol occurrences \( s, s' \) where \( s \) belongs to frame \( f \) and \( s' \) spans \( f \). We denote the symbol occurrence \( s' \) spans frame \( f \) as \( s' \in S_{\text{spans}(f)} \). The set
\( S_{\text{spans}(f)} \) contains either the box symbol occurrence that spans \( f \) or is the empty set if \( f \) is
the top most frame of the snapshot.

Figure 44 illustrates the constraints at the different nesting levels of frames.
(A) The gray car is at the same frame as the somewhere-box. So for (A) placement con-
straints between the s-box and the gray car are derived.
(B) The bike and the white car are at the same frame. For (B) constraints between the bike
and the white car are derived.
(C) Constraints between the s-box and its content, i.e., the bike and the white car, are
derived.

6.7 Translation Scheme

We already presented the different steps of the translation of a snapshot to a formula. We also
sketched the embedding of the different steps into an overall translation scheme. Algorithm 2
presents the overall translation scheme for a single snapshot using the previously presented
translation steps. We explain the algorithm of Algorithm 2 in the following. Readers that
are just interested to grasp the gist of TSCs can skip this section.

A snapshot describes that there are objects. Basically we encode "there is an object" via an
existential quantification of a corresponding object variable.

But since we also allow to track objects across snapshots, we distinguish between symbols
with SC-global scope and symbols that are local to a snapshot (that is we distinguish be-
 tween "a car" and "the car"). In order to specify that we track an object ("the car") along
a snapshot sequence, we introduce in the next section the bulletin board, which is a visual
mean to assign the same identity to all occurrences of a symbol across all snapshots of an
SC. The object variables that correspond to a bulletin board symbol do not get locally quanti-
fied as in Section 6.3.1. Therefore, the translation procedure of Algorithm 2 takes a set \( \mathcal{S}_B \)
containing symbols with a unique identifier as its input—we assume that these are declared
In the following the constraints capturing the relative placement of object symbols in this snapshot are derived.

Object variable *bike* is bound to $\mathcal{B}$, *g-car* to $\mathcal{G}$ and *w-car* to $\mathcal{W}$.

The relative placement of $\mathcal{G}$ and the s-box (sb) is captured via constraints on their respective anchors. We present here only the minimal set of constraints (using $x < X, y < Y$).

$$sb.x < g-car.x \wedge sb.x > g-car.x \wedge g-car.y < sb.y$$

The relative placement of s-box wrt its content is also expressed via constraints. Again only the minimal set of anchors is given (using $bike.x < w-car.x$).

$$sb.x > bike.x \wedge sb.x > w-car.x \wedge sb.y > w-car.y \wedge sb.y < w-car.y$$

The relative placement of objects within the s-box is expressed via constraints on the object anchors.

$$bike.x < w-car.x \wedge w-car.y < bike.y \wedge w-car.y < bike.y$$

Figure 44: Spatial constraints and somewhere-boxes.

The spatial arrangement is derived between objects within the same frame. By transitivity of the order relation, constraints between the car at the bottom lane and the objects within the somewhere-box follow.

The black dotted and dashed lines of this figure are not part of the SC syntax, they only highlight in this figure the generated constraints. Dotted lines visualize constraints on the $x, x$ and dashed lines visualize constraints on $y, y$. 
circumcurrence by a constraint regarding the relative placement). Within the snapshot, instead—just like object variables for symbols of the b-board. But all variable of symbols not quantify these position pins variables inline 4 or 5, since they will be globally quantified because these variables will be globally quantified (cf. Section 9).

Where the quantifiers’ scopes end. We do not quantify an object variable box’s sides. Predicates on the variables introduced at lines 4 and 5 are added at lines 7 to line 13, where the quantifiers’ scopes end. We do not quantify an object variable o.s.id, if s ∈ δ, because these variables will be globally quantified (cf. Section 9).

Note that a position pin symbol s ∈ Sn introduces two variable p.x.s.id and p.y.s.id. We do not quantify these position pin variables in line 4 or 5 since they will be globally quantified instead—just like object variables for symbols of the b-board. But all variable of symbols within the snapshot, Sf, are restricted by the constraints in the following lines (for instance by a constraint regarding the relative placement).

At line 4 of Algorithm 2, a quantifier opens a (snapshot local) scope for each symbol occurrence s ∈ Sf. At line 5, for each box symbol occurrence the variables are introduced that represent the symbol occurrences at a frame transitively containing f.

At line 7, we require an object o.s.id to denote the conjunction p(s) for a set S = {s1, ..., sn} and likewise we use ∃s∈S p(s) to denote the existential quantification ∃p(s). In case S = ∅, ∨s∈S p(s) is True while ∃s∈S p(s) is void. We use Sf, Af, Lf, SF, SF, SF, SF and SF as before and summarized in Table 1.

We build the formula φ.sn for snapshot sn as conjunction of the predicates introduced before. We use \( \bigwedge_{s \in S} p(s) \) to denote the conjunction p(s1) ∧ ... ∧ p(sn) for a set S = {s1, ..., sn} and likewise we use \( \bigvee_{s \in S} p(s) \) to denote the existential quantification \( \exists p(s) \). In case S = ∅, \( \bigwedge_{s \in S} p(s) \) is True while \( \bigvee_{s \in S} p(s) \) is void. We use Sf, Af, Lf, SF, SF, SF, SF and SF as before and summarized in Table 1.

At line 4 of Algorithm 2, a quantifier opens a (snapshot local) scope for each symbol occurrence s ∈ Sf. At line 5, for each box symbol occurrence the variables are introduced that represent the box’s sides. Predicates on the variables introduced at lines 4 and 5 are added at lines 7 to line 13, where the quantifiers’ scopes end. We do not quantify an object variable o.s.id, if s ∈ δ, because these variables will be globally quantified (cf. Section 9).

At line 7, we require an object o.s.id is alive (cf. Section 4), when s is placed in a frame. The predicate \( \delta_{s,\cdot} \) at line 8 encodes constraints expressing the features visualized by s.δ. These constraints are defined at the s-dictionary.

At line 9, we capture the placement of symbol occurrences s of frame f, Sf, relative to (i) symbols in the same frame s’ ∈ Sf and (ii) the symbol that spans the frame, \( S_{\text{span}(f)} \). \( S_{\text{span}(f)} \) contains either the box symbol that spans f or is empty if f is the top most frame. \( T_{s,s'}(o.s.id, o.s.id) \) translates the relative placement of symbols to constraints on the placement of objects.

At line 10, the predicates at arrows between two symbols s and s’ get translated to predicates on o.s.id and o.s.id. Likewise at line 11 the distance lines between two symbols s and s’ get translated to predicates on constraining the distance of o.s.id and o.s.id.

\( \delta_{s} \) set of symbols with a global scope
\( S_{f} \) set of all symbol occurrences (i.e., object and box symbols) that belong to frame f
\( A_{f} \) occurrences of predicate arrows that connect a symbol occurrence \( s \in S_{f} \) to a symbol occurrence \( s' \) at a frame (transitively) containing f
\( L_{f} \) set of all distance lines that connect symbol occurrences in \( S_{f} \) with symbol occurrences at a frame transitively containing f.
\( s_{\text{SB}} \) set of s-box occurrences at frame f
\( s_{\text{SF}} \) set of n-box occurrences at frame f
\( s_{B} \) equals \( S_{\text{SB}} \cup S_{\text{SF}} \), i.e. set of box occurrences at frame f
\( s_{P} \) set of position pin occurrences at frame f
\( s_{f} \) \( S_{f} \setminus s_{\text{SB}} \)

### Table 1: Recapitulation of set names used in the translation

<table>
<thead>
<tr>
<th>Set Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \delta_{s} )</td>
<td>set of symbols with a global scope</td>
</tr>
<tr>
<td>( S_{f} )</td>
<td>set of all symbol occurrences (i.e., object and box symbols) that belong to frame f</td>
</tr>
<tr>
<td>( A_{f} )</td>
<td>occurrences of predicate arrows that connect a symbol occurrence ( s \in S_{f} ) to a symbol occurrence ( s' ) at a frame (transitively) containing f</td>
</tr>
<tr>
<td>( L_{f} )</td>
<td>set of all distance lines that connect symbol occurrences in ( S_{f} ) with symbol occurrences at a frame transitively containing f.</td>
</tr>
<tr>
<td>( s_{\text{SB}} )</td>
<td>set of s-box occurrences at frame f</td>
</tr>
<tr>
<td>( s_{\text{SF}} )</td>
<td>set of n-box occurrences at frame f</td>
</tr>
<tr>
<td>( s_{B} )</td>
<td>equals ( S_{\text{SB}} \cup S_{\text{SF}} ), i.e. set of box occurrences at frame f</td>
</tr>
<tr>
<td>( s_{P} )</td>
<td>set of position pin occurrences at frame f</td>
</tr>
<tr>
<td>( s_{f} )</td>
<td>( S_{f} \setminus s_{\text{SB}} )</td>
</tr>
</tbody>
</table>

\(^{11}\)in abuse of notation \( S_{f} \setminus \delta_{s} \) denotes \( \{ s \in S_{f} \setminus \forall s' \in \delta_{B} : s \neq s' \} \)
Lines 12 and 13 fill in the content of box symbols, where an n-box’s content formula gets negated.

To summarize, Algorithm 2 translates a given snapshot \( sn \), considered as the outer most frame, and a set of symbols \( S_B \), into a multi-sorted first-order formula \( \varphi_{sn} \) with free variables \( o_{s.id} \) for each \( s \in S_B \) and \( p.x_{s.id}, p.y_{s.id} \) for each \( s \in S_{sp} \).

**Algorithm 2: Translation of a Frame of the Spatial View**

1. **Function** `translateFrame`
2. **input**: frame \( f \), set of bound symbols \( S_B \)
3. **output**: first-order formula \( \varphi_f \)

4. \( \varphi_f \leftarrow \exists s \in S^e \setminus S_B \ s \in \text{type}(s.\delta) \)
   //object variable \( o_s \) for occurrence \( s \)

5. \( \exists s \in S^e \min-x_{s.id}, \max-x_{s.id}, \min-y_{s.id}, \max-y_{s.id} \in \mathbb{R} \)

6. \( \min-x_{s.id} \leq \max-x_{s.id} \wedge \min-y_{s.id} \leq \max-x_{s.id} \)
   //box symbol occurrence

7. \( \wedge s \in S_f \text{ alive}(o_{s.id}) \)
   //\( o_s \) is required to be alive

8. \( \wedge \exists s \in S^o \setminus S_s \emptyset_{s,\delta}(o_{s.id}) \)
   //features of \( s.\delta \)

9. \( \wedge s, s' \in S_f \times (S_f \cup S_{spans(f)}) T_{s,s'}(o_{s.id}, o_{s'.id}) \)
   //relative placement

10. \( \wedge l \in A_f P(o_{l.(s.id)}, o_{l.(s'.id)}) \)
    //arrow predicates

11. \( \wedge l \in L_f D(o_{l.(s.id)}, o_{l.(s'.id)}) \)
    //distance predicates

12. \( \wedge s \in S_f \text{ translateFrame}(s) \)
    //nested somewhere-box constraints

13. \( \wedge s \in S_f \neg \text{translateFrame}(s) \)
    //nested nowhere-box constraints

14. If \( f \) is negated then return \( \neg \varphi_f \); else return \( \varphi_f \);
7 Snapshot Charts and their Visualization

By now, we have introduced the visual elements of snapshots and explained how a snapshot can be translated into a first order formula. In the following, we show how snapshots are used within snapshot charts (SCs). SCs are used to describe an evolution over time. Within an SC a snapshot is used to specify (invariant) properties that hold for a time span strictly greater zero. Like the pages of a flip book, a sequence of snapshots then describes a story that evolves over time.

For instance Figure 45 tells the story of a car and cow. The car is driving in the bottom lane and heading towards a cow, which is standing in the same lane. The car then changes lane and circumvents the cow. When considering Figure 45 we intuitively suppose that the car (and also the cow and the lane) in all snapshots is the same but at different moments in time.

First the car is in front of the cow (sn₁), then the car changes lane (sn₂), then the car is in the other lane (sn₃) and so on.

