Requirements Validation for Hybrid Systems*

Alessandro Cimatti, Marco Roveri, and Stefano Tonetta
Fondazione Bruno Kessler (FBK-irst), Trento, Italy

Abstract. The importance of requirements for the whole development flow calls for strong validation techniques based on formal methods. In the case of discrete systems, some approaches based on temporal logic satisfiability are gaining increasing momentum. However, in many real-world domains (e.g. railways signaling), the requirements constrain the temporal evolution of both discrete and continuous variables. These hybrid domains pose substantial problems: on one side, a continuous domain requires very expressive formal languages; on the other side, the resulting expressiveness results in highly intractable problems.
In this paper, we address the problem of requirements validation for real-world hybrid domains, and present two main contributions. First, we propose the HRELTL logic, that extends the Linear-time Temporal Logic with Regular Expressions (RELTL) with hybrid aspects. Second, we show that the satisfiability problem for the linear fragment can be reduced to an equi-satisfiable problem for RELTL. This makes it possible to use automatic (albeit incomplete) techniques based on Bounded Model Checking and on Satisfiability Modulo Theory.
The choice of the language is inspired by and validated within a project funded by the European Railway Agency, on the formalization and validation of the European Train Control System specifications. The activity showed that most of requirements can be formalized into HRELTL, and an experimental evaluation confirmed the practicality of the analyses.

1 Introduction

Requirements analysis is a fundamental step in the development process of software and system design. In fact, flaws and ambiguities in the requirements can lead to the development of correct systems that do not do what they were supposed to. This is often unacceptable, especially in safety-critical domains, and calls for strong tools for requirements validation based on formal techniques. The problem of requirements validation is significantly different from traditional formal verification, where a system model (the entity under analysis) is compared against a set of requirements (formalized as properties in a temporal logic), which are assumed to be "golden". In requirements validation, on the contrary, there is no system to be analyzed (yet), and the requirements themselves are the entity under analysis.

Formal methods for requirements validation are being devoted increasing interest [12, 15, 27, 1]. In such approaches, referred to as *property-based*, the requirements are represented as statements in some temporal logics. This allows to retain a correspondence between the informal requirements and the formal statement, and gives the

* The first and second authors are supported by the European Commission (FP7-2007-IST-1-217069 COCONUT). The third author is supported by the Provincia Autonoma di Trento (project ANACONDA).

ability to reason at the level of abstraction of the requirements engineer. Typical analysis functionalities include the ability to check whether the specification is consistent, whether it is strict enough to rule out some undesirable behaviors, and whether it is weak enough not to rule out some desirable scenarios. These analysis functionalities can in turn be obtained by reduction to temporal logics satisfiability [27].

Property-based approaches have been typically applied in digital domains, where the requirements are intended to specify a set of behaviors over discrete variables, and the wealth of results and tools in temporal logic and model checking provides a substantial technological basis. However, in many real-world domains (e.g. railways, space, industrial control), the requirements are intended to constrain the evolution over time of a combination of discrete and continuous variables. Hybrid domains pose substantial problems: on the one side, a continuous domain requires very expressive formal languages; on the other side, the high expressiveness leads to highly intractable problems.

In this paper, we address the problem of requirements validation in such hybrid domains by making two main contributions.

First, we define a suitable *logic for the representation of requirements in hybrid domains*. The logic, called Hybrid Linear Temporal Logic with Regular Expressions (HRELTL), is interpreted over hybrid traces. This allows to evaluate constraints on continuous evolutions as well as discrete changes. The basic atoms (predicates) of the logic include continuous variables and their derivatives over time, and are interpreted both over time points and over open time intervals.[1] The semantics relies on the fact that each open interval of a hybrid trace can be split if it does not have a uniform evaluation of predicates in the temporal formulas under analysis (cf. [26, 13]), and that the formulas satisfy properties of sample invariance and finite variability [13]. The logic encompasses regular expressions and linear-time operators, and suitable choices have been made to interpret the "next" operator and regular expressions over the open time intervals, and the derivative of continuous variables over the time points.

Second, we define a *translation method for automated verification*. The translation encodes satisfiability problems for the linear fragment of HRELTL into an equisatisfiable problem in a logic over discrete traces. The restrictions of the linear fragment guarantee that if the formula is satisfiable, there exists a piecewise-linear solution. We exploit the linearity of the predicates with regard to the continuous variables to encode the continuity of the function into quantifier-free constraints. We can therefore compile the resulting formula into a fair transition system and use it to solve the satisfiability and the model checking problem with an automata-theoretic approach. We apply infinite-state model checking techniques to verify the language emptiness of the resulting fair transition system.

Our work has been inspired by and applied within a project funded by the European Railway Agency (http://www.era.europa.eu). The aim of the project was to develop a methodology supported by a tool for the validation of requirements in railway domains. Within the project, we collaborated with domain experts in a team that tackled the formalization of substantial fragments of the European Train Control System (ETCS) specification. With regard to the hybrid aspects of ETCS requirements, the formaliza-

[1] This is known to be a nontrivial issue: see for instance [26, 13, 22] for a discussion on which type of intervals and hybrid traces must be considered.

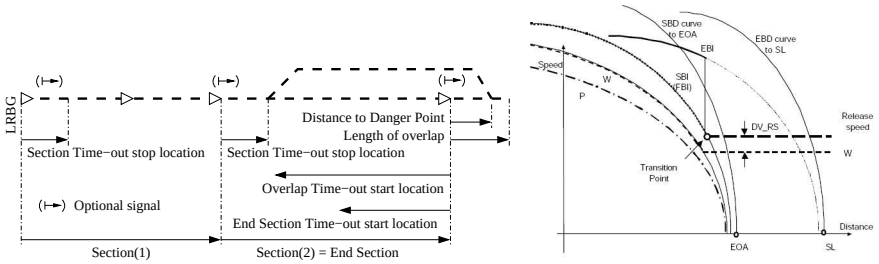


Fig. 1. Structure of an MA (left) and a speed monitoring curve (right) [14].

tion and the validation were based on the language and techniques described in this paper, and were successfully applied by the domain experts.

2 Motivating Application Domain

The ETCS specification is a set of requirements related to the automatic supervision of the location and speed performed by the train on-board system. The system is intended to be progressively installed on all European trains in order to guarantee the interoperability with the track-side system which are currently governed by national rules. ETCS specifies how the train should behave in the proximity of the target location. In particular, the Chapter 3 of the System Requirement Specification (SRS) [14] describes how trains move on a line and periodically receive a so-called Movement Authority (MA). The MA consists of a set of sections and a series of timeout that define some deadlines of the authorization to move in each section while a number of curves limit the speed of the train approaching the end of the MA (see Fig. 1). The specific curves are not defined by ETCS, but only constrained by high-level requirements (*"The algorithm for their calculation is an implementation matter"* SRS Sec. 3.13.4.1). Moreover, when the trains pass some limit, particular actions must be taken on board: e.g. when the train passes the end of the MA the "train trip" must be started.

The ETCS specification poses demanding requisites to the formal methods adopted for its validation. First, the adopted formalism shall be able to capture the meaning of the requirements. Second, the formalism shall be as simple as possible to be used by non-experts in formal methods: the requirements are usually ambiguous English sentences that only an expert in the domain can formalize and validate.

In this context, a model-based approach is not natural. First, designing a hybrid system that captures all behaviors allowed by the requirements requires to consider all possible intricate combinations of timeout values, locations where to reset the timers, speed limits for given locations. Second, these are not parameters of the system but variables that change their value at discrete steps. Finally, in a model-based approach it is hard to maintain the link between a requirement and its formal counterpart in the model.

A property-based approach to requirements validation relies on the availability of an expressive temporal logic, so that each informal statement has a formal counterpart with similar structure. A natural choice are formulas in Linear-time Temporal Logic (LTL) [25] extended with Regular Expressions (RELTL) [6], because temporal formulas often resemble their informal counterpart. This is of paramount importance when

experts in the application domain have the task of disambiguating and formalizing the requirements. For instance, a complex statement such as *"The train trip shall issue an emergency brake command, which shall not be revoked until the train has reached standstill and the driver has acknowledged the trip"* SRS Sec. 3.13.8.2 can be formalized into $\mathbf{G}\ (train\_trip \rightarrow (emergency\_brake\ \mathbf{U}\ (train\_speed = 0 \wedge ack\_trip)))$.

In order to deal with an application domain such as ETCS, first, we need to model the dynamic of continuous variables, such as position, speed, time elapse, and timers, in a way that is reasonably accurate from the physical point of view. For example, we expect that a train can not move forward and reach a location without passing over all intermediate positions. Second, we must be able to express properties of continuous variables over time intervals. This poses problems of the satisfiability of formulas like $(pos \leq P\ \mathbf{U}\ pos > P)$ or $(speed > 0\ \mathbf{U}\ speed = 0)$. The first formula is satisfiable only if we consider left-open intervals, while the second one is satisfiable only if we consider left-closed intervals (see [22]). Considering time points (closed singular intervals) and open intervals is enough fine grained to represent all kinds of intervals.

Last, we need to be able to intermix continuous evolution and discrete steps, intuitively modeling "instantaneous" changes in the status of modes and control procedures. For example, the requirement *"The End Section timer shall be started on-board when the train passes the End Section timer start location"* (SRS Sec. 3.8.4.1.1) demands to interrupt the continuous progress of the train for resetting a timer.

3 Hybrid traces

Let V be the finite disjoint union of the sets of variables V_D (with a discrete evolution) and V_C (with a continuous evolution) with values over the Reals.[2] A state s is an assignment to the variables of V ($s : V \rightarrow \mathbb{R}$). We write Σ for the set of states. Let $f : \mathbb{R} \rightarrow \Sigma$ be a function describing a continuous evolution. We define the projection of f over a variable v, written f^v, as $f^v(t) \doteq f(t)(v)$. We say that a function $f : \mathbb{R} \rightarrow \mathbb{R}$ is piecewise analytic iff there exists a sequence of adjacent intervals $J_0, J_1, ... \subseteq \mathbb{R}$ and a sequence of analytic functions $h_0, h_1, ...$ such that $\cup_i J_i = \mathbb{R}$, and for all $i \in \mathbb{N}$, $f(t) = h_i(t)$ for all $t \in J_i$. Note that, if f is piecewise analytic, the left and right derivatives exist in all points. We denote with $\dot{f}$ the derivative of a real function f, with $\dot{f}(t)_-$ and $\dot{f}(t)_+$ the left and the right derivatives respectively of f in t. Let I be an interval of $\mathbb{R}$ or $\mathbb{N}$; we denote with $le(I)$ and $ue(I)$ the lower and upper endpoints of I, respectively. We denote with $\mathbb{R}^+$ the set of non-negative real numbers.

Hybrid traces describe the evolution of variables in every point of time. Such evolution is allowed to have a countable number of discontinuous points corresponding to changes in the discrete part of the model. These points are usually called *discrete steps*, while we refer to the period of time between two discrete steps as *continuous evolution*.

Definition 1 (Hybrid Trace). *A hybrid trace over V is a sequence $\langle \overline{f}, \overline{I} \rangle \doteq \langle f_0, I_0 \rangle, \langle f_1, I_1 \rangle, \langle f_2, I_2 \rangle, \ldots$ such that, for all $i \in \mathbb{N}$,*

- *either I_i is an open interval ($I_i = (t, t')$ for some $t, t' \in \mathbb{R}^+$, $t < t'$) or is a singular interval ($I_i = [t, t]$ for some $t \in \mathbb{R}^+$);*

[2] In practice, we consider also Boolean and Integer variables with a discrete evolution, but we ignore them to simplify the presentation.

– *the intervals are adjacent, i.e.* $ue(I_i) = le(I_{i+1})$*;*
– *the intervals cover* $\mathbb{R}^+$*:* $\bigcup_{i\in\mathbb{N}} I_i = \mathbb{R}^+$ *(thus* $I_0 = [0, 0]$*);*
– $f_i : \mathbb{R} \to \Sigma$ *is a function such that, for all* $v \in V_C$*,* f_i^v *is continuous and piecewise analytic, and for all* $v \in V_D$*,* f_i^v *is constant;*
– *if* $I_i = (t, t')$ *then* $f_i(t) = f_{i-1}(t)$*,* $f_i(t') = f_{i+1}(t')$*.*

Typically, the f_i are required to be smooth. Since observable events may occur during a continuous evolution, we wish that a predicate over the continuous variable changes its truth value only a finite number of times in a bounded interval. For this reason, we require the analyticity of functions (see similar assumptions in [13]). At the same time, we weaken the condition of smoothness allowing discontinuity in the derivatives also during a continuous evolution. This allows to observe the value of functions and their derivatives without the need to break the continuous evolution with discrete steps not required by the specification.

Fig. 2 shows the evolution of two continuous variables (*speed* and *limit*) and a discrete variable (*warning*). The evolution presents two discrete steps and three continuous evolutions. The figure shows two possible traces, respectively with 10 and 14 intervals. In the second continuous evolution the function associated to *speed* is continuous but not derivable in all points.

Some predicate over the variables in V may evaluate to true only in particular points of a continuous evolution. Therefore, it is important to sample the evolution in particular time points. We say that a trace is a sampling refinement of another one if it has been obtained by splitting an open interval into two parts by adding a sampling point in the middle [13]. In Fig. 2, TRACE2 refines TRACE1 by exposing two more points.