![Figure 45: A sequence of snapshots can tell a story.](image)

At SCs, we have a new dimension that was missing at snapshots: Time. SCs combine snapshots in snapshot graphs and allow additional timing annotations. They thereby define a temporal logic predicate. To introduce SCs,

(s1) we will define snapshot graphs, where snapshots are composed to build a directed graph and, thereby, define constraints on trajectories.

(s2) In order to describe an evolution over time, we need to identify objects within a snapshot with objects within the next snapshot. To this end, we use object identities.

(s3) At SCs, we want to annotate duration of snapshot(subsequence)s and synchronize time points between concurrent evolutions. For this purpose, we use the visual elements hour glasses and time pins.

In the following, we will first give a brief overview of the new visual elements of SCs. We discuss them in more detail in the following subsections. Then, we formally translate SCs to temporal logic formulae in Section 8.

7.1 Overview

SCs provide visualizations for

1. snapshot graphs (cf. Section 7.2). Snapshot graphs result from composing snapshots. So, the simplest snapshot graph is made up of a single snapshot node, i.e., a node of a single snapshot. Snapshot graphs can be composed via the sequential concatenation, choice, concurrency and negation to build complex graph structures. Figure 46 illustrates for two snapshots how the resulting snapshot graphs are visualized. Usually, snapshots are connected
via directed edges to avoid ambiguities. Nevertheless, we may omit the edges, if this is unambiguous.

2. An SC can specify an implication. We provide a dedicated visualization of implications, the *premise-consequence chart*. There are two kinds of premise-consequence patterns, which differ by the time horizon of the premise:

a) "*the past implies a future consequence*"

This pattern basically allows to express that the future consequence is implied by what happened before. So "If *I did* sneeze, then *I will* get a tissue." matches this pattern.

b) "*the past and the future imply a future consequence*"

This pattern basically allows to express that the future consequence is implied by what happened before and also on what *will* happen. So, roughly, "If *I did* feel sick and if *I will* sneeze, then *I will* get a tissue." matches this pattern.
Figure 47 and Figure 48 illustrate the visualization of the two patterns. SC\textsubscript{past}, SG\textsubscript{fut.} and SG\textsubscript{conseq.} are snapshot graphs possibly with time annotations.

3. Snapshot graphs can be annotated with timing constraints.

To visualize these kind of constraints, we introduce the following visual elements

a) hour glasses (cf. Section 7.4.1)
   The hour glass is used to denote durations snapshot (sub)sequences.
   \textbf{Example 7.} Figure 7 shows an example SC, where an hour glass is used to specify a time duration of a snapshot. The car indicates for at most 2s, and

   \[
   \delta \leq 2s
   \]

   \begin{figure}[h]
   \centering
   \includegraphics[width=0.8\textwidth]{hour_glass.png}
   \caption{The hour glass specifies a duration.}
   \end{figure}

   then drives onto the lane separator. The hour glass takes the time of the second snapshot in this example. The duration is specified to be less or equals than 2s. Note that the car object in this example is represented by a symbol with one anchor only (indicated by \(\oplus\) at the middle of the car symbol).

\[\Diamond\]

b) time pins (cf. Section 7.4.2)
   Basically a time pin labels a point in time, i.e., the switching time between two snapshots or a time point during a snapshot. It provides a mean to synchronize concurrent developments.
   \textbf{Example 8.} Figure 50 shows an example SC, where a time pin is used to denote that the two concurrent developments (one at the top and the other at the bottom line) synchronize. We will discuss this SC in more detail on page 51.

   The snapshots of the top snapshot sequence are here referred to as \(sn_{1,1}\) and \(sn_{1,2}\), while the snapshot in the bottom snapshot sequence are referred to as \(sn_{2,1}\), \(sn_{2,2}\) and \(sn_{2,3}\). The time pin labels the switching time between \(sn_{1,1}\) and \(sn_{1,2}\) at the top as sync\textsubscript{1}. The dotted line labelled with sync\textsubscript{1} on top of \(sn_{2,3}\) in the bottom snapshot sequence denotes that \(sn_{2,3}\) happens at a time span that includes sync\textsubscript{1}.

\[\Diamond\]

4. In order to track an object along a snapshot graph, object symbols can be fixed to represent the same object identity along an evolution (cf. Section 7.5). To this end, we provide two equivalent syntactical means.

a) We allow labeling objects with identifiers. This is illustrated in Figure 51 or

b) we allow using a bulletin board (b-board) to declare symbols that represent the same object along an evolution. An example for an SC using a b-board is given in Figure 52.
Figure 50: A concurrency of two snapshot sequences with pinning of a synchronization time

Figure 51: An SC with identifiers.
The car and the two lanes are labeled with an id. So it can be clearly specified, that $sn_1$ and $sn_2$ refer to the same objects.

Figure 52: An SC and its b-board
The SC with b-board is equivalent to the SC of Figure 51.
After having presented this overview, we now introduce the syntactical elements of the SC visualization in more detail.

### 7.2 Snapshot Graphs

Snapshot graphs are directed graphs. Each snapshot graph has a start and an end point. The simplest non-empty snapshot graph is a snapshot node with a start and end point. The start point of a snapshot node is usually the left midpoint of the snapshot frame and the end point usually is the midpoint of the right side of the frame.

The composite of two snapshot graphs $SG_1$ and $SG_2$ build by

1. sequential concatenation as introduced in Section 7.2.1 and Section 8.1.3,
2. choice as introduced in Section 7.2.2 and Section 8.1.4,
3. concurrency as introduced in Section 7.2.4 and Section 8.1.5, and
4. negation as introduced in Section 7.2.3 and Section 8.1.2

is a snapshot graph.

#### 7.2.1 Concatenation (Sequence)

The result of a concatenation (sequence) of two snapshot graphs $SG_1$ and $SG_2$ basically expresses that first $SG_1$ holds and then directly $SG_2$, so there is a time $t$, such that $SG_1$ holds up to (but not including) $t$ and then $SG_2$ holds.

**Example 9.** An example of a sequence of two snapshots is given in Figure 53. The gray car does not indicate in the first snapshot and then starts indicating in the second snapshot. ♦

![Figure 53: Simple sequence](image)

The sequence of snapshot graphs $SG_1$ and $SG_2$ is annotated by connecting the end point of $SG_1$ with the start point of $SG_2$ via a directed edge. Start point of the sequence is the start point of $SG_1$ and endpoint of the sequence is the endpoint of $SG_2$.

#### 7.2.2 Choice (Disjunction)

A trajectory satisfies the choice of two snapshot graphs $SG_1$ and $SG_2$ if it satisfies $SG_1$ or $SG_2$. Visually, the choice of two snapshot graphs is built by introducing a new start point. From the start point an edge leads to each disjunct (i.e. start point of the alternative). Also an end point is created for the choice. From the end point of each disjunct a line leads to the new end point. In Figure 54 the new end point is the midpoint of the line joining the upper and lower snapshots.

**Example 10.** In this example, we use the choice to give an alternative and equivalent denotation of the snapshot sequence of Figure 53. At Figure 54 we replaced the "don't care"
a) The car in the upper lane indicates to its left and not to its right, or it neither indicates to its left nor to its right.

b) The car in the upper lane indicates to its left and to its right, or it does not indicate to its left but indicates to its right.

Figure 54: Choice of snapshot graphs

Figure 55 shows a more complex example of a snapshot sequence, where the two snapshot graphs of Figure 54 are sequentially composed. The snapshot sequence in Figure 55 is equivalent to the snapshot sequence in Figure 53.

7.2.3 Negation

The negation of a snapshot graph SG allows to express that everything may happen except the behavior specified by SG. The negation of a snapshot graph is visualized by placing a (red) frame around the graph, whose corners are connected via dashed (red) diagonal lines.

A negated snapshot graph provides a start point at its left border of the negation frame, preferably the midpoint, and likewise an end point at the right border.

Example 11. Figure 56 shows the negation of the snapshot sequence of Figure 53. Any behavior complies with this snapshot graph except behaviors that comply with the snapshot...
graph within the red box. So basically all trajectories satisfy the SC where there is not "a car in the upper lane that does not indicate to its right and then indicates to its right while staying in the same lane".

Figure 56: A negated snapshot sequence

Snapshot Negation vs Snapshot Graph Negation  
Note, that the negation of a snapshot (cf. page 13) should not be confused with the negation of a snapshot graph, even when the snapshot graph consists of a single snapshot only. Semantically these two are different because of the temporal extension of a snapshot graph, as we will illustrate via an example in Section 8.1.2.

7.2.4 Concurrency (Conjunction)

The concurrency of $SG_1$ and $SG_2$ specifies that simultaneously $SG_1$ and $SG_2$ have to hold. So concurrent behaviors can be nicely described.

Horizontal dashed lines separate the concurrent graphs, whereas vertical dashed lines indicate start and end, respectively. The start point is at the left dashed vertical line while the end point is at the right dashed line—as usual we preferably choose the midpoints.

Example 12. The snapshot graph of Figure 57 describes that the car in the lower lane first indicates only to its left and then sets both indicators, during this maneuver the car in the upper lane does not indicate to its right.

Figure 57: A concurrency of two snapshot sequences

There are two concurrent snapshot graphs. At the top there is only one snapshot. Concurrent to it is the sequence of two snapshots at the bottom.

7.3 Premise-Consequence Charts

We already briefly introduced the two different patterns for implications in Section 7.1. There we distinguished the "past implies a future consequence"—p.i.c for short—implication
from "the past and future imply a future consequence"—p.a.f.i.c for short—implication. Here we will give example SCs for the two implication types and discuss their usage.

7.3.1 "Past implies a future consequence" Implication

The "past implies a future consequence" pattern basically allows to express that the future consequence is implied by what happened before (cf. Section 7.1 item 2a). Note that the last snapshot of the premise snapshot graph has to hold up to a time at which contiguously the (future) consequence snapshot graph has to be satisfied. Figure 61a illustrates the visualization schema for "the past implies a future consequence" pattern.

The premise is specified via a (time-annotated) snapshot graph within a dashed hexagon. Right of the hexagon follows the consequence snapshot graph, which may also be annotated with time constraints.

Figure 58: An SC specifying a "the past implies future consequence" pattern. Roughly, the SC expresses "if first A and next B happened, then (after A next B) first E and next F will happen".

Example 13 (Past implies a future consequence). The premise-consequence chart of Figure 59 is an example of the "past implies a future consequence" pattern. At Figure 59 it is specified that when a car first approached a cow in its lane and next the neighboring lane is free, then as consequence, the car will change to the neighboring lane.

The "past implies future consequence" pattern allows specifications in terms of an implementation, where the (future) consequence is made dependent on the information already accumulated in the past.

Example 14 (Past implies false). The premise-consequence chart of Figure 60 is another example of the "past implies a future consequence" pattern. In this example the consequence is specified to be False. So, Figure 60 expresses that the behavior specified as premise is forbidden. Since the future consequence is false, only trajectories at which the premise does not match are allowed. Note that the car is represented by a symbol with one anchor only, which is indicated by ⊕ at the middle of the car symbol.

Figure 59: Exemplary SC for a premise-consequence charts with the pattern "The past implies a future consequence". It says: if after a car approached a cow in its lane and next the other lane is free, then the car will change to the free lane.
7.3.2 "Past and future imply a future consequence" Implication

This pattern basically allows to express that the (future) consequence is implied by what happened before and also by what will happen (cf. item 2b). Figure 61 illustrates the visualization schema for the "the past and the future imply a future consequence" pattern. The premise is specified within a dashed hexagon. It is divided into two parts: the past SG and the future SG. These two snapshot graphs are possibly time-annotated and divided by a dashed line. Right of the dashed hexagon follows the consequence snapshot graph. Semantically the consequence happens concurrently to the future part of the premise.

The future snapshot graph SG_fut. is given as part of the premise. Since the future is guaranteed by happen concurrently to the consequence, it—or a weaker form of it—might (re-)appears right of the premise in concurrency with the consequence without changing the semantics. The reappearance of the SG_fut. can be used to just highlight, that it logically is part of the premise and by definition will happen concurrently to the consequence. Also, if SG_fut. reappears right of the premise, synchronization information between SG_fut. and SG_conseq. can be annotated conveniently.

The omniscient eye is a visual element that can be used to represent SG_fut. right of the premise. We introduce this as an abbreviation for the future chart. Figure 63 illustrates how the omniscient eye is used. Right of the premise, on top of the consequence, a bar with the omniscient eye appears. The omniscient eye represents the future part of the premise. Its extend is specified via the bar. So, in the sketched example SG_fut. is fully concurrent to SG_conseq. The SC of Figure 63 is hence equivalent to the SCs of Figure 61.
The premise is satisfied, if we could observe past and will be able to observe future. Then conseq is implied to hold concurrently to the future. More precisely, if past holds from $t_0$ up to a time $t \geq t_0$ and if from $t$ to $t'$ future holds, then conseq is implied to hold from $t$ to $t'$.

The snapshot graphs of the premise are represented via the green boxes "past" and "future". The consequence—concurrent to the future—is implied by the pair of past and future.

Figure 63: The SC is equivalent to the SC of Figure 61b.
Example 15. Figure 64 gives an example of the premise-consequence chart with the p.a.f.i.c. pattern. It describes the rule "If ego wants to change lane and no other car will be too close during the lane change, then it changes lane and indicates during the lane change."

The past part of premise describes that initially the car in the upper lane wants to do a lane change to its right. We call the gray car ego. In this example we assume that there is a state predicate "wantsToChangeRight" with an appropriate interpretation on ego's state.

The future snapshot graph of the premise describes that the car changes from the upper lane to the middle lane and there is no car closer than the safety distance sd. Here, dist x abbreviates o-car.x - grey-car.x and sd abbreviates the expression for the safety distance that refers to the cars' velocities and accelerations.

The consequence snapshot graph says that ego changes from the upper lane to the middle lane and indicates to its right. Note, the consequence snapshot graph "extends" the future snapshot graph of the premise, i.e., the snapshots of consequence and future differ only in indication lights of the gray car. Since consequence and future are in concurrency, the only added constraint by the consequence is, hence, the requirement of setting indication lights.

This pattern is very useful for the specification of system requirements. We can use this pattern for instance to express a requirement for a certain future (e.g. have fetched a tissue in advance when you will sneeze).\footnote{You can either always have a tissue or just when you feel your nose is itchy. The prediction of when you actually need to get a tissue is left to you.}

Note that the "past implies a future consequence" (p.i.c.) can be seen as a special case of "past and the future imply a future consequence" (p.a.f.i.c.), where the future constraint is True. Hence, the visualization of "past implies a future consequence" pattern in Figure 61 omits the future part.

7.4 Timing Constraints

Within a snapshot chart, timing constraints can be defined. We visualize timing constraints via hour glasses and time pins. Both implicitly declare time points to anchor constraints.

7.4.1 Hour Glass

A (labelled) timer can be started when a snapshot is entered and a timer can be stopped at the end of the same snapshot or at the end of a snapshot further on the path of the snapshot graph. A started timer may only be stopped along a path from the start timer and within the same negation scope.
Timers are visualized by an hour glass. To annotate the start, a full hour glass is depicted above the left border of the entered snapshot. The start hour glass is labeled with an identifier for the timer. The end is annotated by an hour glass with sand at its bottom. This timer end symbol is placed at the end of the respective snapshot. An arithmetic constraint is specified at the end symbol.

Example 16. We have already seen an example of a timer within the premise of the snapshot chart in Figure 60. We reproduce in Figure 65 the premise for the readers convenience. The hour glass is used to specify a time duration of a snapshot.

Figure 65: Hour glass annotating a duration.

The car first does not indicate to its right, then it indicates for less than \(2s\) and then drives onto the lane separator. The first hour glass takes the time of the first snapshot and the second hour glass takes the time of the second snapshot in this example. The duration is specified to be less than \(2s\).

7.4.2 Time Pin and Synchronisation Bar

We can relate points in time of concurrent graphs via a time pin, \(\circlearrowleft\). More precisely, time pins are a mean to introduce a label for a point in time. Its scope is the complete snapshot chart.

Basically we label (i) the switching time between two snapshots or (ii) a time point during a snapshot. For (i) a time pin is placed at the start or end point of a snapshot, say \(sn_m\), for (ii) a time pin is placed above a snapshot at a synchronization bar, that is a dotted horizontal line. In both cases the pin can be labeled with an identifier, say \(\pi\). By referring to the same time pin \(\pi\) at concurrent paths, we can hence express synchronization (at time point \(\pi\)).

To visually specify synchronization, we place a time pin \(\pi\) somewhere according to (i) or (ii) and at a concurrent snapshot we can

1. synchronize the switching point of a snapshot, say \(sn_m\), with \(\pi\) by placing a dotted vertical line segment (possibly labeled with \(\pi\)) at a \(sn_m\)'s start point, and we then
2. synchronize \(sn_m\) to be active at least at time \(\pi\) by placing the synchronization bar above \(sn_m\) and connecting it to \(\pi\) (or labeling it with \(\pi\)).

Filled circles at the start (end) of a synchronization bar denote that the start time point is included, while a missing or empty circle means that the start/end time is excluded.

If this enhances the readability, we additionally connect the time pin and the synchronization time lines via a dotted line.

In Example 17 we present an instance of item 1 where start times of two concurrent snapshot graphs are synchronized by means of a time pin. In Example 18 we present an instance of item 2 where start time of a snapshot is synchronized to take place during a concurrent snapshot by mean of a time pin and a synchronization bar. To exemplify the synchronisation bar syntax, example 18 shows two variants of the synchronisation bar, one variant without start and end circles and the other variant has an empty start circle and a filled end circle.
Example 17. The snapshot graph in Figure 66 uses the time pin $\text{sync}_1$ to specify that the car starts to indicate simultaneously when changing to its right.

Example 18. We have already presented an example using time pins in Figure 50, which we reproduce here in Figure 67a. It specifies that the car in the upper lane does not indicate to its right while the car in the lower lane first indicates to its left and not to its right and then it does not indicate at all for more then two seconds. Then the upper car starts indicating to its right while the car in the bottom lane still does not set any turn indicators.

Figure 66: The conjunction of two snapshot sequences with pinning of start times for synchronization

Figure 67: A conjunction of two snapshot graphs using a time pin to synchronize start time and synchronization bar.

The time pin $\text{sync}_1$ refers to the switching time between $\text{sn}_{1,1}$ and $\text{sn}_{1,2}$ (at the top snapshot graph path). So $\text{sync}_1$ is the time when the car in the upper lane sets its indicators. The bottom snapshot graph path is specified to synchronize with the top as follows: First $\text{sn}_{2,1}$, then $\text{sn}_{2,2}$ and $\text{sn}_{2,3}$ of the bottom line have to hold. Only when $\text{sn}_{2,3}$ already holds, then $\text{sn}_{1,2}$ of the top lane starts to hold.

While in Figure 67a the start time of $\text{sn}_{1,2}$ may be equals the start time of $\text{sn}_{2,3}$, in Figure 67b the start time of $\text{sn}_{1,2}$ has to be greater than the start time of $\text{sn}_{2,3}$.

Time pins can—more generally—be attached to

1. any start/end point of a snapshot graph\textsuperscript{13} and

\textsuperscript{13}Each snapshot graph has a start and end point (cf. 7.2).
2. Time pins can be used to label a time point between start and end point of a snapshot graph.

We illustrate the use of time pins within snapshot graphs in Example 19.

Example 19. In Figure 68 and Figure 69 we illustrate how time pins can be used within snapshot graphs.

The snapshot graph of Figure 68 specifies that first $s_{n1}$ has to hold and then $s_{n2}$, then $s_{n3}$. The switching time between $s_{n1}$ and $s_{n2}$ equals $p_1$. In parallel to $s_{n1}; s_{n2}; s_{n3}$, first the negation of "first $s_{n7}$ holds and then $s_{n8}$ and $p_1$ is some time at which $s_{n7}; s_{n8}$" holds and then $s_{n9}$ holds. There are many ways to satisfy this snapshot graph, in particular we mention, that it is satisfied by a trajectory like $\tau$ sketched in Figure 70 where $s_{n1}; s_{n2}; s_{n3}$ and simultaneously $s_{n7}; s_{n8}; s_{n9}$ holds but $p_1$ is not during $s_{n7}; s_{n8}$. Note that before time point $p_1$, $s_{n1}$ has to hold and at $p_1$, $s_{n2}; s_{n3}$ has to hold. Accordingly, $t_s$ is the earliest time that could be chosen for $p_1$ at $\tau$, but then $s_{n9}$ already holds. Hence the constraint $\neg \psi$ with $\psi$ saying "first $s_{n7}$ holds and then $s_{n8}$ holds and $p_1$ is some time at which $s_{n7}; s_{n8}$" is satisfied. The negation holds due to the violation of the conjunct "$p_1$ is some time at which $s_{n7}; s_{n8}$".

The snapshot graph of Figure 69 specifies that first $s_{n1}$ has to hold and then $s_{n2}$ or $s_{n6}$ holds. $p_1$ is some time during the disjunction (but neither start nor end time). In parallel to $s_{n1}; (s_{n2} \lor s_{n6})$, it holds that first $s_{n8}$ and then $s_{n9}$ holds, where $p_1$ is the switching time between $s_{n8}$ and $s_{n9}$.

![Figure 68: Time pins and negation](image1)

![Figure 69: Time pins and disjunction](image2)

![Sketch of snapshot validities of along $\tau$ satisfying the SG of Figure 68](image3)

![Figure 70: figure](image4)
7.5 Object Identities

SCs are used to specify an evolution over time (via, e.g., a snapshot sequence). To describe that an object evolves along a sequence, we need to identify the object in the different snapshots. We need to reason about object identities ("Is it the same car in two different snapshots, or not?"). The importance of identities is highlighted by Example 20.

Example 20. The snapshot sequence (A) of Figure 71 can be read as: (i1) First there is a car that drives at least $100\text{km/h}$ at a distance closer than $2\text{m}$ to an obstacle, until the car collides with the obstacle. So, the snapshot sequence describes a severe collision scenario.

But if we do not assume that the objects (lanes, cars and obstacles) are identical in $s_{n_1}$ and $s_{n_2}$, then the snapshot sequence can be understood to merely describe, that (i2) first a car $c_1$ drives at least $100\text{km/h}$ at a distance less than $2\text{m}$ close to an obstacle $o_1$ and then somewhere there is a car $c_2$ right in front of an obstacle $o_2$. According to (i2), the car $c_1$ may change lane to avoid collision and car $c_2$ may be parking in front of $o_2$. At (B) of Figure 71 we adopted the visualization of (A) just to make interpretation (i2) more intuitive. Although it may be less obvious, (A) allows trajectories of the type (i2), if we do not fix symbols to object identities.

To keep track of objects, we can declare identities for object symbols. To this end, we provide two equivalent syntactical means.

1. We allow to label objects with identifiers. We label an object with a special predicate (cf. Section 5.6). The predicate has the form

$$id=\text{unique-identifier},$$

where unique-identifier is unique within the TSC. The use of these id labels is illustrated in Figure 72.

Figure 72: An SC with identifiers.

---

14Note, that we require that the snapshots of a sequence contiguously hold during a trajectory.
2. We use a b-board, where we declare the symbols that represent the same object along an evolution. Each SC may have its own b-board. As objects evolve along a sequence and also change state visually, also the symbol may be modified (cf. Section 5.2.3). Hence we declare the allowed symbol transformations, i.e., which variations of the symbol still represent the object identity but a different state.

An example for an SC using a b-board is given in Figure 73.

![Figure 73: An SC and its b-board](image)

The attentive reader has noticed that in the examples presented so far, we neither labeled object identities nor did we use a b-board to declare object identities. We silently did assume a b-board as presented in Figure 73.

An example where a symbol with fixed id and a symbol without fixed id is used, is given in Figure 64 and reproduced here in Figure 74 for convenience. The SC shows a premise-consequence chart, that describes that if the ego car first indicates right and wants to change \( (s_{n_1}) \) and if then no other car will get within safety distance \( sd \) to ego while ego changes lanes \( (s_{n_2}, s_{n_3}) \), then ego will change lane and indicate. In this example the identity of the gray car is fixed to represent the same object, which we called ego. In contrast, the white car represents just an instance of the class Car. Each time the symbol is used it may represent a different object of the Car class.