Definition 2 (Partitioning Function [13]). *A partitioning function* μ *is a sequence* $\mu_0, \mu_1, \mu_2, \ldots$ *of non-empty, adjacent and disjoint intervals of* $\mathbb{N}$ *partitioning* $\mathbb{N}$*. Formally,* $\bigcup_{i\in\mathbb{N}} \mu_i = \mathbb{N}$ *and* $ue(\mu_i) = le(\mu_{i+1}) - 1$*.*

Definition 3 (Trace Sampling Refinement [13]). *A hybrid trace* $\langle \overline{f}', \overline{I}' \rangle$ *is a sampling refinement of* $\langle \overline{f}, \overline{I} \rangle$ *by the partitioning* μ *(denoted with* $\langle \overline{f}', \overline{I}' \rangle \preceq^{\mu} \langle \overline{f}, \overline{I} \rangle$*) iff, for all* $i \in \mathbb{N}$*,* $I_i = \bigcup_{j\in\mu_i} I'_j$ *and, for all* $j \in \mu_j$*,* $f'_j = f_i$*.*

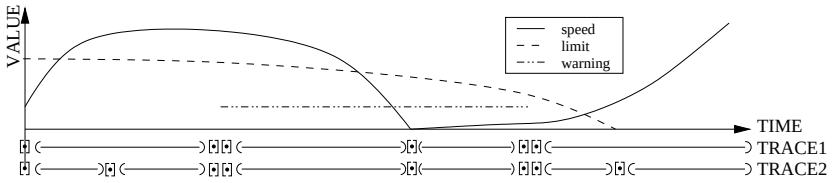


Fig. 2. Possible evolution of two continuous variables (*speed* and *limit*) and a discrete variable (*warning*), and two possible hybrid traces that represent it. TRACE2 is a refinement of TRACE1.

4 A temporal logic for hybrid traces

In this section we define HRELTL, i.e. linear temporal logic extended with regular expressions equipped to deal with hybrid traces. The language is presented in a general

form with real arithmetic predicates without details on the syntax and the semantics of the real functions. It is indeed possible that some requirements need such expressiveness to be faithfully represented. A linear sub-case is then presented for which we have a discretization that produces equi-satisfiable formulas.

Syntax. If v is a variable we denote with $\text{NEXT}(v)$ the value of v after a discrete step and with $\text{DER}(v)$ the derivative of v. If V is the set of variables, we denote with V_{next} the set of next variables and with V_{der} the set of derivatives. HRELTL is built over a set of basic atoms, that are real arithmetic predicates over $V \cup V_{next}$, or over $V \cup V_{der}$.[3] We denote with $PRED$ the set of predicates, with p a generic predicate, with p_{curr} a predicate over V only, with p_{next} a predicate over V and V_{next}, and with p_{der} a predicate over V and V_{der}. We denote with $\overline{p}$ the predicate obtained from p by replacing $<$ with $\geq$, $>$ with $\leq$, $=$ with $\neq$ and vice versa. We denote with $p_{\bowtie}$ the predicate obtained from p by substituting the top-level operator with $\bowtie$, for $\bowtie \in \{<, >, =, \leq, \geq, \neq\}$.

The subset $PRED_{la}$ of $PRED$ over *linear arithmetic* constraints consists of the predicates in one of the following forms

- $a_0 + a_1 v_1 + a_2 v_2 + \cdots + a_n v_n \bowtie 0$ where $v_1, \ldots, v_n \in V_C$, $a_0, \ldots, a_n$ are arithmetic predicates over variables in V_D, and $\bowtie \in \{<, >, =, \leq, \geq, \neq\}$.
- $a_0 + a_1 \dot{v} \bowtie 0$ where $v \in V_C$, a_0, a_1 are arithmetic predicates over variables in V_D, and $\bowtie \in \{<, >, =, \leq, \geq, \neq\}$.

Example 1. $x \leq y + z$, $\text{NEXT}(x) = 0$, $\text{DER}(x) \leq d$ are in $PRED$. The first two predicates are also in $PRED_{la}$, while the third one is in $PRED_{la}$ only if d is a discrete variable.

We remark that, the class of predicates generalizes the class of constraints used for linear hybrid automata [2, 19]; in particular, we replace constants with discrete (dense-domain) variables.

The HRELTL is defined by combining extended regular expressions (SEREs) and temporal operators from LTL. The *linear fragment* of HRELTL is defined by considering only predicates in $PRED_{la}$.

Definition 4 (SERE syntax). *If $p \in PRED$, r, r_1 and r_2 are SEREs, then:*

- *p is a SERE;*
- *ϵ is a SERE;*
- *r**[*]**, r_1 **;** r_2, r_1 **:** r_2, r_1 **|** r_2, and r_1 **&&** r_2 are SEREs.*

Definition 5 (HRELTL syntax). *If $p \in PRED$, ϕ, ϕ_1 and ϕ_2 are HRELTL formulas, and r is a SERE, then:*

- *p is a HRELTL formula;*
- *$\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\mathbf{X}\ \phi_1$, $\phi_1\ \mathbf{U}\ \phi_2$ are HRELTL formulas;*
- *$r \diamondsuit\!\!\rightarrow \phi$ is an HRELTL formula.*

We use standard abbreviations for $\vee$, $\rightarrow$, $\mathbf{G}$, $\mathbf{F}$, and $\mapsto$ (see, e.g., [11]).

Example 2. $\mathbf{G}\ (warning = 1 \rightarrow (warning = 1\ \mathbf{U}\ speed \leq limit))$ is in HRELTL.

[3] In practice, we consider also predicates with next variables and derivatives, and derivatives after a discrete step, but we ignore this extensions to simplify the presentation.

Semantics. Some choices underlie the definition of the semantics in order to guarantee that the satisfaction of a formula by a hybrid trace does not depend on the sampling of continuous evolutions, rather it depends only on the discrete steps and on the shape of the functions that describe the continuous evolutions (*sampling invariance* [13]). Other choices have been taken to make the formalization of requirements more natural. For example, predicates including next variables can be true only in discrete steps.

Definition 6 (PRED semantics).

- $\langle \overline{f}, \overline{I} \rangle, i \models p_{curr}$ *iff, for all* $t \in I_i$, p_{curr} *evaluates to true when* v *is equal to* $f_i^v(t)$, *denoted with* $f_i(t) \models p$;
- $\langle \overline{f}, \overline{I} \rangle, i \models p_{next}$ *iff there is a discrete step between* i *and* $i+1$, *i.e.* $I_i = I_{i+1} = [t, t]$, *and* p_{next} *evaluates to true when* v *is equal to* $f_i^v(t)$ *and* $\text{NEXT}(v)$ *to* $f_{i+1}^v(t)$, *denoted with* $f_i(t), f_{i+1}(t) \models p_{next}$;
- $\langle \overline{f}, \overline{I} \rangle, i \models p_{der}$ *iff, for all* $t \in I_i$, p_{der} *evaluates to true both when* v *is equal to* $f_i^v(t)$ *and* $\text{DER}(v)$ *to* $\dot{f}_i^v(t)_+$, *and when* v *is equal to* $f_i^v(t)$ *and* $\text{DER}(v)$ *to* $\dot{f}_i^v(t)_-$, *denoted with* $f_i(t), \dot{f}_i(t)_+ \models p_{der}$ *and* $f_i(t), \dot{f}_i(t)_- \models p_{der}$ *(when* $\dot{f}_i(t)$ *is defined this means that* $f_i(t), \dot{f}_i(t) \models p_{der}$*).*

Note that, for all $i \in \mathbb{N}$, f_i is defined on all reals, and thus the left and right derivatives are defined in all points of I_i.

In order to ensure sample invariance, the predicates inside a SERE can be true over a sequence of more than one moment. This is different from the standard discrete approach, where they are usually true only if evaluated on just one state. Moreover, we require that if a sequence satisfies the concatenation or repetition of two SEREs, the sequence must contain a discrete step.

Definition 7 (SERE semantics).

- $\langle \overline{f}, \overline{I} \rangle, i, j \models p$ *iff, for all* k, $i \leq k < j$, *there is no discrete step at* k $(I_k \neq I_{k+1})$, *and, for all* k, $i \leq k \leq j$, $\langle \overline{f}, \overline{I} \rangle, k \models p$;
- $\langle \overline{f}, \overline{I} \rangle, i, j \models \epsilon$ *iff* $i > j$;
- $\langle \overline{f}, \overline{I} \rangle, i, j \models r$**[*]** *iff* $i > j$, *or* $\langle \overline{f}, \overline{I} \rangle, i, j \models r$, *or there exists a discrete step at* k $(I_k = I_{k+1})$, $i \leq k < j$, *such that* $\langle \overline{f}, \overline{I} \rangle, i, k \models r$, $\langle \overline{f}, \overline{I} \rangle, k+1, j \models r$**[*]**;
- $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$ **;** r_2 *iff* $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$, $\langle \overline{f}, \overline{I} \rangle, j+1, j \models r_2$ *(i.e.,* r_2 *accepts the empty word), or;* $\langle \overline{f}, \overline{I} \rangle, i, i-1 \models r_1$, $\langle \overline{f}, \overline{I} \rangle, i, j \models r_2$ *(i.e.,* r_1 *accepts the empty word), or there exists a discrete step at* k $(I_k = I_{k+1})$, $i \leq k < j$, *such that* $\langle \overline{f}, \overline{I} \rangle, i, k \models r_1$, $\langle \overline{f}, \overline{I} \rangle, k+1, j \models r_2$;
- $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$ **:** r_2 *iff there exists a discrete step at* k $(I_k = I_{k+1})$, $i \leq k \leq j$, *such that* $\langle \overline{f}, \overline{I} \rangle, i, k \models r_1$, $\langle \overline{f}, \overline{I} \rangle, k, j \models r_2$;
- $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$ **|** r_2 *iff* $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$ *or* $\langle \overline{f}, \overline{I} \rangle, i, j \models r_2$;
- $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$ **&&** r_2 *iff* $\langle \overline{f}, \overline{I} \rangle, i, j \models r_1$ *and* $\langle \overline{f}, \overline{I} \rangle, i, j \models r_2$.

Definition 8 (HRELTL semantics).

- $\langle \overline{f}, \overline{I} \rangle, i \models p$ *iff* $\langle \overline{f}, \overline{I} \rangle, i \models p$;
- $\langle \overline{f}, \overline{I} \rangle, i \models \neg\phi$ *iff* $\langle \overline{f}, \overline{I} \rangle, i \not\models \phi$;
- $\langle \overline{f}, \overline{I} \rangle, i \models \phi \wedge \psi$ *iff* $\langle \overline{f}, \overline{I} \rangle, i \models \phi$ *and* $\langle \overline{f}, \overline{I} \rangle, i \models \psi$;
- $\langle \overline{f}, \overline{I} \rangle, i \models \mathbf{X}\ \phi$ *iff there is a discrete step at i* $(I_i = I_{i+1})$, *and* $\langle \overline{f}, \overline{I} \rangle, i+1 \models \phi$;

- $\langle \overline{f}, \overline{I} \rangle, i \models \phi \mathbf{U} \psi$ *iff, for some* $j \geq i$, $\langle \overline{f}, \overline{I} \rangle, j \models \psi$ *and, for all* $i \leq k < j$, $\langle \overline{f}, \overline{I} \rangle, k \models \phi$;
- $\langle \overline{f}, \overline{I} \rangle, i \models r \diamondsuit\!\!\rightarrow \phi$ *iff, there exists a discrete step at* $j \geq i$ ($I_j = I_{j+1}$) *such that* $\langle \overline{f}, \overline{I} \rangle, i, j \models r$, *and* $\langle \overline{f}, \overline{I} \rangle, j \models \phi$.

Definition 9 (Ground Hybrid Trace [13]). *A hybrid trace* $\langle \overline{f}, \overline{I} \rangle$ *is a ground hybrid trace for a predicate* p *iff the interpretation of* p *is constant throughout every open interval: if* $I_i = (t, t')$ *then either* $\langle \overline{f}, \overline{I} \rangle, i \models p$ *or* $\langle \overline{f}, \overline{I} \rangle, i \models \overline{p}$. *A hybrid trace* $\langle \overline{f}, \overline{I} \rangle$ *is a ground hybrid trace for a formula* ϕ *iff it is ground for all predicates of* ϕ.

Given an HRELTL formula ϕ, and a hybrid trace $\langle \overline{f}, \overline{I} \rangle$ ground for ϕ, we say that $\langle \overline{f}, \overline{I} \rangle \models \phi$ iff $\langle \overline{f}, \overline{I} \rangle, 0 \models \phi$.

Given an HRELTL formula ϕ, and any hybrid trace $\langle \overline{f}, \overline{I} \rangle$, we say that $\langle \overline{f}, \overline{I} \rangle \models \phi$ iff there exists a sampling refinement $\langle \overline{f}', \overline{I}' \rangle$ of $\langle \overline{f}, \overline{I} \rangle$ such that $\langle \overline{f}', \overline{I}' \rangle$ is ground for ϕ and $\langle \overline{f}', \overline{I}' \rangle \models \phi$.

For example, the hybrid traces depicted in Fig. 2 satisfy the formula of Example 2.

The following theorems guarantee that the semantics is well defined. (We refer the reader to Sec. B of the appendix for the proofs.)

Theorem 1 (Finite variability). *Given a formula* ϕ, *for every hybrid trace* $\langle \overline{f}, \overline{I} \rangle$ *there exists another hybrid trace* $\langle \overline{f}', \overline{I}' \rangle$ *which is a sampling refinement of* $\langle \overline{f}, \overline{I} \rangle$ *and ground for* ϕ.

Theorem 2 (Sample invariance). *If* $\langle \overline{f}', \overline{I}' \rangle$ *is a sampling refinement of* $\langle \overline{f}, \overline{I} \rangle$, *then the two hybrid traces satisfy the same formulas.*

Note that we can encode the reachability problem for linear hybrid automata into the satisfiability problem of a linear HRELTL formula. Despite the undecidability of the satisfiability problem, we provide automatic techniques to look for satisfying hybrid traces, by constructing an equi-satisfiable discrete problem.

5 Reduction to discrete semantics

RELTL is the temporal logic that combines LTL with regular expressions and constitutes the core of many specification languages. Here, we refer to a first-order version of RELTL with real arithmetic predicates. RELTL syntax can be seen as a subset of HRELTL where predicates are allowed to include only current and next variables, but not derivatives.

RELTL formulas are interpreted over discrete traces. A discrete trace is a sequence of states $\sigma = s_0, s_1, s_2, \ldots$ with $s_i \in \Sigma$ for all $i \in \mathbb{N}$. The semantics for RELTL is analogue to the one of HRELTL but restricted to discrete steps only. We refer the reader to Sec. C for more details.

Encoding Hybrid RELTL into Discrete RELTL. We now present a translation of formulas of the linear fragment of HRELTL into equi-satisfiable formulas of RELTL. In the rest of this document we assume that formulas contain only predicates in $PRED_{la}$.

We introduce two Real variables δ_t and ζ respectively to track the time elapsing between two consecutive steps, and to enforce the non-Zeno property (i.e. to guarantee that time diverges). We introduce a Boolean variable ι that tracks if the current state samples a singular interval or an open interval.

We define a formula ψ_ι that encodes the possible evolution of δ_t and ι:

$$\begin{aligned}\psi_\iota := \; & \iota \wedge \zeta > 0 \; \wedge \\ & \mathbf{G}\,((\iota \wedge \delta_t = 0 \wedge \mathbf{X}\,(\iota)) \vee (\iota \wedge \delta_t > 0 \wedge \mathbf{X}\,(\neg\iota)) \vee (\neg\iota \wedge \delta_t > 0 \wedge \mathbf{X}\,(\iota))) \; \wedge \\ & \mathbf{G}\,(\text{NEXT}(\zeta) = \zeta) \; \wedge \\ & \mathbf{G}\,\mathbf{F}\,\delta_t \geq \zeta.\end{aligned} \tag{1}$$

In particular, we force to have a discrete step, which is characterized by two consecutive singular intervals, if and only if $\delta_t = 0$.

For every continuous variable $v \in V_C$ we introduce the Real variable $\dot{v}_l$ and $\dot{v}_r$ that track the left and right derivative of v. We define a formula ψ_{DER} that encodes the relation among continuous variables and their derivatives:

$$\begin{aligned}\psi_{\text{DER}} := \bigwedge_{v \in V_C} \; & ((\delta_t > 0 \wedge \iota) \rightarrow (\text{NEXT}(v) - v) = (\delta_t \times \text{NEXT}(\dot{v}_l))) \wedge \\ & ((\delta_t > 0 \wedge \neg\iota) \rightarrow (\text{NEXT}(v) - v) = (\delta_t \times \dot{v}_r)).\end{aligned} \tag{2}$$

The equation says that, before a point that samples an open interval, the evolution is tracked with the value of left derivative assigned in the sampling point, while afterwards, the evolution is tracked with the right derivative in the same point.

We define a formula ψ_{PRED_ϕ}, being $PRED_\phi$ the set of predicates occurring in ϕ without next variables and derivatives, that encodes the continuous evolution of the predicates.

$$\begin{aligned}\psi_{PRED_\phi} := \; & (\delta_t > 0 \wedge \iota) \; \rightarrow \bigwedge_{p \in PRED_\phi} \text{NEXT}(p_=) \rightarrow p_= \; \wedge \\ & (\delta_t > 0 \wedge \neg\iota) \rightarrow \bigwedge_{p \in PRED_\phi} p_= \rightarrow \text{NEXT}(p_=) \; \wedge \\ & \delta_t > 0 \; \rightarrow \bigwedge_{p \in PRED_\phi} ((p_< \rightarrow \neg\mathbf{X}\,p_>) \wedge (p_> \rightarrow \neg\mathbf{X}\,p_<)).\end{aligned} \tag{3}$$

The first two conjuncts encode that if $p_=$ holds in an open interval, then $p_=$ holds in the immediately adjacent singular intervals too. The third conjuncts encodes that if $p_<$ holds we cannot move to an immediately following state where $p_>$ holds (and vice versa) without passing through a state where $p_=$ holds.

We define ψ_{V_D} to encode that discrete variables do not change value during a continuous evolution:

$$\psi_{V_D} := \delta_t > 0 \rightarrow (\textstyle\bigwedge_{v \in V_d} (\text{NEXT}(v) = v)). \tag{4}$$

Finally, we define the partial translation $\tau'(\phi)$ recursively over ϕ. The translation τ'_a of predicates is defined as:

- $\tau'_a(p_{curr}) = p_{curr}$;
- $\tau'_a(p_{next}) = (\delta_t = 0) \wedge p_{next}$;
- $\tau'_a(p_{der}) = p_{der}[\dot{v}_l/\text{DER}(v)] \wedge p_{der}[\dot{v}_r/\text{DER}(v)]$.

Where, $p[v'/v]$ is predicate p where every occurrence of v is replaced with v'.

The translation τ'_r of SEREs is defined as:

- $\tau'_r(p) = (\delta_t > 0 \wedge \tau'_a(p))$**[*]** ; $\tau'_a(p)$; [4]
- $\tau'_r(\epsilon) = \epsilon$;
- $\tau'_r(r$**[*]**$) = \epsilon \mid \{\tau'_r(r) : \delta_t = 0\}$**[*]** ; $\tau'_r(r)$;
- $\tau'_r(r_1 \text{ ; } r_2) = \{\{\epsilon \text{ \&\& } \tau'_r(r_1)\} \text{ ; } \tau'_r(r_2)\} \mid \{\tau'_r(r_1) \text{ ; } \{\epsilon \text{ \&\& } \tau'_r(r_2)\}\} \mid \{\{\tau'_r(r_1) : \delta_t = 0\} \text{ ; } \{\tau'_r(r_2) : \top\}\}$;

[4] This has the effect that $\tau'_r(p_{next}) = \tau'_a(p_{next}) = (\delta_t = 0 \wedge p_{next})$.

– $\tau'_r(r_1 : r_2) = \tau'_r(r_1) : \delta_t = 0 : \tau'_r(r_2)$;
– $\tau'_r(r_1 \mid r_2) = \tau'_r(r_1) \mid \tau'_r(r_2)$;
– $\tau'_r(r_1 \,\&\&\, r_2) = \tau'_r(r_1) \,\&\&\, \tau'_r(r_2)$.

The translation τ' of HRELTL is defined as:

– $\tau'(p) = \tau'_a(p)$;
– $\tau'(\neg\phi_1) = \neg\tau'(\phi_1)$;
– $\tau'(\phi_1 \wedge \phi_2) = \tau'(\phi_1) \wedge \tau'(\phi_2)$;
– $\tau'(\mathbf{X}\ \phi_1) = \delta_t = 0 \wedge \mathbf{X}\ \tau'(\phi_1)$;
– $\tau'(\phi_1\ \mathbf{U}\ \phi_2) = \tau'(\phi_1)\ \mathbf{U}\ \tau'(\phi_2)$;
– $\tau'(r \diamondsuit\!\!\rightarrow \phi) = \{\tau'_r(r) : \delta_t = 0\} \diamondsuit\!\!\rightarrow \tau'(\phi)$.

Thus, the translation τ for a generic HRELTL formula is defined as:

$$\tau(\phi) := \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{PRED_\phi} \wedge \psi_{V_D} \wedge \tau'(\phi). \tag{5}$$

Remark 1. If ϕ contains only quantifier-free predicates then also $\tau(\phi)$ is quantifier-free. In general, the predicates in $\tau(\phi)$ are non linear. ϕ may contain non linear predicates, even in the case ϕ is in the linear fragment of HRELTL, since it may contain polynomials over discrete variables or multiplications of a continuous variable with discrete variables. Moreover, Equation 2 introduces quadratic equations. Finally, note that if ϕ does not contain SEREs, then the translation is linear in the size of ϕ.

We now define a mapping from the hybrid traces of ϕ to the discrete traces of $\tau(\phi)$, and vice versa. Without loss of generality, we assume that the hybrid trace does not have discontinuous points in the derivatives in the open intervals.

Definition 10. *Given a hybrid trace $\langle \overline{f}, \overline{I} \rangle$, the discrete trace $\sigma = \Omega(\langle \overline{f}, \overline{I} \rangle)$ is defined as follows: for all $i \in \mathbb{N}$,*

– *$t_i = t$ if if $I_i = [t, t]$, and $t_i = (t + t')/2$ if $I_i = (t, t')$;*
– *$s_i(v) = f_i^v(t_i)$;*
– *if $I_i = (t, t')$, $s_i(\dot{v}_l) = (f_i^v(t_i) - f_i^v(t_{i-1}))/(t_i - t_{i-1})$ and $s_i(\dot{v}_r) = (f_i^v(t_{i+1}) - f_i^v(t_i))/(t_{i+1} - t_i)$; if $I_i = [t, t]$ then $s_i(\dot{v}_l) = \dot{f}_i^v(t)_-$ and $s_i(\dot{v}_r) = \dot{f}_i^v(t)_+$;*
– *$s_i(\iota) = \top$ if $I_i = [t, t]$, and $s_i(\iota) = \bot$ if $I_i = (t, t')$;*
– *$s_i(\delta_t) = t_{i+1} - t_i$;*
– *$s_i(\zeta) = \alpha$, such that for all $i \in \mathbb{N}$, there exists $j \geq i$ such that $t_{i+1} - t_i \geq \alpha$ (such α exists for the Cauchy's condition on the divergent sequence $\{t_i\}_{i\in\mathbb{N}}$).*

We then define the mapping in the opposite direction.

Definition 11. *Given a discrete trace σ, the hybrid trace $\langle \overline{f}, \overline{I} \rangle = \Upsilon(\sigma)$ is defined as follows: for all $i \in \mathbb{N}$,*

– *$t_i = \sum_{0 \leq j < i-1} s_j(\delta_t)$,*
– *if $s_i(\iota) = \top$ then $I_i = [t_i, t_i]$ else $I_i = (t_{i-1}, t_{i+1})$,*
– *if $s_i(\delta_t) > 0$ then f_i is the piecewise linear function defined as $f_i^v(t) = s_i(v) - s_i(\dot{v}_l) \times (t_i - t)$ if $t \leq t_i$ and as $f_i^v(t) = s_i(v) + s_i(\dot{v}_r) \times (t - t_i)$ if $t > t_i$.*

Theorem 3 (Equi-satisfiability). *If $\langle \overline{f}, \overline{I} \rangle$ is ground for ϕ and $\langle \overline{f}, \overline{I} \rangle \models \phi$, then $\Omega(\langle \overline{f}, \overline{I} \rangle) \models \tau(\phi)$. If σ is a discrete trace such that $\sigma \models \tau(\phi)$, then $\Upsilon(\sigma) \models \phi$ and $\Upsilon(\sigma)$ is ground for ϕ. Thus ϕ and $\tau(\phi)$ are equi-satisfiable.*

For the proofs we refer the reader to Sec. C of the appendix.

6 Fair transition systems and language emptiness

Fair Transition Systems (FTS) [25] are a symbolic representation of infinite-state systems. First-order formulas are used to represent the initial set of states I, the transition relation T, and each fairness condition $\psi \in F$.

To check the satisfiability of an RELTL formula ϕ with first-order constraints we build a fair transition system S_ϕ and we check whether the language accepted by S_ϕ is not empty with standard techniques. For the compilation of the RELTL formula S_ϕ into an equivalent FTS S_ϕ we rely on the works described in [11, 10].

The language non-emptiness check for the FTS S_ϕ is performed by looking for a lasso-shape trace of length up to a given bound. We encode this trace into an SMT formula using a standard Bounded Model Checking (BMC) encoding and we submit it to a suitable SMT solver. This procedure is incomplete from two point of views: first, we are performing BMC limiting the number of different transitions in the trace; second, unlike the Boolean case, we cannot guarantee that if there is no lasso-shape trace, there does not exist an infinite trace satisfying the model (since a real variable may be forced to increase forever). Nevertheless, we find the procedure extremely efficient in the framework of requirements validation.

The BMC encoding allows us to perform some optimizations. First, as we are considering a lasso-shape path, the Cauchy condition for the non-Zeno property can be reduced to $\mathbf{G}\,\mathbf{F}\,\delta_t > 0$ and no extra variables are needed. Second, whenever we have a variable whose value is forced to remain the same in all moments, we can remove such constraint and use a unique copy of the variable in the encoding.

The definition of HRELTL restricts the predicates that occur in the formula to linear function in the continuous variables in order to allow the translation to the discrete case. Nevertheless, we may have non-linear functions in the whole set of variables (including discrete variables). Moreover, the translation introduces non-linear predicates to encode the relation of a variable with its derivatives.