8 From Snapshot Charts to Multi-Sorted Timed First-Order Logic

By now we have introduced the syntactical elements of SCs. In the following, we show how SCs are translated to a temporal logic formula.

![Figure 74: Example of an SC where symbols are used to represent a fixed object (gray car) and no fixed object (white car).](image)

The SC roughly specifies the rule “Change lane and indicate, if you want to change lane and no other car will be too close during the lane change.” Therefore the gray car is fixed to represent a car that changes lane, while the white car represents any other cars.
8.1 Snapshot Graphs

We start by deriving a formula for snapshot graphs. For the translation into a formula, we assume that each snapshot graph $SG_{id_i}$ has assigned an identifier $id_i$.

Similarly to the definition of snapshot graphs (cf. page 43), the formula corresponding to a snapshot graph is compositionally built. The snapshot node formulas are composed in an appropriate way. To compose the snapshot node formulas so that they contiguously hold along a path of a snapshot graph, we introduce in analogy to the start and end nodes of a snapshot graph, a pair of free variable $b_{id_i}$ and $e_{id_i}$ for each snapshot graph $SG_{id_i}$ as handles for the start and end time of a snapshot. These handles allow us to express the contiguity.

8.1.1 Snapshot (Node)

We defined in Section 6 the semantics of elements within a snapshot and at Section 6.7, page 32 we presented a translation scheme, that derives a snapshot formula $\varphi_{id_{sn}}$, that is a first-order formula for a single snapshot $sn$ with identifier $id_{sn}$.

A snapshot node $SG_{id_{sn}}$ is required to hold for a certain while. We hence extend the snapshot formula $\varphi_{id_{sn}}$ as follows: Let $b_{id_{sn}}$, $e_{id_{sn}}$ be unique variable names.

$$G_{id_{sn}}(b_{id_{sn}}, e_{id_{sn}}) = [b_{id_{sn}}, e_{id_{sn}}] \varphi_{id_{sn}},$$

where $b_{id_{sn}}$ and $e_{id_{sn}}$ are free variables that refer to the begin and end time of the time span within which the snapshot formula $\varphi_{id_{sn}}$ holds.

**Example 21.** Let us consider for example, the two snapshot nodes at Figure 75 and Figure 76.

![Figure 75: A snapshot showing a car at the upper lane that does not indicate to its right.](image)

![Figure 76: A snapshot showing a car at the upper lane that indicates to its right.](image)

In the following we will derive corresponding temporal formulas. We first determine the snapshot formulas according to Section 6:

For snapshot $sn_1$ at Figure 75, the presence of the (car and lane) symbols introduces $E := \exists c_1 \in Car, l_1, l_2 \in Lane \land alive(c_1) \land alive(l_1) \land alive(l_2)$. The symbol features are captured as $O := \neg c_1. indicateRight$. The topology of symbols implies $T := c_1. y < l_1. y \land c_1, y > l_2. y \land c_1. y = l_2. y$. Note that $T$ is the minimal set of constraints w.r.t. the assumption that $a. y > o. y$ and $o. x > a. x$ for any object $o$. $sn_1$ hence is translated to $\varphi_1 := E \land \emptyset \land T$.

Further note, that we assume that the lane symbol does not specify anchors $\overline{x}$, $\overline{x}$, so that no topology constraints w.r.t. $\overline{x}$ and $\overline{x}$ between objects are implied.

Similarly snapshot $sn_2$ at Figure 76 is translated—with the only difference that the indicators are required to be switched on, that is $O_2 := c_1. indicateRight$ has to hold instead of $\emptyset$. So the snapshot nodes $SG_{sn_1}$, $SG_{sn_2}$ with the snapshots $sn_1$ and $sn_2$ mean

$^{15}$alive for a lane might sound strange. Semantically this expresses that there actually is a lane, since our semantic model has "potential" or "not materialized" objects. A created object satisfies alive and a "not materialized" object satisfies $\neg alive$. See Section 4 for more information.
• $G_{s_n_1}(b_{s_n_1}, e_{s_n_1}) = \Box_{b_{s_n_1}, e_{s_n_1}} \varphi_{s_n_1}$.
• $G_{s_n_2}(b_{s_n_2}, e_{s_n_2}) = \Box_{b_{s_n_2}, e_{s_n_2}} \varphi_{s_n_2}$.

So both snapshots have to hold for a while. The free variables $b_{s_n_1}, e_{s_n_1}, b_{s_n_2}, e_{s_n_2}$ are the handles to start and end times of the respective snapshot nodes.

8.1.2 Negation

A negated snapshot graph $SG_\neg$ allows us to express that everything may happen except the behavior specified by $SG$, that is the graph within the negation frame. To derive a formula for the negation, we basically negate the formula of $SG$.

$$G_{SG_\neg}(b_{SG_\neg}, e_{SG_\neg}) = \neg G_{SG}(b_{SG}, e_{SG}) \setminus \{b_{SG_\neg}, e_{SG_\neg}\}.$$ 

**Example 22.** Figure 77 shows a simple (unnegated) snapshot graph with a single snapshot node $s_n_3$ and Figure 78 the negated snapshot node $s_n_3$.

The snapshot node $s_n_3$ translates to $G_{s_n_3}(b_{s_n_3}, e_{s_n_3}) := \Box_{b_{s_n_3}, e_{s_n_3}} (\exists c \in Car : alive(c) \land \neg c.\text{indicateRight}). Figure 77 shows $SG_{\neg s_n_3}$, the negation of the snapshot graph of Figure 77. $SG_{\neg s_n_3}$ of Figure 78 is translated to $G_{\neg s_n_3}(b_{\neg s_n_3}, e_{\neg s_n_3})$ which is

$$\neg \Box_{b_{\neg s_n_3}, e_{\neg s_n_3}} (\exists c \in Car : alive(c) \land \neg c.\text{indicateRight}).$$

Given a begin time $b_{\neg s_n_3}$ and an end time $e_{\neg s_n_3}$ with $b_{\neg s_n_3} < e_{\neg s_n_3}$, a trajectory satisfies $G_{\neg s_n_3}$ if there is at least a time point $t \in [b_{\neg s_n_3}, e_{\neg s_n_3}]$ at which $(\exists c \in Car \land \neg c.\text{indicateRight})$ holds—but $(\exists c \in Car \land \neg c.\text{indicateRight})$ may possibly hold at all other time points $t' \in [b_{\neg s_n_3}, e_{\neg s_n_3}] \setminus \{t\}$.

To highlight the difference of snapshot negation and negation of a snapshot graph, let us now consider the following example.

**Example 23.** Figure 79 shows a snapshot node of the negated snapshot $s_n_5$. The snapshot

graph $SG_{s_n_5}$ of just snapshot $s_n_5$ (Figure 79) translates to $G_{s_n_5}(b_{s_n_5}, e_{s_n_5}) = \Box_{b_{s_n_5}, e_{s_n_5}} \neg (\exists c \in Car : alive(c) \land \neg c.\text{indicateRight}).$ So $SG_{s_n_5}$ expresses that at all times between its start and end time points $(\exists c \in Car : alive(c) \land \neg c.\text{indicateRight})$ does not hold (there is no car indicating right the whole time).
8.1.3 Sequence

Let \( SG \) be the sequence of two snapshot graphs \( SG_1 \) and \( SG_2 \). The sequence of \( SG_1 \) and \( SG_2 \) basically expresses that first \( SG_1 \) holds and then directly \( SG_2 \), so there is a time \( t \), such that \( SG_1 \) holds up to (but not including) \( t \) and then \( SG_2 \) holds.

\[
G_{SG}(b,e) = \exists t' : b < t' < e \land (G_{SG_1}(b_1,e_1)[b_1 \setminus t] \land G_{SG_2}(b_2,e_2)[b_2 \setminus t',e_2])
\]

where \( \phi[x \setminus y] \) denotes the renaming of variable \( x \) to \( y \) within formula \( \phi \). So at \( SG \) from \( b_1 \) up to \( t' \) \( SG_1 \) holds and from \( t' \) to \( e_2 \) \( SG_2 \) holds.

We translate sequences from left to right, so \( SG_1 \) is not the result of concatenation. For snapshot charts without time annotations, the order of translation can be chosen arbitrarily. Otherwise, the order from left to right allows to insert the start and end timers appropriately.

**Example 24.** We reproduce here the example of a sequence of two snapshots as given in Figure 53 and present the derived formula. We already derived the formulas for the two constituent snapshot nodes \( sn_1 \) and \( sn_2 \) at example 21. So that we derive for the snapshot graph \( SG \) of Figure 80,

\[
G_{SG}(b,e) = \exists t' : b < t' < e \land (G_{sn_1}(b,t')[b \setminus t'] \land G_{sn_2}(t',e)[t',e])
\]

This—omitting the lane predicates—extends to

\[
G_{SG}(b,e) = \exists t' : b < t' < e \land \square_{[t',e]}(\exists c \in Car : alive(c) \land \neg c.\text{indicateRight}) \land \square_{[t',e]}(\exists c \in Car : alive(c) \land c.\text{indicateRight}).
\]

So this sequence basically expresses that first a car does not indicate to the right, then a car indicates to the right. The attentive reader may make the objection that this predicate is satisfied if different cars indicate. This is true, since we did not fix the car’s identity via, e.g., a bulletin board (b-board). As we will see at Section 8.4 with the car symbol at the b-board, the derived predicate will be

\[
G_{SG}(b,e) = \exists t' : b < t' < e \land \square_{[t',e]}(alive(c) \land \neg c.\text{indicateRight}) \land \square_{[t',e]}(alive(c) \land c.\text{indicateRight}).
\]


8.1.4 Choice

The choice of two snapshot graphs describes alternatives. Given \( SG \) is a snapshot graph made up by the choice of \( SG_1 \) and \( SG_2 \), then

\[
G_{SG}(b,e) = G_{SG_1}(b_1,e_1)[b_1 \setminus b,e_1] \lor G_{SG_2}(b_2,e_2)[b_2 \setminus b,e_2].
\]

So \( SG \) is satisfied if between start and end time \( SG_1 \) or \( SG_2 \) holds.

8.1.5 Concurrency (Conjunction)

The concurrency allows to describe that two snapshot graphs \( SG_1 \) and \( SG_2 \) hold simultaneously. Let \( SG \) be the concurrency of two snapshot graphs \( SG_1, SG_2 \).
\[ G_{SG}(b, e) = G_{SG_{1}}(b_1, e_1)[b_1 \backslash b, e_1 \backslash e] \land G_{SG_{2}}(b_2, e_2)[b_2 \backslash b, e_2 \backslash e] \]

Between the start and end times of SG, SG\textsubscript{1} and SG\textsubscript{2} have to hold.

### 8.2 Premise Consequence Charts

Premise consequence charts can be used to specify "if then" rules. In the following we will only consider the "the past and the future imply a future consequence" pattern, because "the past implies a future consequence" pattern is a special case of the further where SG\textsubscript{fut.} is the True snapshot.

Let the premise consequence chart specify the past via snapshot graph SG\textsubscript{past}, the future part of the premise via SG\textsubscript{fut.}, and the consequence via snapshot graph SG\textsubscript{conseq}. The pattern means, that if first SG\textsubscript{past} has been observed and right after that SG\textsubscript{fut.} happens, then the system is specified to behave according to SG\textsubscript{conseq} while \( \varphi_{fut} \) happens. The semantics of a conditional snapshot graph is given by a formula with three free variables: the begin time of the SC and its end time, as usual, and the switching time \( t_s \) between past and the pair of future and consequence, which start simultaneously.

\[
G_{SC}(b, e, t_s) = (G_{SG_{past}}(b, t_s) \land G_{SG_{fut}}(t_s, e)) \Rightarrow G_{SG_{conseq}}(t_s, e)
\]

![Figure 81: Simple premise-consequence chart. If the car has been driving at more than 50 km/h and next will face a cow at its lane, then the car decelerates maximally.](image)

**Example 25.** The SC of Figure 81 is translated to \( G_{SC}(b, e, t) = (G_{SG_{past}}(b, t) \land G_{SG_{fut}}(t, e)) \Rightarrow G_{SG_{conseq}}(t, e) \) where

- we abbreviate (for this example) by
  - \textbf{between}(o, o') the predicate \( o \bar{y} < o' \bar{y} \land o \bar{x} > o' \bar{x} \) and it means that object \( o \) is vertically between (the y anchors of) \( o' \);
  - \textbf{infront}(o, o') the predicate \( o \bar{x} < o' \bar{x} \) and it means that object \( o \) is horizontally left of \( o' \);
  - \textbf{zdist}(o, o') the expression \( |o' \bar{x} - o \bar{x}| \).