We aim at solving the BMC problem with an SMT solver for linear arithmetics over Reals. To this purpose, first, we assume that the input formula does not contain non-linear constraints; second, we approximate (2) with linear constraints. Suppose $\text{DER}(v)$ is compared with constants $c_1, \ldots, c_n$ in the formula, we replace the non-linear equations of (2) that are in the form $\text{NEXT}(v) - v = h \times \delta_t$ with:

$$\begin{array}{r} \bigwedge_{1 \le i \le n} (h < c_i \leftrightarrow \text{NEXT}(v) - v < c_i \times \delta_t \quad \wedge \\ h = c_i \leftrightarrow \text{NEXT}(v) - v = c_i \times \delta_t \quad \wedge \\ h > c_i \leftrightarrow \text{NEXT}(v) - v > c_i \times \delta_t)). \end{array} \tag{6}$$

7 Practical experience

The HRELTL language has been evaluated in a real-world project that aims at formalizing and validating the ETCS specification. The project is in response to the ERA tender ERA/2007/ERTMS/OP/01 ("Feasibility study for the formal specification of ETCS functions"), awarded to a consortium composed by RINA SpA, Fondazione Bruno Kessler, and Dr. Graband and Partner GmbH (see http://www.era.europa.eu/public/core/ertms/Pages/Feasibility_Study.aspx for further information on the project). The language used within the project is actually a

superset of HRELTL that encompasses first-order constructs to represent classes of objects and their relationships. The extension enriches the representation power of the discrete part of the specification, and therefore it is orthogonal to the hybrid aspects of the language. The techniques used to handle objects and first-order constraints are described in [10].

We implemented the translation from linear HRELTL to RELTL in an extended version of the NuSMV [9] model checker that interfaces with the MathSAT [5] SMT solver. For an RELTL formula ϕ, we use NuSMV to compile ϕ into an equivalent FTS S_ϕ. Then, we check the language non-emptiness of S_ϕ by submitting the corresponding BMC problem to the MathSAT SMT solver.

We ran the experiments on a 2.20GHz Intel Core2 Duo Laptop equipped with 2GB of memory running Linux version 2.6.24. All the data and binaries necessary to reproduce the results here presented are available at http://es.fbk.eu/people/tonetta/tests/cav09/.

We extracted from the fragment of the ETCS specification a set of requirements that falls in HRELTL and that are relevant for their hybrid aspects. This resulted in a case study consisting of 83 HRELTL formulas, with 15 continuous variables, of which three are timers and two are stop watches. An excerpt of the ETCS specification in HRELTL format is reported in Sec. D.

We first checked whether the specification is consistent, i.e. if it is satisfiable (SAT). Then, we validated the formalization with 3 different scenarios (SCEN_$\{1,2,3\}$), checking the satisfiability of the conjunction of the specification with a formula that represents some assumptions on the evolution of the variables. In all cases, the tool generated a trace proving the satisfiability of the formulas. We then asked the tool to generate witness traces of different increasing lengths k (10, 20, and 30 respectively). We obtained the results reported in Fig. 3(a). In the table we report also the size, in terms of number of variables and number of fairness conditions of the FTS we submit to underlying verification tool. (We use $\#r$,$\#b$,$\#f$ with the meaning, r Real variables, b Boolean variables, and f fairness conditions.) Fig. 3(b) also reports some curves that we can extract from the trace generated by the tool. These curves are the same that are manually depicted in ETCS (Fig. 1(b)). The fact that the automated generated traces resemble the ones inserted in the requirements document makes us more confident that the requirements captures what the designers have in mind.

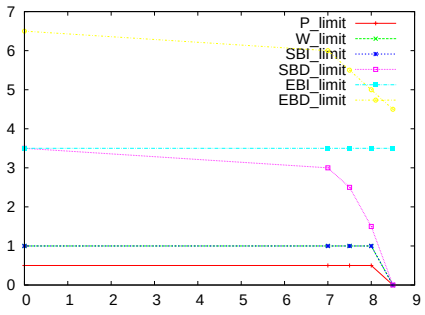
8 Related Work

To the best of our knowledge, this is the first attempt to generalize requirements validation to the case of hybrid domains.

The work most closed to the current paper is described in [13], where LTL with continuous and hybrid semantics is compared with the discrete semantics. It is proved that a positive answer to the model checking problem and to the validity problem with the discrete semantics implies a positive answer to the corresponding problem in the continuous semantics. The properties of finite variability and sampling invariance are introduced. Notably, the hybrid semantics of the logic relies on the hybrid traces accepted by a hybrid system (while our definition is independent).

Name:	Sat	k	$\#r,\#b,\#f$	Mem (MB)	Time (sec)
SAT	Y	10	56,310,17	74.8	11.38
SAT	Y	20	56,310,17	122.4	55.38
SAT	Y	30	56,310,17	173.0	141.20
SCEN_1	Y	10	56,312,19	75.4	15.00
SCEN_1	Y	20	56,312,19	123.1	78.09
SCEN_1	Y	30	56,312,19	174.3	190.73
SCEN_2	Y	10	56,312,19	76.8	17.56
SCEN_2	Y	20	56,312,19	127.1	78.26
SCEN_2	Y	30	56,312,19	180.0	270.59
SCEN_3	Y	10	56,323,18	75.9	19.16
SCEN_3	Y	20	56,323,18	122.5	52.52
SCEN_3	Y	30	56,323,18	173.3	147.97

(a)



(b)

Fig. 3. The results of the experimental evaluation.

Besides [13], our work is also inspired by the ones of [23, 26, 22]. [23, 26] face the problem of the observability of predicates during continuous evolutions, and define phase transition systems, which are a symbolic version of hybrid automata. [22] formally defines a continuous semantics for LTL without next operator and derivatives. In all these works, there is no attempt to solve the satisfiability problem for LTL with the continuous semantics.

Many logics have been introduced to describe properties of timed systems (see [3, 4] for a survey), but none of them can force the continuity of functions, because the semantics is discrete, though in some cases even a dense time domain is considered. Hybrid systems [26, 19] assume that some functions are continuous but the logic used to express the properties is discrete.

In [21, 17], a translation from dense time to discrete time is proposed for a particular class of specifications, but only discrete semantics is considered. In [29], the discretization of hybrid systems is obtained with an over-approximation, while our translation produces an equi-satisfiable discrete formula.

In [18], a framework for specifying requirements of hybrid systems is proposed. However, the techniques are model based, and the requirements are formalized in a tabular notation, which can be seen as a symbolic representation of an automaton.

In [28], a hybrid dynamic logic is proposed for the verification of hybrid systems and it was used to prove safety properties for ETCS. The approach is still model-based since the description of the system implementation is embedded in the logical formula. The regular expression operations are used to define hybrid programs, that represent the hybrid systems. Properties of hybrid programs are expressed with the modalities of first-order dynamic logic. As in RELTL, we use regular expressions with a linear semantics. Moreover, the constraints on the continuous evolution are part of the requirements rather than the system description. As explained in Sec. 2, our approach to the validation of ETCS specifications is property-based.

In [2, 19], linear hybrid automata are defined and a symbolic procedure is proposed to check their emptiness. Besides the different property-based approach that we propose, our techniques differ in the following points. First, instead of a finite set of states, the discrete modes are represented by the infinite (uncountable) set of assignments to the discrete variables. Second, instead of fix-point computations where the image is

based on the quantifier elimination in the theory of Reals, we propose a BMC-based approach with a quantifier-free encoding (this is accomplished by forcing each step to move in a convex region). Related to the encoding of invariants that hold in a continuous evolution, also [30] faces the problem of concave conditions, and splits concave time conditions into convex segments. The condition (3) of our translation has the purpose to split the trace into convex regions in an analogue way.

In [24], a continuous semantics to a temporal logic which does not consider next operators and derivatives is presented. The paper addresses the problem of monitoring temporal properties of circuits with continuous signals.

On a different line of research, Duration Calculus (DC) [7] specifies requirements of real-time systems with predicates over the integrals of Boolean functions over finite intervals of time. Extensions of DC such as Extended Duration Calculus [8] can specify properties over continuous and differentiable functions. DC has been used to specify properties for ETCS [16]. Similarly to DC, Hybrid Temporal Logic (HTL) [20] uses the "chop" operator to express the temporal succession and can express temporal constraints on the derivatives of dynamic functions. Both DC and HTL interpret formulas over intervals of time (rather than infinite sequences of intervals). On the contrary, HRELTL is based on RELTL, which has been consolidated as specification language at the industrial level. HRELTL has the advantage to allow the reuse of requirements analysis techniques for RELTL.

9 Conclusions and future work

In this paper, we tackled the problem of validating requirements for hybrid systems. We defined a new logic HRELTL, that allows to predicate over properties of hybrid traces. Then, we showed that the satisfiability for the linear fragment of HRELTL can be reduced to an equi-satisfiable problem for RELTL over discrete traces. HRELTL was used for modeling in a real-world project aiming at the validation of a subset of the ETCS specification. The validation showed that the temporal requirements of ETCS can be formalized with HRELTL, and the experimental evaluation we carried out showed the practicality of the analysis, based on the use of SMT techniques.

As future work, we will enhance the scalability of the satisfiability procedure, by means of incrementality, lemmas on demand, and abstraction-refinement techniques. We will also consider alternative ways to deal with nonlinear constraints.

References

1. The PROSYD project on property-based system design, 2007. http://www.prosyd.org.
2. R. Alur, C. Courcoubetis, T. A. Henzinger, and P-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, pages 209–229, 1992.
3. R. Alur and T. A. Henzinger. Logics and Models of Real Time: A Survey. 1992.
4. R. Alur and T. A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.
5. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *CAV*, pages 299–303, 2008.

6. D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M.Y. Vardi. Regular Vacuity. In *CHARME*, pages 191–206, 2005.
7. Z. Chaochen, C. A R. Hoare, and A. P. Ravn. A calculus of durations. *Inf. Process. Lett.*, 40(5):269–276, 1991.
8. Z. Chaochen, A. P. Ravn, and M. R. Hansen. An extended duration calculus for hybrid real-time systems. In *Hybrid Systems*, pages 36–59, 1992.
9. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: a new Symbolic Model Verifier. In *CAV 1999*, volume 1633 of *LNCS*, 1999.
10. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Object models with temporal constraints. In *SEFM 2008*, pages 249–258. IEEE press, 2008.
11. A. Cimatti, M. Roveri, and S. Tonetta. Symbolic Compilation of PSL. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(10):1737–1750, 2008.
12. K. Claessen. A coverage analysis for safety property lists. In *FMCAD*, pages 139–145. IEEE, 2007.
13. L. de Alfaro and Z. Manna. Verification in Continuous Time by Discrete Reasoning. In *AMAST*, pages 292–306, 1995.
14. ERTMS/ETCS — Baseline 3: System Requirements Specifications. SUBSET-026-1, i3.0.0, 2008. http://www.era.europa.eu/core/ertms/Pages/FirstETCSSRS300.aspx.
15. H. Eveking, M. Braun, M. Schickel, M. Schweikert, and V. Nimbler. Multi-level assertion-based design. In *MEMOCODE*, pages 85–86. IEEE, 2007.
16. J. Faber and R. Meyer. Model checking data-dependent real-time properties of the european train control system. In *FMCAD*, pages 76–77, 2006.
17. C. A. Furia, M. Pradella, and M. Rossi. Automated Verification of Dense-Time MTL Specifications Via Discrete-Time Approximation. In *FM*, pages 132–147, 2008.
18. C. L. Heitmeyer. Requirements Specifications for Hybrid Systems. In *Hybrid Systems*, pages 304–314, 1995.
19. T. A. Henzinger. The Theory of Hybrid Automata. In *LICS*, pages 278–292, 1996.
20. T. A. Henzinger, Z. Manna, and A. Pnueli. Towards refining temporal specifications into hybrid systems. In *Hybrid Systems*, pages 60–76, 1992.
21. T. A. Henzinger, Z. Manna, and A. Pnueli. What Good Are Digital Clocks? In *ICALP*, pages 545–558, 1992.
22. A. Kapur. *Interval and point-based approaches to hybrid system verification*. PhD thesis, Stanford, CA, USA, 1998.
23. O. Maler, Z. Manna, and A. Pnueli. From Timed to Hybrid Systems. In *REX Workshop*, pages 447–484, 1991.
24. O. Maler, D. Nickovic, and A. Pnueli. Checking Temporal Properties of Discrete, Timed and Continuous Behaviors. In *Pillars of Computer Science*, pages 475–505, 2008.
25. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.
26. Z. Manna and A. Pnueli. Verifying Hybrid Systems. In *Hybrid Systems*, pages 4–35, 1992.
27. I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *DAC*, pages 821–826, 2006.
28. A. Platzer. Differential Dynamic Logic for Verifying Parametric Hybrid Systems. In *TABLEAUX*, pages 216–232, 2007.
29. A. Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.
30. F. Wang. Time-Progress Evaluation for Dense-Time Automata with Concave Path Conditions. In *ATVA*, pages 258–273, 2008.

A Abbreviations

We use the following standard abbreviations:

$\top \doteq p \vee \neg p$	$\textsc{step} \doteq (p_{next} \vee \overline{p}_{next})$
$\varphi_1 \vee \varphi_2 \doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$	$\varphi_1 \rightarrow \varphi_2 \doteq \neg\varphi_1 \vee \varphi_2$
$\mathbf{F}\ \varphi \doteq \top\ \mathbf{U}\ \varphi$	$\mathbf{G}\ \varphi \doteq \neg\mathbf{F}\ \neg\varphi$
$\varphi_1\ \mathbf{V}\ \varphi_2 \doteq \neg(\neg\varphi_1\ \mathbf{U}\ \neg\varphi_2)$	$\varphi_1\ \mathbf{W}\ \varphi_2 \doteq \varphi_1\ \mathbf{U}\ \varphi_2 \vee \mathbf{G}\ \varphi_1$
$r+\ \doteq r; r*$	$r \pmapsto \varphi \doteq \neg r \diamondsuit\!\!\rightarrow \neg\varphi$

B A temporal logic for hybrid traces

Theorem 1 (Finite variability). *Given a formula ϕ, for every hybrid trace $\langle \overline{f}, \overline{I} \rangle$ there exists another hybrid trace $\langle \overline{f}', \overline{I}' \rangle$ which is a sampling refinement of $\langle \overline{f}, \overline{I} \rangle$ and ground for ϕ.*

Proof. The theorem is guaranteed by the analyticity of every function f_i and the analyticity of the functions used to form predicates. Since the composition of analytic functions is analytic, the truth value of a predicate cannot change infinitely many times in a given interval.

Formally, given a predicate p occurring in ϕ, p is in the form $g \bowtie 0$, with $g : \Sigma \rightarrow \mathbb{R}$ analytic. For every $i \in \mathbb{N}$, $f_i : \mathbb{R} \rightarrow \Sigma$ is piecewise analytic. Therefore the composition $g \cdot f_i$ is a real analytic function. Thus, the function is either constantly zero or has a finite number of zeros in I_i. Let such points be the sequence $t_i^1, t_i^2, ..., t_i^{n_i}$. Finally, consider the sampling refinement of $\langle \overline{f}, \overline{I} \rangle$ of $\langle f, I \rangle$ that splits every open interval $I_i = (t_i, t_{i+1})$ into $(t_i, t_i^1), [t_i^1, t_i^1], (t_i^1, t_i^2), ..., (t_i^{n_i}, t_{i+1})$. Note that for all i, for all predicate p occurring in ϕ, suppose p is in the form $g \bowtie 0$, then the interpretation of g in $\overline{I}_i$ either is constantly zero or is never zero. Since g is continuous, the interpretation of p in $\overline{I}_i$ is constant. Therefore $\langle \overline{f}, \overline{I} \rangle$ is ground for ϕ.

Lemma 1. *Given two hybrid traces $\langle \overline{f}, \overline{I} \rangle$ and $\langle \overline{f}', \overline{I}' \rangle$, if $\langle \overline{f}, \overline{I} \rangle$ is ground for ϕ and $\langle \overline{f}', \overline{I}' \rangle \preceq^{\mu} \langle \overline{f}, \overline{I} \rangle$, then, for all $i \in \mathbb{N}$, for all $j \in \mu_i$, $\langle \overline{f}, \overline{I} \rangle, i \models p$ iff $\langle \overline{f}', \overline{I}' \rangle, j \models p$.*

Proof. If $I_i = [t, t]$ then $I_j = I_i$ and thus $\langle \overline{f}, \overline{I} \rangle, i \models p$ iff $\langle \overline{f}', \overline{I}' \rangle, j \models p$. Suppose $I_i = (t, t')$ then either $I_j = (t_1, t_2)$ with $t \leq t_1 < t_2 \leq t'$ or $I_j = [t_1, t_1]$ with $t < t_1 < t'$.

Suppose $I_j = (t_1, t_2)$.

- If p contains only variables in V, $\sigma, i \models p$ iff, for all $t \in I_i$, $f_i(t) \models p$. If, for all $t \in I_i$, $f_i(t) \models p$, then, for all $t \in I_j$, $f_j(t) \models p$ since $I_j \subseteq I_i$ and $f_j = f_i$. If, for all $t \in I_j$, $f_j(t) \models p$, then, for all $t \in I_i$, $f_i(t) \models p$ since $\langle \overline{f}, \overline{I} \rangle$ is ground for p.
- If p contains variables in $V_{\textsc{next}}$ then $\sigma, i \not\models p$ and $\sigma', j \not\models p$.
- If p contains only variables in V, $\sigma, i \models p$ iff, for all $t \in I_i$, $f_i(t), \dot{f}_i(t)_- \models p$ and $f_i(t), \dot{f}_i(t) + - \models p$. If, for all $t \in I_i$, $f_i(t), \dot{f}_i(t)_- \models p$, then, for all $t \in I_j$, $f_j(t), \dot{f}_j(t)_- \models p$ since $I_j \subseteq I_i$ and $f_j = f_i\ \dot{f}_i(t)_- = \dot{f}_j(t)_-$. If, for all $t \in I_j$, $f_j(t), \dot{f}_j(t)_- \models p$, then, for all $t \in I_i$, $f_i(t), \dot{f}_i(t)_- \models p$ since $\langle \overline{f}, \overline{I} \rangle$ is ground for p. The same applies to right derivative.

Lemma 2. *If $\langle \overline{f}, \overline{I} \rangle$ is a trace ground for ϕ and $\langle \overline{f}', \overline{I}' \rangle$ is a sampling refinement of $\langle \overline{f}, \overline{I} \rangle$ by the partitioning μ, then for all $i, j \in \mathbb{N}$, for all $i' \in \mu_i$, $j' \in \mu_j$, for all SERE r that occur in ϕ, $\langle \overline{f}, \overline{I} \rangle, i, j \models r$ iff $\langle \overline{f}', \overline{I}' \rangle, i', j' \models r$.*

Proof. We prove the lemma by induction on the definition of r:

- If $r = p$, for some predicate p, and $i = j$, then i' and j' are indexes of the same continuous evolution. Since $\langle \overline{f}, \overline{I} \rangle$ is ground, $\langle \overline{f}, \overline{I} \rangle, i' \models p$ iff $\langle \overline{f}', \overline{I}' \rangle, j' \models p$. Thus, $\langle \overline{f}, \overline{I} \rangle, i, j \models p$ iff $\langle \overline{f}', \overline{I}' \rangle, i', j' \models p$. Suppose now $r = p$ and $i < j$. By definition of sampling refinement, there are no discrete steps between i and j iff there are no discrete steps between i' and j'. Moreover, since the trace is ground for p, $\langle \overline{f}, \overline{I} \rangle, k \models p$ for all k, $i \leq k \leq j$, iff $\langle \overline{f}, \overline{I} \rangle, k \models p$ for all k', $i' \leq k' \leq j'$.
- If $r = \epsilon$, then $\langle \overline{f}, \overline{I} \rangle, i, j \models r$ iff $i > j$. Though, $i > j$ iff $i' > j'$.
- Suppose $r = r_1*$. This case is proved by induction on $j - i$. As for the base case, $i > j$ iff $i' > j'$. For the inductive hypothesis on r, $\langle f, I \rangle, i, j \models r$, iff $\langle \overline{f}, \overline{I} \rangle, i', j' \models r$ for all $i, j \in \mathbb{N}$, for all $i' \in \mu_i$, $j' \in \mu_j$. Thus, if $i \leq k < j$, $\langle \overline{f}, \overline{I} \rangle, i, k \models r$, $\langle \overline{f}, \overline{I} \rangle, k + 1, j \models r*$ iff $\langle \overline{f}, \overline{I} \rangle, i', k' \models r$, $\langle \overline{f}, \overline{I} \rangle, k' + 1, j' \models r*$. Finally, note that if $I_{k'} = I_{k'+1} = [t, t]$, then $k' \in \mu_k$ iff $k' + 1 \in \mu_{k'+1}$.
- The cases $r = r_1; r_2$ and $r = r_1 : r_2$ are similar to previous one.
- The cases $r_1 | r_2$ and $r_1 \&\& r_2$ can be proved directly from the inductive hypothesis.

Lemma 3. *If two hybrid traces $\langle \overline{f}, \overline{I} \rangle$ and $\langle \overline{f}', \overline{I}' \rangle$ are ground for a formula ϕ and they are one a sampling refinement of the other, then $\langle \overline{f}, \overline{I} \rangle \models \phi$ iff $\langle \overline{f}', \overline{I}' \rangle \models \phi$.*

Proof. We prove by induction on ϕ that, for all $i \in \mathbb{N}$, for all $j \in \mu_i$, $\langle \overline{f}, \overline{I} \rangle, i \models \phi$ iff $\langle \overline{f}', \overline{I}' \rangle, j \models \phi$.

- Suppose p is a predicate; then the claim is true by Lemma 1.
- Suppose ϕ is in the form $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\mathbf{X}\ \phi_1$, $\phi_1\ \mathbf{U}\ \phi_2$. Then the claim follows directly from the inductive hypothesis.
- Suppose ϕ is in the form $r \Diamond\!\!\!\rightarrow \phi_1$. Then for Lemma 2, $\langle f, I \rangle, i, k \models r$, iff $\langle \overline{f}, \overline{I} \rangle, i', k' \models r$ for all $i, k \in \mathbb{N}$, for all $i' \in \mu_i$, $k' \in \mu_k$. Thus, if $i \leq k$, $\langle \overline{f}, \overline{I} \rangle, i, k \models r$, $\langle \overline{f}, \overline{I} \rangle, k \models \phi_1$ then $\langle \overline{f}, \overline{I} \rangle, j, k' \models r$, $\langle \overline{f}, \overline{I} \rangle, k' \models \phi_1$ for some $k' \in \mu_k$ (since μ_k is not empty). If $\langle \overline{f}, \overline{I} \rangle, j, k' \models r$, $\langle \overline{f}, \overline{I} \rangle, k' \models \phi_1$ for some $k' \in \mu_k$, then $i \leq k$, $\langle \overline{f}, \overline{I} \rangle, i, k \models r$, $\langle \overline{f}, \overline{I} \rangle, k \models \phi_1$.

Theorem 2 (Sample invariance). *If $\langle \overline{f}', \overline{I}' \rangle$ is a sampling refinement of $\langle \overline{f}, \overline{I} \rangle$, then the two hybrid traces satisfy the same formulas.*

Proof. Given a formula ϕ, for Theorem 1, both traces $\langle \overline{f}, \overline{I} \rangle$ and $\langle \overline{f}', \overline{I}' \rangle$ have a sampling refinement ground for ϕ, named resp. $\langle \overline{f}_G, \overline{I}_G \rangle$ and $\langle \overline{f}'_G, \overline{I}'_G \rangle$. Since $\langle \overline{f}, \overline{I} \rangle \preceq_{\mu_1} \langle \overline{f}', \overline{I}' \rangle$, $\langle \overline{f}_G, \overline{I}_G \rangle \preceq_{\mu_2} \langle \overline{f}, \overline{I} \rangle$, there exists a refinement μ_3 such that $\langle \overline{f}_G, \overline{I}_G \rangle \preceq_{\mu_3} \langle \overline{f}', \overline{I}' \rangle$. Since $\langle \overline{f}'_G, \overline{I}'_G \rangle \preceq_{\mu_4} \langle \overline{f}', \overline{I}' \rangle$, there exists a sampling refinement $\langle \overline{f}'', \overline{I}'' \rangle$ that merges the splitting of μ_3 and μ_4, thus refining both $\langle \overline{f}_G, \overline{I}_G \rangle$ and $\langle \overline{f}'_G, \overline{I}'_G \rangle$. For Lemma 3, $\langle \overline{f}_G, \overline{I}_G \rangle \models \phi$ iff $\langle \overline{f}'', \overline{I}'' \rangle \models \phi$ iff $\langle \overline{f}'_G, \overline{I}'_G \rangle \models \phi$.

C Reduction to discrete semantics

In this section we give the formal semantics of RELTL and the proofs of our translation.

Given a predicate p and a state s, we assume that the relation $s \models p$ defining when the state satisfies the predicate is given.

Definition 12 (SERE semantics).

- $\sigma, i, j \models p$ *iff,* $i = j$, *and* $\sigma_i \models p$;
- $\sigma, i, j \models \epsilon$ *iff* $i > j$;
- $\sigma, i, j \models r$**[*]** *iff* $i > j$ *or* $\sigma, i, j \models r$, *or there exists* k, $i \leq k < j$, *such that* $\sigma, i, k \models r$, $\sigma, k+1, j \models r$**[*]**;
- $\sigma, i, j \models r_1$ **;** r_2 *iff* $\sigma, i, j \models r_1$, $\sigma, j+1, j \models r_2$, *or* $\sigma, i, i-1 \models r_1$, $\sigma, i, j \models r_2$, *or there exists* k, $i \leq k < j$, *such that* $\sigma, i, k \models r_1$, $\sigma, k+1, j \models r_2$;
- $\sigma, i, j \models r_1$ **:** r_2 *iff there exists* k, $i \leq k \leq j$, *such that* $\sigma, i, k \models r_1$, $\sigma, k, j \models r_2$;
- $\sigma, i, j \models r_1$ **|** r_2 *iff* $\sigma, i, j \models r_1$ *or* $\sigma, i, j \models r_2$;
- $\sigma, i, j \models r_1$ **&&** r_2 *iff* $\sigma, i, j \models r_1$ *and* $\sigma, i, j \models r_2$.

Definition 13 (RELTL semantics).

- $\sigma, i \models p$ *iff* $\sigma_i \models p$;
- $\sigma, i \models \neg\phi$ *iff* $\sigma, i \not\models \phi$;
- $\sigma, i \models \phi \wedge \psi$ *iff* $\sigma, i \models \phi$ *and* $\sigma, i \models \psi$;
- $\sigma, i \models \mathbf{X}\ \phi$ *iff* $\sigma, i+1 \models \phi$;
- $\sigma, i \models \phi\ \mathbf{U}\ \psi$ *iff, for some* $j \geq i$, $\sigma, j \models \psi$ *and, for all* $i \leq k < j$, $\sigma, k \models \phi$;
- $\sigma, i \models r \Diamond\!\!\rightarrow \phi$ *iff, there exists* $j \geq i$, *such that* $\sigma, i, j \models r$ *and* $\sigma, j \models \phi$.

The following three lemmas prove that if σ is a hybrid trace than $\Omega(\sigma) \models \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{PRED_\phi} \wedge \psi_{V_D}$. Vice versa, if $\sigma \models \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{V_D}$, then $\Upsilon(\sigma)$ is a hybrid trace.

Lemma 4. *If* σ *a hybrid trace, then* $\Omega(\sigma) \models \psi_\iota$. *If* $\sigma \models \psi_\iota$, *then the intervals of* $\Upsilon(\sigma)$ *are adjacent and cover* $\mathbb{R}^+$.