- The past formula is then \( G_{SC_{past}}(b, t) := b < t \land \square_{[b,t]}(\exists c_1 \in \text{Car} : \exists l_1 \in \text{Lane} : \text{alive}(c_1) \land \text{alive}(l_1) \land \text{between}(c_1, l_1) \land c_1.\text{vel} \geq 50\text{km/h}) \)

- The future formula is \( G_{SC_{fut}}(t, e) := t < e \land \square_{[t,e]}(\exists c_1 \in \text{Car} : \exists l_1 \in \text{Lane} \exists c_2 \in \text{Cow} : \text{alive}(c_1) \land \text{alive}(l_1) \land \text{alive}(c_2) \land \text{between}(c_1, l_1) \land \text{between}(c_2, l_1) \land \text{zdist}(c_1, c_2) \leq bd_{50} \land \text{infront}(c_1, c_2)) \)

- \( G_{SC_{conseq}}(t, e) := t' < e \land \square_{[t,e]}(\exists c_1 \in \text{Car} : l_1 \in \text{Lane} : \text{alive}(c_1) \land \text{alive}(l_2) \land \text{between}(c_1, l_1) \land c_1.\text{accel} = \text{MAX\_DECEL} \).
So if a car drives 50 km/h between the time points \( b \) and \( t \) and if next, that is between \( t \) and \( e \), a car approaches a cow closer than comfortable braking distance, \( bd_{SG} \), then a car has to apply maximal deceleration between \( t \) and \( e \).

Note that above again a car object is existentially quantified anew. Declaring object identities allows us to speak about the car that decelerates when approaching a cow (cf. Section 8.4).

8.3 Timing Constraints

To encode the timing constraints as introduced at Section 7.4 we modify slightly the translations of the snapshot graph by inserting the timer constraints at appropriate positions within the formula of the snapshot graph.

8.3.1 Hour Glass

According to Section 7.4.1 a (labelled) timer can be started when a snapshot is entered and a timer can be stopped at the end of the same snapshot or at the end of a snapshot further down the path. The start and stop of a timer is denoted via \( \Xi \), the start hour glass symbol, and \( \Box \), the end hour glass symbol, respectively. At the start hour glass a name for the timer is specified, say \( \delta \), and at the end hour glass a constraint on \( \delta \) is specified, say \( \psi \). Let the timer be started at snapshot \( sn_m \) and stopped at a snapshot \( sn_n \).

Let \( G_{SG}(b, e) \) be the formula derived for the snapshot graph \( SG \). For both points in time \( G_{SG} \) introduces variables that represent the start time of \( sn_m \) and the end time of \( sn_n \), \( b_m \) and \( e_n \). An hour glass timing constraint is derived from \( \psi \) by substituting \( \delta \) by \( e_n - b_m \). This timing constraint is inserted into the formula \( G_{SG} \). We hence modify the translation of snapshot \( sn_n \) – the snapshot with an end hour glass at its end point: We replace \( \delta \) in \( \psi \) by \( e_n - b_m \), \( \psi' := \psi[\delta \leftarrow e_n - b_m] \), and \( \psi' \) becomes a conjunct of the node formula of \( sn_n \), \( \Box_{[b_m,e_n]} \varphi_n \land \psi' \).

The modification of \( sn_n \)'s translation (cf. 8.1.1):

\[
G_{id_{sn_n}}(b_n, e_n) = \Box_{[b_n,e_n]} \varphi_{id_{sn_n}} \land \psi[\delta \leftarrow e_n - b_m].
\]

Example 26. Figure 82 shows an example SC using an hour glass. We already presented this example at p. 7 and use it here to demonstrate the encoding of the time constraint.

\[\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\bigcirc \quad \Xi \quad 2s \quad \Box \quad \bigcirc
\end{array}
\end{array}
\end{array}\]

Figure 82: The hour glass specifies a duration.

The car indicates for at most 2s, and then drives onto the lane separator. The hour glass takes the time of the second snapshot in this example. The duration is specified to be less than or equals 2s.

\[\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\bigcirc \quad \Xi \quad 2s \quad \Box \quad \bigcirc
\end{array}
\end{array}
\end{array}\]

We sketch the formula for the snapshot graph without the hour glass as \( SG(b, e) = \exists t_1 (b < t_1 < e \land \Box_{[t_1,t_2]} s_{n_1} \land \exists t_2 (t_1 < t_2 < e \land \Box_{[t_2,t_3]} s_{n_2} \land \Box_{[t_2,e]} s_{n_3})) \). The constraint \( \psi \) is in our example \( \delta \leq 2s \). To derive \( \psi' \), we replace \( \delta \) by \( t_2 - t_1 \) , i.e., the end time variable \( t_2 \) of \( sn_2 \) minus the start time variable \( t_1 \) of \( sn_2 \). So we get \( \psi' = (t_2 - t_1) < 2s \). After insertion
of \( w' \) as conjunct to \( sn_2 \) we get

\[
SG'((b, e)) = \exists t_1 (b < t_1 < e \land \Box_{[t_1, e]} sn_1 \land \exists t_2 (t_1 < t_2 < e \land \Box_{[t_1, t_2]} sn_2 \land (t_2 - t_1) < 2s \land \Box_{[t_2, e]} sn_3)).
\]

### 8.3.2 Time Pin

We introduced time pins already at Section 7.4.2. We summarize how a time pin can be used within a snapshot graph in the following and then present the formal semantics.

A time pin can be placed at the start or end point of a snapshot graph and be labeled with an identifier, say \( \pi \), or it can be placed above a snapshot graph at a synchronization bar, that anchors at start and end points of a snapshot graph. The synchronization bar may be decorated by filled circles at its start (end) to denote that the start (end) time is included. An empty circle or no circle at its start (end) means that the start (end) time is excluded.

Let time pin \( \pi \) occur at a start or end point of \( SG_1 \), where \( SG_1 \) is subgraph of a more complex snapshot graph \( SG \). Let \( SG_2 \) be concurrent to \( SG_1 \), as for instance sketched in Figure 83.

![Figure 83: \( SG_1 \) and \( SG_2 \) are concurrent. Time pin \( p \) is used to specify synchronisation of the end points of \( SG_1 \) and \( SG_2 \).](image)

Within a \( SG_2 \) we can

1. synchronize a start or end point with \( \pi \) by placing (i) a dotted vertical line segment (possibly labeled with \( \pi \)) at \( SG_2 \)'s start or end point or (ii) by placing again the time pin symbol \( \pi \) at \( SG_2 \)'s start or end point, and we can

2. synchronize the duration of \( SG_2 \) by placing the synchronization bar above \( SG_2 \) (labeled with \( \pi \), if this enhances readability) which anchors at \( SG_2 \)'s start and end point.

A time pin—like the position pin (cf. Section 8.5, Section 6.3.1)—introduces a variable of global scope. We existentially quantify a time variable \( t_{s.id} \) for a time pin occurrence.

\[ \exists t_{s.id} \]

Within the overall formula that represents the SC, this snipped is a prefix, so that the scope of quantification covers the formula derived for the SC. In the sequel we explain how we link the time variable \( t_{s.id} \) to the time constraints specified by use of the time pins.

We first consider item 1 and remind the reader that we already presented an example of item 1 in Figure 66, which is reproduced for the reader’s convenience in Figure 84 of Example 27.

**Example 27.** The snapshot graph in Figure 84 uses the time pin sync1 to specify that the car starts to indicate when reaching the the lane separator during a change to its right.

In the following, let \( G_{SG} \) be the formula derived for \( SG \) where a time pin is used in subgraphs \( SG_1 \) and \( SG_2 \). \( G_{SG} \) uses a variable (among others) to represent \( SG_1 \)'s and \( SG_2 \)'s start and end points. These variables are introduced by the snapshot nodes to represent their start and end time. They are renamed to also the represent the start and end point of the snapshot graph, when the snapshot graph’s formula is derived from the snapshot node formulas (cf. 8.1).
Let $t_m$ be the variable representing the point at which $\pi$ anchors in $SG_1$ and let $t_n$ be the variable at which $\pi$ anchors in $SG_2$.

The synchronization with $\pi$ means that the time $t_m$ equals the time $t_n$. So, we insert the expression $t_{s.id} = t_m$ conjunctively into the snapshot graph formula of $SG_1$ and $t_{s.id} = t_n$ into the snapshot graph formula $SG_2$.

Let us now consider item 2. Figure 85 reproduces for convenience of the reader the example SC already given in Figure 50, where a time pin and a synchronization bar is used to synchronize the two concurrent developments.

Synchronizing with $\pi$ means that the time $t_m$ is somewhere between the start time, $b_n$, and end time $e_n$ of $SG_2$. So, we insert the expression $t_{s.id} = t_m$ into the snapshot graph formula of $SG_1$, and the expression $b_n < t_{s.id} < e_n$ into the formula $SG_2$ of snapshot graph $SG_2$.

**Example 28.** The snapshots of the top snapshot sequence are here referred to as $sn_{1,1}$ and $sn_{1,2}$, while the snapshot in the bottom snapshot sequence are referred to as $sn_{2,1}$, $sn_{2,2}$ and $sn_{2,3}$. The time pin labels the switching time between $sn_{1,1}$ and $sn_{1,2}$ at the top as $sync_1$. The dotted line labelled with $sync_1$ on top of $sn_{2,3}$ in the bottom snapshot sequence denotes that $sn_{2,3}$ happens at a time span that includes $sync_1$. We illustrate the translation of the

---

$$\delta \geq 2s$$

---

In other words we replace $SG_1$ by $t_{s.id} = t_m \land SG_1$.
time pin. Without the time pin, the formula is the conjunction of the top snapshot graph \( SG_{\text{top}} \)—consisting of \( s_{n1,1} \) and \( s_{n1,2} \)— and the snapshot graph \( SG_{\text{bottom}} \)—consisting of \( s_{n2,1} \) to \( s_{n2,3} \). We sketch the formula for the top snapshot graph as \( SG_{\text{top}}(b, e) = \exists t_{1}(b < t_{1} < e_{t} \land \Box_{[b, t_{1}]} s_{n1,1} \land \Box_{[t_{1}, e]} s_{n1,2}) \). Analogously, to Example 26 we sketch the formula for the bottom snapshot graph \( (s_{n2,1} \to s_{n2,3}) \) \( SG_{\text{bottom}}(b, e) = \exists t_{1}(b < t_{1} < e_{b} \land \Box_{[b, t_{1}]} s_{n2,1} \land \exists t_{2}(t_{1} < t_{2} < e_{b} \land \Box_{[t_{1}, t_{2}]} s_{n2,2} \land (t_{2} - t_{1}) < 2s \land \Box_{[t_{2}, e_{b}]} s_{n2,3}) \). Their conjunction is sketched as

\[
SG(b, e) = SG_{\text{top}}[b, \neg b, e_{t} \land e] \land SG_{\text{bottom}}[b, \neg b, e_{b} \land e] = \exists t_{1}(b < t_{1} < e \land \Box_{[b, t_{1}]} s_{n1,1} \land \Box_{[t_{1}, e]} s_{n1,2}) \land \\
\exists t_{1}(b < t_{1} < e \land \Box_{[b, t_{1}]} s_{n2,1} \land \exists t_{2}(t_{1} < t_{2} < e \land \Box_{[t_{1}, t_{2}]} s_{n2,2} \land (t_{2} - t_{1}) < 2s \land \Box_{[t_{2}, e]} s_{n2,3})).
\]

The time pin constraint refers to the start time of \( s_{n1,2} \) and the duration of \( s_{n2,3} \). We existentially quantify \( t_{p} \) to represent the pin’s time. Further we insert the two constraints “\( t_{p} \) equals start of \( s_{n1,2} \)” and “\( t_{p} \) is some time between start and end of \( s_{n2,3} \)” into the formulas of \( s_{n1,2} \) and \( s_{n2,3} \). So we derive:

\[
\exists t_{p} \exists t_{1}(b < t_{1} < e \land \Box_{[b, t_{1}]} s_{n1,1} \land t_{p} = t_{1} \land t_{1} < e \land \Box_{[t_{1}, e]} s_{n1,2}) \land \\
\exists t_{1}(b < t_{1} < e \land \Box_{[b, t_{1}]} s_{n2,1} \land \exists t_{2}(t_{1} < t_{2} < e \land \Box_{[t_{1}, t_{2}]} s_{n2,2} \land (t_{2} - t_{1}) < 2s \land t_{2} < t_{p} < e \land t_{2} < e \land \Box_{[t_{2}, e]} s_{n2,3})).
\]

\[\diamond\]

### 8.4 Object Identities

A symbol declared as global via the bulletin board (and similarly a symbol with identity label) represents the same object each time it appears along the SC. We may also declare modifications of the symbol that still refer to the same object. That way we can say that \( \square \) and \( \underline{\square} \) refer to the same car object, but the latter symbol additionally expresses a constraint on the state of the car. The example of Figure 73 is reproduced in Figure 86 for convenience.