Proof. Suppose σ is a hybrid trace. Then $I_0 = [0,0]$ and $s_0(\iota) = \top$. If $I_i = I_{i+1} = [t,t]$ then $s_i(\iota) = s_{i+1}(\iota) = \top$ and $s_i(\delta_t) = 0$. If $I_i = [t,t]$ and $I_{i+1} = (t,t')$ then $s_i(\iota) = \top$, $s_i(\delta_t) = (t'-t)/2 > 0$ and $s_{i+1}(\iota) = \bot$. If $I_i = (t,t')$ and $I_{i+1} = [t,t]$ then $s_i(\iota) = \bot$, $s_i(\delta_t) = (t'-t)/2 > 0$ and $s_{i+1}(\iota) = \top$. Finally, $\bigcup_{i\in\mathbb{N}} I_i = \mathbb{R}^+$ so that the sequence $\{t_i\}_{i\in\mathbb{N}}$ must be divergent. Then, for the Chauchy's condition, there exists $\alpha = s_0(\zeta) > 0$ such that for all $i \in \mathbb{N}$, there exists $j \geq i$ such that $t_{i+1} - t_i \geq \alpha$. and thus $s_j(\delta_t) \geq s_j(\zeta)$.

Suppose $\sigma \models \psi_\iota$. Then $I_0 = [0,0]$ and there exists $\alpha = s_0(\zeta) > 0$ such that for all $i \in \mathbb{N}$, there exists $j \geq i$ such that $s_j(\delta_t) \geq s_j(\zeta)$, and thus $t_{i+1} - t_i \geq \alpha$. Hence, $t_0 = 0$ and $\{t_i\}_{i\in\mathbb{N}}$ diverges, and so $\bigcup_{i\in\mathbb{N}} I_i = \mathbb{R}^+$. If $s_i(\iota) = s_{i+1}(\iota) = \top$ then $I_i = I_{i+1} = [t_i, t_i]$. If $s_i(\iota) = \top$, and $s_{i+1}(\iota) = \bot$ then $I_i = [t_i, t_i]$ and $I_{i+1} = (t_i, t_{i+2})$. If $s_i(\iota) = \bot$, and $s_{i+1}(\iota) = \top$ then $I_i = (t_{i-1}, t_{i+1})$ and $I_{i+1} = [t_{i+1}, t_{i+1}]$.

Lemma 5. *If* σ *a hybrid trace, then* $\Omega(\sigma) \models \psi_{\text{DER}}$. *If* $\sigma \models \psi_{\text{DER}}$, *then for all* $i \in \mathbb{N}$, *for all* $v \in V_C$, *if* $I_i = (t,t')$ *then* $f_i(t) = f_{i-1}(t)$, $f_i(t') = f_{i+1}(t')$.

Proof. Suppose σ is a hybrid trace. If $s_i(\delta_t) > 0 \wedge \iota$ then $I_i = [t,t]$ and $I_{i+1} = (t,t')$ and $s_{i+1}(\dot{v}_l) = (f^v_{i+1}(t_{i+1}) - f^v_{i+1}(t_i))/(t_{i+1} - t_i)$. Since $f^v_{i+1}(t_i) = f^v_i(t_i)$, $s_{i+1}(v) - s_i(v) = \delta_t \times s_{i+1}(\dot{v}_l)$.

If $s_i(\delta_t) > 0 \wedge \neg\iota$ then $I_i = (t,t')$ and $I_{i+1} = [t',t']$ and $s_i(\dot{v}_r) = (f^v_{i+1}(t_{i+1}) - f^v_{i+1}(t_i))/(t_{i+1} - t_i)$. Since $f^v_{i+1}(t_i) = f^v_i(t_i)$, $s_{i+1}(v) - s_i(v) = \delta_t \times s_i(\dot{v}_r)$.

Now, suppose $\sigma \models \psi_{\text{DER}}$.

If $I_i = [t',t']$ and $I_{i+1} = (t',t)$ (first case of Equation 2), $s_{i+1}(v) - s_i(v) = s_i(\delta_t) \times s_{i+1}(\dot{v}_l)$. By Definition 11, $f^v_{i+1}(t) = s_{i+1}(v) - s_{i+1}(\dot{v}_l) \times (t_{i+1} - t)$ if $t \leq t_{i+1}$. In particular, $f^v_{i+1}(t') = s_{i+1}(v) - s_{i+1}(\dot{v}_l) \times s_i(\delta_t)$. Substituting, we get $f^v_{i+1}(t') = s_i(v)$. Since, by Definition 11, $s_i(v) = f^v_i(t')$, the result holds.

If $I_i = (t,t')$ and $I_{i+1} = [t',t']$ (second case of Equation 2), $s_{i+1}(v) - s_i(v) = s_i(\delta_t) \times s_i(\dot{v}_r)$. By Definition 11, $f^v_i(t) = s_i(v) + s_i(\dot{v}_r) \times (t - t_i)$ if $t > t_i$. In particular, $f^v_i(t') = s_i(v) + s_i(\dot{v}_r) \times s_i(\delta_t)$. Substituting, we get $f^v_i(t') = s_{i+1}(v)$. Since, by Definition 11, $s_{i+1}(v) = f^v_{i+1}(t')$, the result holds.

Lemma 6. *If σ a hybrid trace, then If σ is ground for ϕ, $\Omega(\sigma) \models \psi_{PRED_\phi}$. If $\sigma \models \psi_{PRED_\phi}$, then $\Upsilon(\sigma)$ is ground for ϕ.*

Proof. Suppose $I_i = [t,t]$, $I_{i+1} = (t,t')$, and $\Omega(\sigma), i \not\models \text{NEXT}(p_=) \rightarrow p_=$. Since $\overline{p}_=$ is an open set and f_{i+1} is continuous, by topology $f^{-1}(\overline{p}_=)$ must be open. Thus there exist $t'' \in (t,t')$ such that $f_{i+1}(t'') \models \overline{p}_=$, which violates the hypothesis that σ is ground. The case $p_= \rightarrow \text{NEXT}(p_=)$ is similar.

Suppose $\Omega(\sigma)$ does not satisfy $(p_< \rightarrow \neg\mathbf{X}\, p_>)$ for some $p \in PRED$, then by Bolzano's theorem, we can pick a point in the interval such that $p_=$ holds, which violates the hypothesis that σ is ground. The case in which $\Omega(\sigma)$ does not satisfy $(p_> \rightarrow \neg\mathbf{X}\, p_<)$ is similar.

Suppose $\sigma \models \psi_{PRED_\phi}$. Suppose $I_i = (t,t')$ and $f_i(t_i) \models p_=$ then, by hypothesis, $f_i(t) \models p_=$ and $f_i(t') \models p_=$. Since f_i is linear over $[t,t_i]$ and $[t_i,t']$, and $p_=$ is convex, then $f_i(t'') \models p_=$ for all $t'' \in (t,t')$. Suppose $f_i(t_i) \models p_<$ then, by hypothesis, $f_i(t) \models \neg p_>$ and $f_i(t') \models \neg p_>$. Since f_i is linear over $[t,t_i]$ and $[t_i,t']$, and $p_=$ is convex, then $f_i(t'') \models p_<$ for all $t'' \in (t,t')$. The case for $p_>$ is similar. We conclude that $\Upsilon(\sigma)$ is ground.

Lemma 7. *If σ a hybrid trace, then $\Omega(\sigma) \models \psi_{V_D}$. If $\sigma \models \psi_{\text{DER}} \wedge \psi_{V_D}$, then for all $v \in V_D$, f^v_i is constant.*

Proof. For all $v \in V_D$, v is constant when $\delta_t > 0$ and thus $\Omega(\sigma) \models \psi_{V_D}$.

If $\sigma \models \psi_{\text{DER}} \wedge \psi_{V_D}$, if $\sigma_i(\delta_t) > 0$ then for all $v \in V_D$, $\sigma_i(\dot{v}_l)$ and $\sigma_i(\dot{v}_r)$ are zero, and therefore f^v_i is constant.

Lemma 8. *If σ is ground for a predicate p, $\sigma, i \models p$ iff $\Omega(\sigma), i \models \tau'_a(p)$.*

Proof.

- If $\sigma, i \models p_{curr}$ then, for all $t \in I_i$, $f_i(t) \models p_{curr}$ and in particular $f_i(t_i) \models p_{curr}$, i.e., $s_i \models p_{curr}$. Vice versa, if $s_i \models p_{curr}$, then $f_i(t_i) \models p_{curr}$, and since σ is ground for p_{curr}, for all $t \in I_i$, $f_i(t) \models p_{curr}$.
- $\sigma, i \models p_{next}$ iff $I_i = I_{i+1} = [t,t]$ and $f_i(t), f_{i+1}(t) \models p_{next}$, i.e. $s_i(t), s_{i+1}(t) \models p_{next}$ and $\sigma, i \models \delta_t = 0$.

– If $\sigma, i \models p_{der}$ and $I_i = (t, t')$, then, for all $t \in I_i$, $f_i(t), \dot{f}_i(t) \models p_{der}$. In particular $f_i(t_i), \dot{f}_i(t_i) \models p_{der}$. By the mean value theorem there is a point $\bar{t}$ in $(t, (t+t')/2)$ such that $\dot{f}_i(\bar{t}) = s_i(\dot{v}_l)$ and another point in $((t+t')/2, t')$ such that $\dot{f}_i(\bar{t}) = s_i(\dot{v}_r)$. In both cases, since g contains only discrete variables, $f_i(t_i), \dot{f}_i(\bar{t}) \models p_{der}$. Vice versa, if $f_i(t_i), \dot{f}_i(t_i) \models p_{der}$, then there exists $\bar{t}$ as above and since σ is ground for p_{der}, for all $t \in I_i$, $f_i(t), \dot{f}_i(t) \models p_{der}$. If $I_i = [t, t]$, then $f_i(t), \dot{f}_i(t)_- \models p_{der}$ and $f_i(t), \dot{f}_i(t)_+ \models p_{der}$ iff $s_i \models p_{der}[\dot{v}_l/\text{DER}(v)] \wedge p_{der}[\dot{v}_r/\text{DER}(v)]$.

Lemma 9. *If σ is ground for a SERE r, then $\sigma, i, j \models r$ iff $\Omega(\sigma), i, j \models \tau'_r(r)$.*

Proof.
– $\sigma, i, j \models p$ iff, for all k, $i \leq k < j$, there is no discrete step at k ($I_k \neq I_{k+1}$), and, for all k, $i \leq k \leq j$, $\sigma, k \models p$; thus, $\sigma, i, j \models p$, iff for all k, $i \leq k < j$, $\Omega(\sigma), k \models \delta_t > 0$, and, for all k, $i \leq k \leq j$, $\Omega(\sigma), k \models \tau'_a(p)$ (for Lemma 8);
– $\sigma, i, j \models \epsilon$ iff $i > j$ iff $\Omega(\sigma), i, j \models \epsilon$;
– $\sigma, i, j \models r$**[*]** iff $i > j$, or $\sigma, i, j \models r$, or there exists a discrete step at k ($I_k = I_{k+1}$), $i \leq k < j$, such that $\sigma, i, k \models r$, $\sigma, k+1, j \models r$**[*]**; the first case holds iff $\Omega(\sigma), i, j \models \epsilon$; the second case iff there exist n discrete steps at $k_1, k_2, \ldots k_n$ such that $I_{k_i} = I_{k_i+1}$ and $\sigma, i, k_1 \models r$, $\sigma, k_1, k_2 \models r$, ... $\sigma, k_n, j \models r$ iff (for inductive hypothesis) $\Omega(\sigma), i, k_1 \models \tau'_r(r)$, $\Omega(\sigma), k_1 + 1, k_2 \models \tau'_r(r)$, ... $\Omega(\sigma), k_n + 1, j \models \tau'_r(r)$, and $\Omega(\sigma), k_1 \models \delta_t = 0$, $\Omega(\sigma), k_1 + 1, k_2 \models \delta_t = 0$, ... $\Omega(\sigma), k_n + 1, j \models \delta_t = 0$;
– $\sigma, i, j \models r_1$ **;** r_2 iff $\sigma, i, j \models r_1$, $\sigma, j+1, j \models r_2$ (i.e., r_2 accepts the empty word), or $\sigma, i, i-1 \models r_1$, $\sigma, i, j \models r_2$ (i.e., r_1 accepts the empty word), or there exists a discrete step at k ($I_k = I_{k+1}$), $i \leq k < j$, such that $\sigma, i, k \models r_1$, $\sigma, k+1, j \models r_2$; the first case holds, by induction, iff $\Omega(\sigma), i, j \models r_1$, $\Omega(\sigma), j+1, j \models r_2$ and thus iff $\Omega(\sigma), i, j \models \tau'_r(r_1)$ **;** $\{\epsilon$ **&&** $\tau'_r(r_2)\}$; the second case holds, by induction, iff $\Omega(\sigma), i, i-1 \models r_1$, $\Omega(\sigma), i, j \models r_2$ and thus iff $\Omega(\sigma), i, j \models \{\epsilon$ **&&** $\tau'_r(r_1)\}$ **;** $\tau'_r(r_2)$; the third case holds, by induction, iff $\Omega(\sigma), i, i-1 \models r_1$, $\Omega(\sigma), i, j \models r_2$ and thus iff $\Omega(\sigma), i, j \models \{\epsilon$ **&&** $\tau'_r(r_1)\}$ **;** $\tau'_r(r_2)$;
– $\sigma, i, j \models r_1$ **:** r_2 iff there exists a discrete step at k ($I_k = I_{k+1}$), $i \leq k \leq j$, such that $\sigma, i, k \models r_1$, $\sigma, k, j \models r_2$ iff (by inductive hypothesis) $\Omega(\sigma), i, k \models \tau'_r(r_1)$, $\Omega(\sigma), k, j \models \tau'_r(r_2)$, and $\Omega(\sigma), k \models \delta_t = 0$,
– $\sigma, i, j \models r_1$ **|** r_2 iff $\sigma, i, j \models r_1$ or $\sigma, i, j \models r_2$ iff (by inductive hypothesis) $\Omega(\sigma), i, j \models \tau'_r(r_1)$ or $\Omega(\sigma), i, j \models \tau'_r(r_2)$;
– $\sigma, i, j \models r_1$ **&&** r_2 iff $\sigma, i, j \models r_1$ and $\sigma, i, j \models r_2$ iff (by inductive hypothesis) $\Omega(\sigma), i, j \models \tau'_r(r_1)$ and $\Omega(\sigma), i, j \models \tau'_r(r_2)$.