![Bulletin board](image)

**Figure 86:** An SC and its b-board

Identifier labels and symbols at the b-board are translated along the same lines. Hence, we only discuss the b-board here.

Basically, a symbol at the bulletin board corresponds to a variable that is quantified globally, that is the quantifier’s scope spans the complete SC. Whether the b-board variable is quantified existential or universal will be specified at the TSC header (cf. Section 9.5).

Since our translation algorithm derives the formula for the SC compositionally from the snapshot node formulas, the algorithm has to distinguish between the b-board symbols and the local symbols. Local symbols will be quantified within the snapshot node, that is, their scope is limited to the snapshot node.
In the following we explain the technicalities on how the translation algorithm takes care of b-board symbols. Therefore let $\mathcal{S}_B$ be the set of symbols at the b-board and let each of them have a unique identifier $s.id$.

The unique identifier of $s$ ensures that all symbol occurrences refers to the same object variable $o_{s.id}$. If $s$ (modified or non-modified) occurs in a snapshot $sn$ (at some frame $f$), then the symbol occurrence $s \in S_f$ always refers to $o_{s.id}$, since $s$ has a unique identifier, i.e., $s.id = s.id$.

Further, the snapshot translation algorithm [Algorithm 2] does not introduce a local quantifier for an object variable in $\mathcal{S}_B$. Therefore the algorithm gets the set of symbols $\mathcal{S}_B$ as input.

The variables corresponding to $\mathcal{S}_B$ are bound by existential/universal quantifiers according to the header information (cf. Section 9.5) with a scope that cover the complete SC.

### 8.5 Position Pins

Position pins allow us to label a position with a name—globally for a TSC. Analogously to object identities, labelling a position means that the position is globally bound and keeps this binding for the whole formula.

Let $\mathcal{S}_P$ be the set of position pin symbols together with their labels. We assume a unique identifier $p.id$ for each $p \in \mathcal{S}_P$ and we denote the label name as $p.label$. Just like for object ids, we globally bind a variable via quantification for each (labelled) position pin occurrence. So, if $p = \mathcal{P}_p$ occurs in a snapshot $sn$ at a frame $f$, then the symbol occurrence gets the id $p.id$. That way [Algorithm 2] generates snapshot formulas that all refer to the same object variable, $o_{s.id}$. Further, the translation of snapshots via [Algorithm 2] gets as input $\mathcal{S}_B$ (fixed object identities) that also include $\mathcal{S}_P$ (positions pins), so that their variables do not get quantified locally within a snapshot formula.

### 8.6 Translation Scheme

In this section we sketch how the temporal logic formula is derived for a given snapshot chart, that is, how the translation steps described in the previous sections are applied in combination.

Let the snapshot chart SC be given. The translation scheme first determines the kind of SC.

#### 8.6.1 Determining the Time Annotated Snapshots Graphs

For the translation, we decompose the SC into time annotated snapshot graphs. We therefore consider the following three cases:

1. SC is not a premise consequence chart.

2. SC is a premise consequence chart.
   - In this case we decompose the SC into three subcharts:
     a) Let $SC_{past}$ be the past part of the SC, that is the (time-annotated) snapshot graph within the dashed hexagon left of the dashed vertical line.
     b) Let $SC_{fut.}$ be the future part, that is the (time-annotated) snapshot graph within the dashed hexagon right of the dashed vertical line.
     c) Let $SC_{conseq}$ be the consequence part, that is the (time-annotated) snapshot graph right of the dashed hexagon. Here we do not distinguish between the snapshot
Algorithm 3 inductively translates an annotated snapshot graph $SG_{id}$ into a formula $\phi_{id}$ by composing node formulas following the graph’s structure. A snapshot graph of just one node, $SG_\nu$, gets at line 15 of Algorithm 3 translated to a node formula of the form $\phi_\nu = \Box_{b_\nu,e_\nu}\phi_{sn}$, which in turn uses Algorithm 2 to translate $\phi_{sn}$. $\psi$ encodes the annotated timing constraints as explained at Section 8.3. The function $getAlias$ stores the pair of time label and corresponding variable name uniquely for an $SG$. $getAlias$ returns the stored variable name for the time label.

$\phi_\nu$ expresses that the constraints encoded by $sn_\nu$ invariantly holds from time $b_\nu$ up to $e_\nu$ and captures timing constraints. These node formulas are composed so that they contiguously hold along a graph’s path. This is realized by substituting the end time variable $e_1$ and the start time variable $b_2$ of concatenated subgraphs $SG_1;SG_2$ to the same time variable (cf. Section 8.1.3). Therefore, we assume that each (sub)graph has a unique id. Algorithm 3 constructs a formula $\phi_{id} = translateSG(SG_{id})$ where the start and end time variables are uniquely referable as $b_{id}$ and $e_{id}$. More precisely, Algorithm 3 builds a list of variable substitutions $\xi_{id}$ along with the formula $\phi_{id}$, that rename the start and end time variables appropriately.

Note that we chose to first translate a sequence $SG_{id_0}$ from left to right i.e. we chose to decompose a sequence $SG_{id_0}$ into $SG_{id_1};SG_{id_2}$ where $SG_{id_1}$ is not a sequence while $SG_{id_2}$ may be. This order is convenient, when timers have to be introduced. The order of translation for choice and concurrency at lines 5 and 8 has not been fixed. So, e.g., at a snapshot graph $SG_{id_0} = SG_{id_0}||SG_{id_1}||SG_{id_2}$ or $SG_{id_1}$ may be arbitrary snapshot graphs. If we, for instance, translate a choice of 4 alternatives $SG_{id_0} = SG_{id_0}||SG_{id_1}||SG_{id_2}||SG_{id_3}$ for instance as $SG_{id_0} = SG_{id_0};SG_{id_1};SG_{id_2}$ and $SG_{id_1} = SG_{id_1}||SG_{id_2}$ and $SG_{id_2} = SG_{id_2}||SG_{id_3}$. To the translation algorithm may nondeterministically choose the decomposition of sequences and choice and concurrency, but the decomposition of a chart sequence is constrained when timers have to be introduced. We hence chose to always fix the decomposition order of sequences from left to right, for simplicity.

Figure 87 illustrates the translation process on an exemplary snapshot graph $SG$, which is shown in Figure 87(a). We assume that snapshots $sn_1$ have already been translated to formulas $\phi_1$ by Algorithm 2.

Figure 87(b): The graph is considered as sequence of subgraphs $A,B$, which are translated to $\phi_A, \phi_B$. $\phi_B$ gets $\exists_{SG} \in Time : b_{SG} < t_{SG} < e_{SG} \wedge \xi_B(\phi_B)$ with $\xi_B = [b_1 \setminus b_{SG}, e_1 \setminus e_{SG}]$. $\phi_A$ gets $\exists_{SG} \in Time : b_{SG} < t_{SG} < e_{SG} \wedge \xi_A(\phi_A) \wedge \xi_B(\phi_B)$ with $\xi_A = [b_2 \setminus b_{SG}, e_2 \setminus e_{SG}]$. $\phi_A$ is the choice of snapshots $sn_1$ and $sn_2$, hence $\phi_A = \xi_1(\Box_{b_1,e_1}\phi_1) \lor \xi_2(\Box_{b_2,e_2}\phi_2)$ with $\xi_1 = [b_1 \setminus b_A, e_1 \setminus e_A], \xi_2 = [b_2 \setminus b_A, e_2 \setminus e_A]$. $B$ is the concurrency of $sn_3$ with subgraph $C$. So Algorithm 3 sets $\phi_B = \xi_3(\Box_{b_3,e_3}\phi_3) \land \xi_C(\phi_C)$ with $\xi_3 = [b_3 \setminus b_B, e_3 \setminus e_B], \xi_C = [b_C \setminus b_B, e_C \setminus e_B]$. $\phi_C$ gets $\exists_{SG} \in Time : b_{SG} < t_{SG} < e_{SG} \wedge \xi_3(\Box_{b_3,e_3}\phi_3) \land \xi_4(\Box_{b_4,e_4}\phi_4)$ with $\xi_4 = [b_4 \setminus b_C, e_4 \setminus e_C], \xi_4 = [b_4 \setminus b_C, e_4 \setminus e_C]$. To sum up, according to Algorithm 3 $\phi_{SG}$ translates to $\exists_{SG} : b_{SG} < t_{SG} < e_{SG} \wedge (\Box_{b_{SG},e_{SG}}\xi(\phi_1) \lor \Box_{b_{SG},e_{SG}}\xi(\phi_2) \lor \Box_{b_{SG},e_{SG}}\xi(\phi_3) \lor \Box_{b_{SG},e_{SG}}\xi(\phi_4)) \land \text{...}$. 

As the next step, we translate the snapshot graphs.

8.6.2 Translating Snapshot Graphs

The graph that constitutes the consequence and the concurrent future snapshot graph that may optionally reappear. In our context here we refer to the time annotated snapshot graph right of the dashed hexagon as $SC_{\text{conseq}}$.
Algorithm 3: Translation of an Annotated Snapshot Graph

Function translateSG

input : Annotated snapshot graph $SG_{id_0}$ with id $id_0$
output: temporal first-order formula

1) if $SG_{id_0} = SG_{id_1} \land SG_{id_2}$ and $SG_{id_1}$ is not seq. comp. then $/SG_{id_0}$ describes sequence

2) Let $t_{id_0}$ be an unused time variable name.

3) Let $\xi_1 = [b_{id_0} \land t_{id_0}, e_{id_0} \lor t_{id_0} \land e_{id_0}]$ and $\xi_2 = [b_{id_0} \lor t_{id_0}, e_{id_0} \land e_{id_0}]$.

4) return $\exists t_{id_0} : (\text{translate}SG(SG_{id_0})) \land (\text{translate}SG(SG_{id_1}))$.

5) if $SG_{id_0} = SG_{id_1} || SG_{id_2}$ then $/SG_{id_0}$ describes choice

6) Let $\xi_1 = [b_{id_0} \land b_{id_1}, e_{id_0} \lor e_{id_1}]$ and $\xi_2 = [b_{id_0} \lor b_{id_1}, e_{id_0} \land e_{id_1}]$.

7) return $\xi_1(\text{translate}SG(SG_{id_1})) \lor \xi_2(\text{translate}SG(SG_{id_2}))$.

8) if $SG_{id_0} = SG_{id_1} \& SG_{id_2}$ then $/SG_{id_0}$ describes concurrency

9) Let $\xi_1 = [b_{id_0} \land b_{id_1}, e_{id_0} \lor e_{id_1}]$ and $\xi_2 = [b_{id_0} \lor b_{id_1}, e_{id_0} \land e_{id_1}]$.

10) return $\xi_1(\text{translate}SG(SG_{id_1})) \land \xi_2(\text{translate}SG(SG_{id_2}))$.