Lemma 10. *If σ is ground for ϕ, then $\sigma, i \models \phi$ iff $\Omega(\sigma), i \models \tau'(\phi)$.*

Proof. – $\sigma, i \models p$ iff $\Omega(\sigma), i \models p$ for Lemma 8;
– $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$ iff $\Omega(\sigma), i \not\models \tau'(\phi)$ (by inductive hypothesis);
– $\sigma, i \models \phi \wedge \psi$ iff $\sigma, i \models \phi$ and $\sigma, i \models \psi$ iff $\Omega(\sigma), i \models \tau'(\phi)$ and $\Omega(\sigma), i \models \tau(\psi)$;
– $\sigma, i \models \mathbf{X}\ \phi$ iff there is a discrete step at i ($I_i = I_{i+1}$), and $\sigma, i+1 \models \phi$ iff $\Omega(\sigma), i \models \delta_t = 0$ and $\Omega(\sigma), i \models \tau'(\mathbf{X}\ \phi)$;
– $\sigma, i \models \phi\ \mathbf{U}\ \psi$ iff, for some $j \geq i$, $\sigma, j \models \psi$ and, for all $i \leq k < j$, $\sigma, k \models \phi$ iff $\Omega(\sigma), j \models \tau(\psi)$ and, for all $i \leq k < j$, $\Omega(\sigma), k \models \tau'(\phi)$;

– $\sigma, i \models r \diamond\!\!\rightarrow \phi$ iff, there exists a discrete step at $j \geq i$ ($I_j = I_{j+1}$) such that $\sigma, i, j \models r$, and $\sigma, j \models \phi$ iff $\Omega(\sigma), i, j \models \tau'_r(r)$, and $\Omega(\sigma), j \models \tau'(\phi)$.

Lemma 11. *If* $\sigma \models \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{PRED_\phi} \wedge \psi_{V_D}$, $\sigma, i \models \tau'(\phi)$ *iff* $\Upsilon(\sigma), i \models \phi$.

Proof.
– If $\sigma, i \models p_{curr}$ then $f_i(t_i) \models p_{curr}$, and since $\Upsilon(\sigma)$ is ground, for all $t \in I_i$, $f_i(t) \models p_{curr}$.
– If $\sigma, i \models (\delta_t = 0) \wedge p_{next}$, then $I_i = I_{i+1} = [t, t]$ and $f_i(t), f_{i+1}(t) \models p_{next}$.
– If $\sigma, i \models p_{der}[\dot{v}_l/\text{DER}(v)] \wedge p_{der}[\dot{v}_r/\text{DER}(v)]$, then, for all $t < t_i$, $f_i(t), \dot{f}_i(t) \models p_{der}$ because $\dot{f}_i(t) = \dot{v}_l$, and, for all $t > t_i$, $f_i(t), \dot{f}_i(t) \models p_{der}$ because $\dot{f}_i(t) = \dot{v}_r$.
– The other cases result directly from the inductive hypothesis similarly to Lemmas 9 and 10.

Theorem 3 (Equi-satisfiability). *If* $\langle \overline{f}, \overline{I} \rangle$ *is ground for* ϕ *and* $\langle \overline{f}, \overline{I} \rangle \models \phi$*, then* $\Omega(\langle \overline{f}, \overline{I} \rangle) \models \tau(\phi)$*. If* σ *is a discrete trace such that* $\sigma \models \tau(\phi)$*, then* $\Upsilon(\sigma) \models \phi$ *and* $\Upsilon(\sigma)$ *is ground for* ϕ*. Thus* ϕ *and* $\tau(\phi)$ *are equi-satisfiable.*

Proof. Suppose $\sigma \models \phi$. Then for Lemmas 4, 5, 6, and 7, $\Omega(\sigma) \models \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{PRED_\phi} \wedge \psi_{V_D}$. For Lemma 10, $\Omega(\sigma) \models \tau'(\phi)$. Thus, $\Omega(\sigma) \models \tau(\phi)$.

Suppose $\sigma \models \tau(\phi)$. Then $\sigma \models \tau'(\phi)$, and, for Lemma 11, $\Upsilon(\sigma) \models \phi$. Finally, for Lemmas 4, 5, 6, and 7, $\Upsilon(\sigma)$ is a hybrid trace.