11) if $SG_{id_0} = \neg SG_{id_1}$ then $/SG_{id_0}$ describes negation

12) Let $\xi_1 = [b_{id_0} \land b_{id_0}, e_{id_0} \lor e_{id_0}]$.

13) return $\xi_1(b_{id_0} \land \neg \text{translate}SG(SG_{id_1}))$.

14) return translateAnnotatedSN($SG_{id_0}$); $/SG_{id_0}$ is a (time annotated) snapshot node

8.6.3 Translating Snapshot Charts

Given $SC$ is a premise-consequence chart and we have already translated (according to Section 8.6.2):

- $SG_{past}$ to $\varphi_{past}$
- $SG_{fut}$ to $\varphi_{fut}$
- $SG_{conseq}$ to $\varphi_{conseq}$.

Via $\text{translate}SC$ we compose these subformulas to $\varphi_{SC}$ (following Section 8.2): 

$$\varphi_{SC} = \varphi_{past} \land \varphi_{fut} \Rightarrow \varphi_{conseq}$$ (translateSC)

$$\xi = [b_{past} \land SC \cup e_{past} \cup SC \cup e_{fut} \cup SC \cup e_{conseq} \cup SC \cup e_{conseq} \cup SC)]$$.

If $SC$ is not a premise-consequence chart $\text{translate}SC$ determines the formula $\varphi_{SC}$ directly from the translation according to Section 8.2.
Algorithm 4: Translation of a time annotated snapshot

1. **Function** `translateAnnotatedSN` 
   ```
   input: Snapshot node `sn_id` possibly with time annotation 
   output: temporal first-order formula \( \varphi_{id} \) with start and end variables \( b_{id} \) and \( e_{id} \)
   ```

2. Let \( b_{id}, e_{id} \) be a new variable names;
3. \( \varphi_{id} := \Box (b,e) \) `translateSnapshot(sn, getLabeledSymbols());`
4. **if** `sn_id` is labeled with start hour glass or time pin \( \tau \) **then**
   ```
   saveAlias(r, b_{id});
   ```
5. **if** `sn_id` is labeled with end hour glass and expression \( \psi \) **then**
   ```
   \varphi_{id} := \varphi_{id} \land \psi[r \setminus e_{id} - \text{getAlias}(r)];
   ```
6. **if** a sync. bar at `sn_id` is labeled with time pin \( \tau \) **then**
   ```
   \varphi_{id} := \varphi_{id} \land s < \text{getAlias}(r) < e;
   ```
7. **if** a sync. line at `sn_id`'s start is labeled with time pin \( \tau \) **then**
   ```
   \varphi_{id} := \varphi_{id} \land b = \text{getAlias}(r);
   ```
8. **return** \( \varphi \);

9. **TSCs and their Header**

Up to now we have introduced the syntax and semantics of snapshot charts. In this section we present the semantics of a TSC. Therefore we first introduce the syntax of TSC headers, to then give the semantics of TSCs (with headers). Note that in this document we only consider TSCs that consist of a header and an SC.

A header of a TSC holds the following elements

1. a unique **identifier** of the TSC,
2. the **activation mode**
   - which is a value in \{initial, invariant\},
3. the **path quantification** with values in \{all, exists\}
4. the **object quantification** with values in \{all, exists\}
5. the **time quantification** with values in \{all, exists\}.

**quantification = v** abbreviates the value combination "path quantification = v" and "object quantification = v" and "time quantification = v" for \( v \in \{exists, all\} \).

For the following we assume that we already translated the SC into a temporal logic formula \( \varphi_{SC} \). We now describe how this formula is extended to reflect the header information. Note, that the TSC identifier is for reference only and does not lead to an extension of \( \varphi_{SC} \).

9.1 Header Examples

In this section we will present exemplary TSCs and their translation into a temporal logic formula. We have already seen TSCs in Section 3. There we used a short hand notation at their headers to declare the TSCs of Figure 2 and Figure 3 as existential and the TSCs of Figure 4 as universal. In Figure 88 we present the TSC of Figure 2 but with detailed header annotation. The intuitive meaning of the TSC of Figure 88 has already been discussed at Section 3. Here we give the temporal logics formula that formally captures the semantics of the TSC. The translation process presented in this section yields the following formula:
Figure 88: Car collides with obstacle.

Figure 89: Rule: Change lane to avoid collision, if next lane is free.

\[ \exists t_e > 0 \in Time \exists o \in Obstacle \exists ego \in AV : \exists t_1 \in Time: 0 < t_1 < t_e \land \square_{[0,t_1]} \text{True} \land (\exists t_2 \in Time: t_1 < t_2 < t_e \land (\square_{[t_1,t_2]} \text{close_to_obstacle} \land \square_{[t_2,t_e]} \text{contact_with_obstacle})\]

It formally expresses, that there has to be at least one system run/evolution at which there exists an end time \( t_e \), an obstacle and an ego car (of class autonomous vehicle in our example), such that the scenario predicates hold contiguously. The symbol dictionary defines the domain of variables ego and obstacle. We abbreviate the predicates that result from translating the snapshots by close_to_obstacle and contact_with_obstacle. We refrain from unfolding the abbreviations close_to_obstacle, contact_with_obstacle, as their intuitive meaning has been sufficiently been discussed already at Section 3.

Such a TSC is well suited to express requirements on the world model (or a test suite) in terms of scenarios that have to be possible within the world model (or executed within tests).

Next, let us consider the universal TSC in Figure 89, a premise-consequence rule using the "past and future imply a future consequence" pattern. We also have seen this TSC in Section 3 but with short hand notation at its header.

The temporal logics formula resulting from translating the TSC in Figure 89 is:

\[ \forall t_e > 0 \in Time \forall o \in Obstacle \forall ego \in AV : \Box ((\Box_{[0,t_0]} \text{close_to_obstacle} \land \exists t_0 \in Time (t_{SG} < t_0 < t_e \land \Box_{[t_0,t_{SG}]} \text{left_will_be_free} \land t_0 - t_{SG} > t \land \Box_{[0,t_e]} \text{True})) \rightarrow \exists t_1 \in Time : t_{SG} < t_1 < t_e \land \Box_{[t_{SG},t_1]} \text{change_lane} \land \exists t_2 \in Time : t_1 < t_2 < t_e \land \Box_{[t_1,t_2]} \text{drive_past_obstacle} \land t_2 - t_{SG} < t \land \exists t_3 \in Time : t_2 < t_3 < t_e \land \Box_{[t_2,t_3]} \text{be_past_obstacle} \land \Box_{[t_1,t_3]} \text{True})\]

It formally expresses, that at any system run/evolution and for all ego cars and obstacles, if an ego car gets close to an obstacle and the left lane will be free, then the ego car has to circumvent the obstacle. Since the activation mode is always, any match of the premise along a system run implies that the consequence has to hold. We have to consider any choice for the switching time \( t_{SG} \) and \( t_e \), since the time quantification is all. So if the premise matches for a pair \( t_{SG} \) and \( t_e \), then the consequence has to hold.

The TSC of Figure 89 expresses an implication that has to hold always. It is well suited to express requirements or implementation rules of the system under design (Circumvent the obstacle, if left lane is free.). Such an implication may also be used to express more generally requirements like complete coverage by a set of scenarios, as sketched in Figure 90. Until now we only used existential (quantification = exists) and universal (quantification = all) TSCs, but for generality we allow more fine grained patterns.
Figure 90: Fully covered left-turn scenario. It always holds "If the car wants to do a left turn, it either turns left or it does not."

9.2 Activation Mode

The activation mode is either initial or always. A TSC with activation mode = initial has to be satisfied only right at the start of a system run, whereas a TSC with activation mode = always has to be satisfied at all times (anew).

So, with activation mode = initial, \( \varphi_{SG} \) is specified to hold at the very start of the system run, that is the start time variable is set to 0: \( \varphi_{SG}[b_{SG}\,0] \). For the activation mode always, \( \Box \), the temporal operator globally, is prepended, i.e., \( \Box \varphi_{SG}[b_{SG}\,0] \) is derived. So at all times, the \( \varphi_{SG} \) has to hold anew.

9.3 Path Quantification Mode

The path quantification mode is either

1. all
   specifying that all trajectories of the system have to satisfy the TSC. Note that it is also okay, that no trajectory satisfies the TSC, if the system has no trajectories,

2. exists
   specifying that there is at least one trajectory of the system, that satisfies the TSC, or

3. exists, all
   which specifies that all trajectories have to satisfy the TSC specification and there has to be at least one trajectory that satisfies the specification.

The path quantification = all is translated to the path quantifier \( \forall \), while path quantifier = exists is translated to path quantifier \( \exists \).

Having a TSC\(_{EA}\) with path quantification = exists, all is a short hand for for having two TSCs – let us say TSC\(_{E}\) and TSC\(_{A}\) – that are just like TSC\(_{EA}\) but TSC\(_{E}\) has path quantification exists and TSC\(_{A}\) has all. So we ignore the value combination exists, all in the following.

9.4 Time Quantification Mode

The time quantification mode specifies the quantifier of the free end time variable \( e_{SG} \) of \( \varphi_{SG} \) and if SG is a premise-consequence chart (cf. Section 8.6.2) also the quantifier of the free switching time variable \( t_{SG} \). The time quantification mode is either all or exists.

If the time quantification mode is all (time quantification mode = all), we prepend

- \( \forall t_{SG} \in \text{Time} : \forall e_{SG} \in \text{Time} : \) if SG is a premise-consequence chart
Traffic Sequence Charts – From Visualization to Semantics

• $\forall e_{SG} \in \text{Time} :$ if $SG$ is not a premise-consequence chart.

to $\varphi_{SG}$. Likewise, if time quantification mode = exists, we prepend

• $\exists e_{SG} \in \text{Time} : \exists e_{SG} \in \text{Time} :$ if $SG$ is a premise-consequence chart

• $\exists e_{SG} \in \text{Time} :$ if $SG$ is a not a premise-consequence chart

to $\varphi_{SG}$. 

9.5 Object Quantification Mode

The object quantification mode specifies the quantifier of

(i) objects at the b-board and of

(ii) positions fixed via a position pin.

The object quantification mode is either

1 all

specifying that all objects of the given type have to satisfy the SC\[17\].

2 exists

specifying that there is at least one object of the given type, that satisfies the TSC.

Let $\delta_B = \{ \delta_1, \ldots, \delta_n \}$ be the set of object symbols of a TSC with fixed identity and $\delta_P = \{ p_1, \ldots, p_m \}$ is the set of labeled position pins. Let symbol $\delta_i$ have the identity $id_i$ and $T_i = \text{type}_{\text{id}} (\delta_i)$, and $p_j$ have identity $id'_j$.

If the object quantification mode is all (object quantification mode = all), we prepend

$$\forall o_{id_1} \in T_1, \ldots , o_{id_n} \in T_n, p.x_{id'_1}, p.y_{id'_1}, \ldots , p.x_{id'_m}, p.y_{id'_m} \in \mathbb{R}$$

to $\varphi_{SG}$. Likewise, if object quantification mode = exists, we prepend

$$\exists o_{id_1} \in T_1, \ldots , o_{id_n} \in T_n, p.x_{id'_1}, p.y_{id'_1}, \ldots , p.x_{id'_m}, p.y_{id'_m} \in \mathbb{R}$$

to $\varphi_{SG}$.

In the current version the object quantification mode is determined for all "objects". A more fine grained specification of quantifiers can be easily introduced. For example the quantifier could be specified via b-boards within the SG (not only proceeding it). We refrain from formally introducing a more fine grained specification within this technical report, since we did not yet need such features in the case studies we considered so far. Since, at the current version, by definition all the quantifiers are the same, the order in which we quantify objects with fixed id and pinned positions does not matter.