D ETCS case study in HRELTL

```
----------------------------
-- Background assumptions --
----------------------------

-- declaration of train front end
VAR
  train.front_end: continuous;

-- the train front end does not change during discrete steps
FORMULA
  G ! ( next(train.front_end)!=train.front_end )

-- declaration of train min and max safe_front_end
VAR
  train.min_safe_front_end: continuous;
  train.max_safe_front_end: continuous;

-- relation between train front_end and min_safe_front_end
FORMULA
  G ( train.front_end>=train.min_safe_front_end )

-- relation between train front_end and max_safe_front_end
FORMULA
  G ( train.front_end<=train.max_safe_front_end )
```

```
-- declaration of train speed
VAR
  train.speed: continuous;

-- relation between train front end and speed
FORMULA
  G ( ( (train.speed<0) <-> (der(train.front_end)<0) ) &
      ( (train.speed=0) <-> (der(train.front_end)=0) ) &
      ( (train.speed>0) <-> (der(train.front_end)>0) ) )

-- declaration of level
VAR
  level: 1..3;

-- declaration of MA message events
VAR
  receive_MA: boolean;

-- MA_message.received is an instantaneous event
FORMULA
  G ( receive_MA -> next(receive_MA)=FALSE )

-- declaration of MA message contents
FROZENVAR
  MA_message.time_stamp: real;

-- declaration of passage time over the first balise of the current
-- balise group

FROZENVAR
  current_balise_group.first_balise.passage_time_stamp: real;

-------------------------
-- Scenario conditions --
-------------------------

-----------------------
-- ETCS requirements --
-----------------------

-- 3.8 Movement authority
-- 3.8.1 Characteristics of a MA
-- 3.8.1.1 The following characteristics can be used in a Movement
--         Authority (see Figure 17:
-- Structure of an MA):
-- a) The End Of Authority (EOA) is the location to which the train is
--    authorized to move.
VAR
  eoa: real;
```

```
FORMULA
  G ( eoa>=0 )

-- b) The Target Speed at the EOA is the permitted speed at the EOA;
--    when the target speed is not zero, the EOA is called the Limit
--    of Authority (LOA). This target speed can be time limited.
VAR
  eoa_target_speed: real;
  eoa_target_speed_timeout: real;

FORMULA
  G ( eoa_target_speed >=0 )

FORMULA
  G ( (train.front_end=eoa) -> train.speed<=eoa_target_speed )

-- c) If no overlap exists, the Danger Point is a location beyond the
--    EOA that can be reached by the front end of the train without a
--    risk for a hazardous situation.
VAR
  ol: real;
  dp: real;
  hazard: boolean;

FORMULA
  G ( dp>eoa )

FORMULA
  G ( (ol=0 & train.front_end>dp) -> hazard )

-- d) The end of an overlap (if used in the existing interlocking
--    system) is a location beyond the Danger Point that can be reached
--    by the front end of the train without a risk for a hazardous
--    situation. This additional distance is only valid for a defined
--    time.
VAR
  end_ol: real;

FORMULA
  G ( end_ol>=dp )

FORMULA
  G ( ol = end_ol - dp )

FORMULA
  G ( (train.front_end>end_ol) -> hazard )

-- e) A release speed is a speed under which the train is allowed to
--    run in the vicinity of the EOA, when the target speed is zero. One
--    release speed can be associated with the Danger Point, and another
```

```
--    one with the overlap. Release speed can also be calculated on-board
--    the train (see section 3.13.7).
VAR
  eoa_release_speed: real;
  eoa_vicinity: real;
  dp_release_speed: real;
  dp_vicinity: real;
  ol_release_speed: real;
  ol_vicinity: real;

FORMULA
  G ( ( train.front_end>=(eoa - eoa_vicinity) & eoa_target_speed=0) ->
          train.speed<=eoa_release_speed )

FORMULA
  G ( ( train.front_end>=(dp - dp_vicinity) & ol=0) ->
          train.speed<=dp_release_speed )

FORMULA
  G ( ( train.front_end>=(ol - ol_vicinity) & ol>0) ->
          train.speed<=ol_release_speed )

-- f) The MA can be split into several sections, The last one is
--    called End Section.
--    * A first time out value can be attached to each section.
--      This value will be used for the revocation of the associated
--      route when the train has not entered into it yet. It is
--      called the Section time-out.
--    * In addition, a second time out value can be attached to the
--      End Section of the MA.  This second time out will be used
--      for the revocation of the last section when it is occupied
--      by the train; it is called the End Section time-out.
VAR
  section[1].start: real;
  section[1].end: real;
  section[1].timeout: real;
  endsection.start: real;
  endsection.end: real;
  endsection.timeout: real;
  endsection.end_timer: continuous;
  endsection.end_timeout: real;

FORMULA
  G (section[1].start>=0)

FORMULA
  G (section[1].start<section[1].end)

FORMULA
  G (section[1].end=endsection.start)
```

```
FORMULA
  G (endsection.start<endsection.end)

FORMULA
  G (endsection.end=eoa)

FORMULA
  G ( der(endsection.end_timer)=1 )

-- 3.8.3 Structure of a Movement Authority (MA)

-- 3.8.3.1 The distance to End of Authority (EOA) can be composed of
--         several sections.
-- 3.8.3.2 For each section composing the MA the following information
--         shall be given;
--         a) Length of the section
VAR
  section[1].length: real;
  endsection.length: real;

FORMULA
  G (section[1].length = section[1].end - section[1].start )

FORMULA
G (endsection.length = endsection.end - endsection.start )

--         b) Optionally, Section time-out value and distance from
--            beginning of section to Section Time-out stop location
VAR
  section[1].timeout_stop_location: real;
  endsection.timeout_stop_location: real;

-- 3.8.3.3 In addition, the End section of the MA may include;

--         a) End Section Time-out value and distance from the End
--            Section Time-out start location to the end of the last
--            section
--         b) Danger point information (distance from end of section
--            to danger point, release speed related to danger point)
--         c) Overlap information (distance from end of section to end
--            of overlap, time-out, distance from overlap time-out start
--            location to end of section, release speed related to overlap)
VAR
  endsection.timeout_start_location: real;
  ol.timeout_start_location: real;

-- 3.8.3.4 The time-out start locations (for Overlap and End Section)
--         and the Section Time-out stop location shall be inside of
--         the corresponding section.
```

```
FORMULA
  G (section[1].timeout_stop_location>=section[1].start &
     section[1].timeout_stop_location<=section[1].end)

FORMULA
  G (endsection.timeout_stop_location>=endsection.start &
     endsection.timeout_stop_location<=endsection.end)

FORMULA
  G (endsection.timeout_start_location>=endsection.start &
     endsection.timeout_start_location<=endsection.end)

FORMULA
  G (ol.timeout_start_location>=endsection.start &
     ol.timeout_start_location<=endsection.end)

-- 3.8.4 Use of the MA on board the train
-- 3.8.4.1   End Section Time Out
-- 3.8.4.1.1 The End Section timer shall be started on-board when the
--           train passes the End Section timer start location given by
--           trackside with its max safe front end.
FORMULA
  G ( train.max_safe_front_end!=endsection.timeout_start_location ->
      ( train.max_safe_front_end!=endsection.timeout_start_location W
        ( train.max_safe_front_end=endsection.timeout_start_location &
          next(endsection.end_timer)=0 ) ) )

-- 3.8.4.1.2 The End Of Authority shall be considered as withdrawn to
--           the current position of the train when the time out has expired.
FORMULA
  G ( endsection.end_timer<endsection.end_timeout ->
      ( endsection.end_timer<endsection.end_timeout W
        ( endsection.end_timer=endsection.end_timeout &
          next(eoa)=train.front_end ) ) )

-- 3.8.4.2     Section Time Outs

-- 3.8.4.2.1 The on-board shall start a Section timer for each section:
--           a) For Level 2: at the value of the time stamp of the
--              message including the MA.
--           b) For Level 1: at the time of passage over the first
--              encountered balise of the balise group giving the MA.
VAR
  section[1].timer: continuous;
  section[1].timer_stop: boolean;
  endsection.timer: continuous;
  endsection.timer_stop: boolean;

FORMULA
  G ( section[1].timer_stop -> der(section[1].timer)=0 )
```

```
FORMULA
  G ( !section[1].timer_stop -> der(section[1].timer)=1 )

FORMULA
  G ( (level=2 & receive_MA) ->
      ( next(section[1].timer)=MA_message.time_stamp &
        next(section[1].timer_stop)=FALSE ) )

FORMULA
  G ( (level=1 & receive_MA) ->
      ( next(section[1].timer)=
        current_balise_group.first_balise.passage_time_stamp &
        next(section[1].timer_stop)=FALSE ) )

FORMULA
  G ( endsection.timer_stop -> der(endsection.timer)=0 )

FORMULA
  G ( !endsection.timer_stop -> der(endsection.timer)=1 )

FORMULA
  G ( (level=2 & receive_MA) ->
      ( next(endsection.timer)=MA_message.time_stamp &
        next(endsection.timer_stop)=FALSE ) )

FORMULA
  G ( (level=1 & receive_MA) ->
      ( next(endsection.timer)=
        current_balise_group.first_balise.passage_time_stamp &
        next(endsection.timer_stop)=FALSE ) )

-- 3.8.4.2.2 When the time out has expired, the EOA shall be withdrawn
--           to the entry point of the revoked section.

FORMULA
  G ( section[1].timer<section[1].timeout ->
      ( section[1].timer<section[1].timeout W
        ( section[1].timer=section[1].timeout &
          next(eoa)=section[1].start) ) )

FORMULA
  G ( endsection.timer<endsection.timeout ->
      ( endsection.timer<endsection.timeout W
        ( endsection.timer=endsection.timeout &
          next(eoa)=endsection.start) ) )

-- 3.8.4.2.3 The Section timer shall be stopped when the min safe
--           front end of the train has passed the associated Section
--           Time-out stop location.
```

```
FORMULA
  G ( train.min_safe_front_end<section[1].timeout_stop_location ->
      ( train.min_safe_front_end<section[1].timeout_stop_location W
        ( train.min_safe_front_end=section[1].timeout_stop_location &
          next(section[1].timer_stop)=TRUE ) ) )

FORMULA
  G ( train.min_safe_front_end<endsection.timeout_stop_location ->
      ( train.min_safe_front_end<endsection.timeout_stop_location W
        ( train.min_safe_front_end=endsection.timeout_stop_location &
          next(endsection.timer_stop)=TRUE ) ) )

-- 3.8.4.3   Time-out of the speed associated with the EOA/LOA
-- 3.8.4.3.1 The on-board shall start a timer for the speed at the
--           EOA/LOA:
--           a) For Level 2: at the value of the time stamp of the
--              message including the MA.
--           b) For Level 1: at the time of passage over the first
--              encountered balise of the balise group giving the MA.
VAR
  eoa_target_speed_timer: continuous;

FORMULA
  G ( der(eoa_target_speed_timer)=1 )

FORMULA
  G ( (level=2 & receive_MA) ->
      ( next(endsection.timer)=MA_message.time_stamp ) )

FORMULA
  G ( (level=1 & receive_MA) ->
      ( next(endsection.timer) =
        current_balise_group.first_balise.passage_time_stamp ) )

-- 3.8.4.3.2 When the time out has expired, the speed value shall be
--           set to zero, i.e. the Limit of Authority becomes an End
--           of Authority.

FORMULA
  G ( eoa_target_speed_timer<eoa_target_speed_timeout ->
      ( eoa_target_speed_timer<eoa_target_speed_timeout W
        ( eoa_target_speed_timer=eoa_target_speed_timeout &
          next(eoa_target_speed)=0) ) )

-- 3.8.4.4   Time-out of Overlap

-- 3.8.4.4.1 The overlap timer shall be started on-board when the
--           train passes the overlap timer start location given by
--           trackside with its max safe front end.
```

```
VAR
  ol.timer: continuous;

FORMULA
  G ( der(ol.timer)=1 )

FORMULA
  G ( train.max_safe_front_end!=ol.timeout_start_location ->
      ( train.max_safe_front_end!=ol.timeout_start_location W
        ( train.max_safe_front_end=ol.timeout_start_location &
          next(ol.timer)=0 ) ) )


-- 3.8.4.4.2 When the timer expires, the overlap shall be considered
--           as released. I.e. the Supervised Location is withdrawn to the
--           Danger Point (if any) or to the EOA.

VAR
  ol.timeout: real;

FORMULA
  G ( ol.timer<ol.timeout ->
      ( ol.timer<ol.timeout W
        ( ol.timer=ol.timeout & next(ol)=0 ) ) )

-- 3.8.4.4.3 When the train has passed the overlap timer start
--           location and the train has come to standstill, the
--           overlap shall be considered as released, even if
--           the time-out has not yet expired.

FORMULA
  G ( ( train.front_end>ol.timeout_start_location &
        der(train.front_end)>0 ) ->
         ( der(train.front_end)>0 W
           ( der(train.front_end)=0 & next(ol)=0 ) ) )

-- 3.8.4.5   Supervised Location
-- 3.8.4.5.1 The Supervised Location shall be defined on board as;
--           a) the end of overlap (if any and before time-out).
--           b) if not, the Danger Point (if any).
--           c) if not, the End Of Authority.

VAR
  SL: real;

FORMULA
  G ( SL=(case ol>0: end_ol; dp>eoa: dp; 1: eoa; esac) )

-- 3.13 Dynamic Speed Monitoring
--
```

```
-- 3.13.4   Supervision Limits
--
-- 3.13.4.1 Note: The supervision limits described in the SRS are only
--          defined as far as required for interoperability. The
--          algorithm for their calculation is an implementation
--          matter. This chapter specifies general rules, the
--          following ones special conditions.
-- 3.13.4.2 The speed limits described in this chapter shall be
--          calculated for the Ceiling Speed and Target Speed
--          monitoring, also a subset for the Release Speed
--          monitoring. By comparing the train speed and location of
--          the train to the various supervision limits, the system
--          shall generate indications to the driver and braking
--          actions.
-- 3.13.4.3 The supervision limits shall take into account the input
--          data specified in the previous chapter. This may be done
--          in a simplified way as long as the safety of the train is
--          ensured.
-- 3.13.4.4 The Permitted Speed limit (abbreviated as P)
-- 3.13.4.4.1 The permitted speed is the speed the driver is requested
--          to follow and shall be indicated to the driver.

VAR
  P_limit: continuous;

FORMULA
  G (train.front_end<eoa -> P_limit>0)

FORMULA
  G ! ( next(P_limit)!=P_limit )

-- 3.13.4.5   The Warning limit (abbreviated as W)
-- 3.13.4.5.1 If the train speed exceeds the warning limit, a warning
--            shall be triggered, allowing the driver to avoid brake
--            intervention.

VAR
  W_limit: continuous;
  warning: boolean;

FORMULA
  G (train.front_end<eoa -> W_limit>0)

FORMULA
  G ! ( next(W_limit)!=W_limit )

FORMULA
  G ( train.speed<W_limit ->
      ( train.speed<W_limit W
        ( train.speed=W_limit & next(warning)=TRUE ) ) )
```

```
-- 3.13.4.5.2 After a warning has been triggered it shall remain
--            active until the train speed is equal to or below the
--            permitted speed.
FORMULA
  G ( warning -> ( warning U train.speed<=P_limit ) )

-- 3.13.4.6 The Service Brake Intervention limit (abbreviated as SBI)
--
-- 3.13.4.6.1 The service brake shall be used as First line of system
--            intervention unless
--            a) No interface to the service brake is available on a
--               train
--            b) The use of the service brake is inhibited for
--               Target Speed Monitoring by means of a National Value.
-- 3.13.4.6.2 The Service Brake is considered not safe, therefore it
--            shall be backed up, as Second line of intervention, by
--            the emergency brake. This is ensured by means of the
--            Emergency Brake Intervention limit (see below).
-- 3.13.4.6.3 If the train speed exceeds the Service Brake
--            Intervention limit, the system shall command the service
--            brake to be applied.
VAR
  SBI_limit: continuous;
  service_brake: boolean;

FORMULA
  G (train.front_end<eoa -> SBI_limit>0)

FORMULA
  G ! ( next(SBI_limit)!=SBI_limit )

FORMULA
  G ( train.speed<SBI_limit ->
      ( train.speed<SBI_limit W
        ( train.speed=SBI_limit & next(service_brake)=TRUE ) ) )

-- 3.13.4.6.4 After the service brake has been triggered, the brake
--            command shall be revoked when the train speed is equal
--            to or below the Permitted Speed limit
FORMULA
  G ( service_brake ->
      ( train.speed>P_limit W
        ( train.speed<=P_limit & next(service_brake)=FALSE ) ) )

-- 3.13.4.7   The Emergency Brake Intervention limit (abbreviated as EBI)
-- 3.13.4.7.1 The emergency brake shall be used as First line of
--            intervention in the following cases:
--            a) Release Speed supervision
--            b) Train trip
```

```
--            c) Service brake is not available (refer to 3.13.4.6.1)
-- 3.13.4.7.2 If the train speed exceeds the Emergency Brake
--            Intervention limit, the system shall command the
--            emergency brake to be applied.
VAR
  EBI_limit: continuous;
  emergency_brake: boolean;

FORMULA
  G (train.front_end<eoa -> EBI_limit>0)

FORMULA
  G ! ( next(EBI_limit)!=EBI_limit )

FORMULA
  G ( train.speed<EBI_limit ->
      ( train.speed<EBI_limit W
        ( train.speed=EBI_limit & next(emergency_brake)=TRUE ) ) )

-- 3.13.4.7.3 After the emergency brake has been triggered, the brake
--            command shall be revoked when the train speed is equal
--            to or below the Permitted Speed limit or the train is at
--            standstill, depending on a National Value. There are the
--            following exceptions:
--            a) If there is no interface to the Service Brake provided,
--               the brake command shall be revoked when the train speed
--               is equal to or below the permitted speed. This only
--               applies if the brake intervention was due to passage
--               of the brake intervention curve.
--            b) For Release Speed supervision the brake command
--               shall be revoked at standstill.
--            c) For train trip special conditions apply (see
--               chapter 3.13.8).
FORMULA
  G ( emergency_brake ->
      ( train.speed>P_limit W
        ( train.speed<=P_limit & next(emergency_brake)=FALSE ) ) )

-- 3.13.6     Special Requirements for the Target Speed Monitoring FRS
--            references: 4.3.2 - Dynamic train speed profile
--            calculation, 4.3.7 - Supervision of movement authorities
--            and speed limits, 4.3.7 - Supervision of movement
--            authorities and speed limits
-- 3.13.6.1   General
-- 3.13.6.1.1 The following requirements have to be considered as
--            addition to the general requirements specified in
--            section 3.13.4.
-- 3.13.6.1.2 Note: No special requirements are necessary for the
--            Permitted Speed limit and Warning limit for target speed
--            monitoring.
```

```
-- 3.13.6.1.3 If the service brake is available for target speed
--            monitoring, the Service Brake Intervention (SBI) limit
--            shall comply, in reference to the expected brake
--            performance, with the most restrictive of the following
--            two requirements:
--            a) The SBI limit shall be calculated such that the
--               SBD limit (reflecting the expected deceleration with
--               the service brake fully applied) is not exceeded at
--               the given target location and for the given target
--               speed.
--            b) It shall avoid that the Emergency Brake Intervention
--               limit is reached.
VAR
  SBD_limit: continuous;

FORMULA
  G (der(train.front_end)>0 -> der(SBD_limit)<0)

FORMULA
  G (der(train.front_end)=0 -> der(SBD_limit)=0)

FORMULA
  G (der(train.front_end)<0 -> der(SBD_limit)>0)

FORMULA
  G ! ( next(SBD_limit)!=SBD_limit )

FORMULA
  G (service_brake -> (der(train.speed)=der(SBD_limit)) )

FORMULA
  G (train.speed>SBI_limit V
     (train.front_end=eoa -> train.speed<=SBD_limit) )

-- 3.13.6.1.4 The EBI limit shall be calculated such that the EBD
--            limit (reflecting the expected deceleration with the
--            emergency brake fully applied) is not exceeded at the
--            given target location and for the given target speed,
--            unless the expected brake performance is not available.
VAR
  EBD_limit: continuous;

FORMULA
  G (der(train.front_end)>0 -> der(EBD_limit)<0)

FORMULA
  G (der(train.front_end)=0 -> der(EBD_limit)=0)

FORMULA
  G (der(train.front_end)<0 -> der(EBD_limit)>0)
```

```
FORMULA
  G ! ( next(EBD_limit)!=EBD_limit )

FORMULA
  G (emergency_brake -> (der(train.speed)=der(EBD_limit)) )

FORMULA
  G (train.speed>EBI_limit V
     (train.front_end=eoa -> train.speed<=EBD_limit) )

-- 3.13.6.3    Special Requirements for the EOA section
--             FRS reference: - none
-- 3.13.6.3.1 The EOA section is a section where the permitted speed
--            curve decreases from the current value to zero speed at
--            the EOA.
VAR
  eoa_section.start: real;

FORMULA
  G ( (train.front_end>eoa_section.start) -> der(P_limit)<0 )

-- 3.13.6.3.1.1 Note: For the end of the section the Release Speed
--              supervision may apply (see section 3.13.7).
--
--              Case 1: Service brake is first line of intervention
-- 3.13.6.3.2.1 The supervision limits of the Braking to the EOA
--              section shall be calculated such that
--              a) The EBD curve reaches the zero speed at
--                 the Supervised Location (SL).
--              b) The SBD, W and P curve reaches the zero speed
--                 at the End Of Authority (EOA).
FORMULA
  G ( (train.front_end=SL) -> EBD_limit=0 )

FORMULA
  G ( (train.front_end=eoa) -> SBD_limit=0 )

FORMULA
  G ( (train.front_end=eoa) -> W_limit=0 )

FORMULA
  G ( (train.front_end=eoa) -> P_limit=0 )
```