9.6 Translation Scheme for a TSC

Given a TSC that consists of a header and an SG. We build $\varphi_{SG}$ according to Section 8. For the activation condition we modify $\varphi_{SG}$ according to Section 9.2, $\varphi_{SG} := \varphi_{SG} \circ [b_{SG} \emptyset]$ for activation mode initial, and $\varphi_{SG} := \Box \varphi_{SG} \circ [b_{SG} \emptyset]$ for activation mode always. Then we construct a prefix for $\varphi_{SG}$ according to Section 9.3 and Section 9.5.

\[17\] and it is okay if no object satisfies the TSC, if the system has no object of the given type
Algorithm 5: Translation of a Traffic Sequence Chart

1 Function translateTSC
   input : TSC
   output: temporal first-order formula
2 Decompose TSC into header $H$ and body $SC$;
3 $\varphi_{SC} := \text{translateSC}(SC)[b_{SC} \setminus 0]$; //translate body SC
4 if activation mode is always then //activation mode
5 $\varphi_{SC} := \Box \varphi_{SC}$;
6 Let $\text{pref}$ be the quantification of b-board objects, position pins according to Section 9.5.
7 $\varphi_{SC} := \text{pref} \varphi_{SC}$; //object quantification
8 Let $\text{pref}$ be the time quantification according to Section 9.4.
9 $\varphi_{SC} := \text{pref} \varphi_{SC}$; //time quantification
10 Let $\text{pref}$ be the path quantifier according to Section 9.3.
11 $\varphi_{SC} := \text{pref} \varphi_{SC}$; //path quantification
12 return $\varphi_{TSC} := \varphi_{SC}$.

10 Specification and Implementation

In this section we define what we consider as a TSC specification and we define when an implementing system satisfies such a TSC specification.

A TSC specification $\text{Spec}$ consists of
1 a world model $WM$
2 a symbol dictionary $\text{sdict}$, and
3 a set $\text{TSC}$ of TSCs.

At a TSC specification the conjunction of all TSCs has to hold, $\wedge_{TSC \in \text{TSC}} \varphi_{TSC}$, where $\varphi_{TSC}$ is the formula of logic $\mathcal{L}$ derived for the $TSC$ according to Algorithm 5.

For now we consider specifications where the behavior of an autonomous car, called $ego$, is specified. The world $WM$—given as the parallel composition of hybrid automaton instances $H_i$ of finitely many classes $H_C$—has a distinct class $H_{ego}$ whose instances represent $ego$ cars. At the specification $H_{ego}$ reflects the physical laws of a mechanical car but leaves the controller open. So $H_{ego}$ neither controls safety distances nor obeys traffic rules.

An implementing system $I$ specifies a controller for $ego$. So $I$ equals $WM$ except for $H_{ego}$ which is replaced by an automaton $H^I_{ego}$ whose controller is implemented. The controller has the task to ensure that $ego$ obeys the TSC specification. We say $I$ implements a TSC specification $\text{Spec}$, iff $I \models \wedge_{TSC \in \text{TSC}} \varphi_{TSC}$.

11 Related Work

Live Sequence Charts (LSCs) are a visual specification language for the description of system traces and communication [1]. The key elements are instances and messages send between them. Indeed, TSCs have been developed with LSCs in mind. We plan to integrate LSCs into TSCs in order to specify communications, such that TSCs extend the LSC formalism by providing a visualization of the continuous evolution via sequences of snapshot invariants.
Multi-Lane Spatial Logic is a spatial interval logic based on the view of each car. It was introduced in [8] to simplify reasoning about safety of road traffic by abstracting from the car dynamics. MLSL has been shown to be undecidable in various relaxations and restrictions, nevertheless sound proof systems for reasoning about safety of traffic situations have been presented [12,9,14]. In [6] it has been shown that it is decidable whether truth of an arbitrary MLSL formula can be safely determined on a given sample size under a reasonable model of technical observation of the traffic situation. TSCs -in contrast to MLSL- do not determine the level of abstraction, but leaves it open to the specification of the world model. TSCs allow to specify timing and spatial properties. The efficient analysis of TSC specifications is future work though. In particular, it seems promising to develop approaches that exploit the results of MLSL for TSCs.

OpenSCENARIO [17] together with OpenDRIVE [15] and OpenCRG [16] forms a set of exchange formats tailored for describing traffic scenarios. These formats aim at becoming a standard and serve as a means to imperatively describe the behavior of the environment and optionally of the ego vehicle. The static road networks are basically described as graphs of lanes labeled with geometric shapes. Dynamic content is described as a storyboard with trigger-action pairs. OpenSCENARIO corresponds to existential TSCs (quantification mode = exists) and, hence, are not suitable for the specification phase. Even the specification of existential scenarios is limited in comparison to TSCs. At the current state—to the best of the authors knowledge—OpenSCENARIO lacks elements to (1) distinguish between possible (existential) and expected (universal) behavior (2) distinguish between past and future behavior, and consequences, (3) express “don’t cares”, “somewhere”, “nowhere”, (4) explicitly express alternative and concurrent behavior.

Realizability of cooperative driving tasks has been addressed in [4]. In particular a formal approach for the verification of time probabilistic requirements (such as collision freedom) of given maneuvering and communication capabilities of the car based on formal specification has been developed. Future work will investigate algorithms for deciding consistency and realizability of TSC specifications under robustness assumptions. While we do not believe, that formally synthesized controllers for the ego vehicle will be used for implementation purposes, they can be used in the concept analysis phase of autonomous vehicles expected to master additional traffic scenarios.

(Probabilistic) Timed Property Sequence Charts (PTPSCs) extend Property Sequence Charts. Both are a scenario-based notation to represent temporal properties of concurrent systems aiming to balance expressive power and simplicity of use [18]. PTPSCs provide pre-charts, borrowed from LSCs, clock resets and clock constraints. Additionally, (sets of) messages can be assigned probabilities which impact the reception of the messages. PTPSCs can be automatically translated into timed-Büchi automata and, hence, are suitable for runtime verification and hypothesis testing can be used to also check the probabilistic part [19] by inspecting several sample runs. However, PTPSCs provide no means to graphically describe spatial constraints, branching, or concurrency, so that the planned addition of probabilities to TSCs is a worthwhile contribution.

The Visual Logic (VL) by Kemper and Etzien was developed to specify sequences of traffic situations on the highway aiming to bridge the gap between engineers, psychologists,
and scientists [11]. VL like TSCs aim to provide an intuitive visualization formalism with a formal semantics. VL combines LSCs and a visual formalism via which spatial relations of traffic scenarios are captured. Kemper and Etzien define the semantics of VL via a translation into timed automata. Conceptually, TSCs can be considered an extension of VL: TSCs share the motivation and borrow from the visual formalism of VL, we plan to integrate LSCs as well. While VL allows sequences of snapshots only, TSCs allow SC with a complex graph structure. Our visual formalism generalizes distance arrows of VL to predicate arrows. The concept of nowhere- and somewhere-boxes is new, as well as the possibility to define anchors. Further, TSCs allow time annotations and identity labels and the "past and presence implies a future consequence" pattern is new.

References


Index

A
Activation Mode, 68
alive, 8, 27, 37, 55
Anchor, 5, 12, 25, 29
   Line Anchor, 29
   Point Anchor, 29
   pos, 12
   Standard Anchors, 13, 29

B
Bounding Box, 21, 25
Box Symbol, 27
Bulletin Board, 35, 41
   Scope, 35

C
Choice, see Snapshot Graph
Concatenation, see Snapshot Graph
Concurrency, see Snapshot Graph

D
Distance Lines, 13, 13, 21, 32
don’t care, 12, 17
Duration, 39, 41, 41, 50

E
Empty Snapshot, 16

F
Frame, 24

H
Header, 4
   Translation into $\mathcal{L}$, 69
Hour Glass, 39, 41, 50, 59
   Negation Scope, 49
Hybrid Automata, 8, 10

I
Identifier Label, 62
Identifier Lable, 53
Implementation, 70

L
Live Sequence Chart, 4, 4
Logic $\mathcal{L}$, 7, 25

M
Mode, see Header, Activation, Path
   Quantification, Object
   Quantification, Time
   Quantification

N
Negated Snapshot, 16
Negation, see Snapshot Graph, Snapshot
   Node
Nowhere-box, 12, 13, 18, 19, 24

O
Object
   Class, 8
   Model, 8
   Quantification, 66
   Symbol, 27, 34
   Visualization of Object State, 17
Object Identities, 39, 53
Object Position, 5
Object Quantification Mode, 66, 69
Omniscient Eye, 47

P
p.a.f.i.c., see Premise-Consequence
   Chart46
Path Quantification Mode, 66, 68
pic, see Premise-Consequence Chart45
Position Pin, 13, 21, 21, 63
   Scope, 13
   Symbol Occurrence, 27
Predicate Arrow, 24, 25, 32
Premise, 46
Premise-Consequence Chart, 40, 58
   past and future imply a future conseq., 45
   past implies a future conseq., 46

Q
Quantification Mode, 66

R
Relative Placement, 12, 18, 28, 37
   Translation Into $\mathcal{L}$, 32

S
Sequence, see Snapshot Graph
Signature, 7, 25
Snapshot
   Empty Snapshot, 12, 13
   False Snapshot, 15
   Snapshot Formula, 35
   Translation into \( \mathcal{L} \), 38
   True Snapshot, 13
Snapshot Chart, 4, 4, 39
   Translation into \( \mathcal{L} \), 65
Snapshot Graph, 39, 39, 40
   Choice, 39, 43
   Concatenation, see Sequence
   Concurrency, 39, 45, 57
   Negation, 39, 44
   Sequence, 39, 57
Snapshot Node, 39
Somewhere-box, 13, 14, 18, 19, 24
Spatial View, 4, 5, 11
State of an Object, 17
Symbol Dictionary, 4, 5, 25
Synchronization Bar, 50, 60

T
Time Constraints, 49, 59

see Time Pin, Hour Glass, 72
Time Pin, 39, 41, 50, 60
   Synchronization Bar, 50
Time Quantification, 66
   Mode, 68
Topology, see Relative Placement
Traffic Sequence Chart, 4, see TSC
translateSC, 65
translateTSC, 70
TSC, 4, 5, 66
   Header, 66
   Semantics, 66
   Translation into \( \mathcal{L} \), 69
TSC Header
   Activation Mode, 68
   Object Quantification Mode see, 69
   Path Quantification Mode, 68
   Time Quantification Mode, 68
TSC Specification, 4, 68, 70
   Satisfaction, 70

W
World Model, 5, 8, 8, 9, 15, 34
Symbols and Abbreviations

\( A_f \) occurrences of predicate arrows that connect a symbol occurrence \( s \in S_f \) to a symbol occurrence \( s' \) at a frame (transitively) containing \( f \)

B-board Bulletin Board

HIOA Hybrid I/O Automata

\( \mathcal{L} \) multi-sorted first-order real-time logic capturing the TSC semantic

\( L_f \) set of all distance lines that connect symbol occurrences in \( S_f \) with symbol occurrences at a frame transitively containing \( f \).

LSC Live Sequence Chart

n-box Nowhere-box

\( \ominus \) omniscient eye

p.a.f.i.c. past and future imply future consequence

p.i.c. past implies future consequence

pos point anchor of an object

SC Snapshot Chart

SG Snapshot Graph

\( \text{SG}_{\text{conseq}} \) Consequence of a past Snapshot Graph

\( \text{SG}_{\text{past}} \) Past Snapshot Graph

\( \Sigma \) Signature of logic \( \mathcal{L} \)

s-box Somewhere-box

s-dictionary Symbol Dictionary

\( S_B^g \) set of symbols with a global scope

\( S_f \) set of all symbol occurrences (i.e., object and box symbols) that belong to frame

\( S_f^{SB} \) set of s-box occurrences at frame \( f \)

\( S_f^{NB} \) set of n-box occurrences at frame \( f \)

\( S_f^B \) equals \( S_f^{SB} \cup S_f^{NB} \), i.e. set of box occurrences at frame \( f \)

\( S_f^{PP} \) set of position pin occurrences at frame \( f \)

\( S_f^g \) \( S_f \setminus S_f^B \), i.e., set of all symbol occurrences that belong to frame \( f \) without box symbol occurrences

TSC Traffic Sequence Chart
WM  World Model

\( \underbar{x} \)  standard anchor, minimal x position covered by an object

\( \overline{x} \)  standard anchor, maximal x position covered by an object

\( \underbar{y} \)  standard anchor, minimal y position covered by an object

\( \overline{y} \)  standard anchor, maximal y position covered by an object