Automatic Verification of Real-Time Systems : A case study

Danièle Beauquier   Tristan Crolard   Evguenia Prokofieva
E-mail: {beauquier,crolard,prokofieva}@univ-paris12.fr

Laboratory for Algorithmics, Complexity and Logic,
Department of Informatics, University Paris-12, France[1]

Abstract

In [BS02], a decidable class of verification is given using a logical framework based on a first order timed logic (FOTL). We illustrate here the implementation of this approach on a well-known benchmark, the Generalized Railroad Crossing Problem (GRCP).

1 Introduction

In [BS02], a first order timed logic (FOTL) for specification of real-time algorithms is proposed, where time is continuous. This logic permits, on the one hand, to rewrite directly and completely requirements and, on the other hand, to describe executions of various timed algorithms — here block Gurevich Abstract State Machines (ASM) are considered because of their theoretical clarity and sufficient expressive power. Then a decidable class of the verification problem that is based on notions reflecting finiteness properties of systems of control is described. We propose here a tool which implements the algorithm proposed for the decidable class and we illustrate the efficiency of the method by giving an automatic treatment of the known benchmark Generalized Railroad Crossing Problem (GRCP). More precisely, we consider the GRCP in all its generality (the number of tracks and all time constraints are abstract parameters) formalizing directly, practically without any change the requirements as they are given, without modeling, then we specify a controller as a 4-line basic ASM. This basic ASM is automatically translated into a FOTL formula which characterizes its set of runs. All existing proofs of the general case do not follow *directly* the requirements – they change the description of the initial requirements without discussing or proving why it is justified, and all, except that of Gurevich-Huggins, give a much more complicated specification of the controller. The GRCP was considered in dozens of papers. However, treatment of it in the whole generality is rare. We point out the importance of [GH96] that was the starting point for us and that contain a profound analysis of continuous time issues. The first papers where the problem was introduced in the form now usually considered are [HL94, HL96]. A proof to be considered as a complete one is in [HL96]. Other papers with formal analysis of the entire problem are: [Sha93], where an early version of the problem from [HJL93] was analyzed by PVS; [BMSU98] where SteP system was used; [AH98] where PVS was used. There are also numerous other papers where the bounded or simplified versions of the GRCP were model-checked. We do not consider them as we speak about *verification of the entire problem.* To our knowledge, it is the first treatment of the benchmark GRCP where the non automatic part is very reduced and simple.

Section 2 introduces the logical framework and presents decidable classes of verification. In section 3, the Generalized Railroad Crossing Problem is analysed inside this framework. The last section gives a short description of the tool we have implemented.

2 A decidable class of verification

A First Order Timed Logic used in this framework is constructed in two steps. Firstly, we choose a simple, decidable theory to deal with concrete mathematical objects (the underlying theory), like reals, and, secondly,

[1]*Address:* Dept. of Informatics, University Paris 12, 61 Av. du Gén. de Gaulle, 94010, Créteil, France.

we extend it in a 'minimal' way by abstract functions to deal with our specifications. Here we take as the underlying theory the theory of real addition and unary multiplications by rational numbers which is known to be decidable [Wei99], and extend it by functions with at most one time argument and with other arguments being of finite abstract sorts. Precisions are given below.

Notations:

- $\mathbb{R}$ is the set of reals.
- $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ time treated as a subsort of $\mathbb{R}$.
- For a time interval ζ we denote by ζ^- and ζ^+ respectively its *left* and *right ends*.
- *Bool* $= \{\mathbf{true}, \mathbf{false}\}$ are Boolean values; *undef* will be used for *undefined*.
- $\mathbb{Z}$ is the set of integers and $\mathbb{N}$ is the set of natural numbers.

2.1 Syntax and semantics of FOTL.

Syntax of FOTL.

The vocabulary W of a FOTL consists of a finite set of *sorts*, a finite set of *function symbols* and a finite set of *predicate symbols*. To each sort there is attributed a set of variables. Some sorts are predefined, i. e. have fixed interpretations. Here the predefined sorts are the real numbers $\mathbb{R}$ and time $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ treated as a subsort of $\mathbb{R}$. The other sorts are finite. If a finite sort has a concrete cardinality it can be considered as predefined. The interesting finite sorts are those whose cardinality is not specified, for example the number of processes in a distributed algorithm.

Some functions and predicates are also predefined. As predefined constants we take *Bool* for boolean values, *undef* and $\mathbb{Q}$ for rational numbers. Addition $+$, subtraction $-$ and scalar multiplications of reals by rational numbers are predefined functions of the vocabulary. The predicates $=, \leq, <$ over reals are predefined predicates of W. The vocabulary contains $=$ for all types of objects, and the identity function *id* of the type $\mathcal{T} \to \mathcal{T}$ to represent current time.

Any *abstract function* (i. e. without any a priori fixed interpretation) is of the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$, and any *abstract predicate* is of the type $\mathcal{T} \times \mathcal{X} \to Bool$, where $\mathcal{X}$ is a direct product of finite sorts and $\mathcal{Z}$ is an arbitrary sort. The (sub)vocabulary of abstract functions and predicates will be denoted V.

A vocabulary W being fixed, the notion of *term* and that of *formula* over W are defined in a usual way.
Semantics of FOTL.

A priori we impose no constraints on the admissible interpretations. Thus, the notions of interpretation, model, satisfiability and validity are treated as in first order predicate logic modulo the preinterpreted part of the vocabulary. Thus $\mathcal{M} \models F$, $\mathcal{M} \not\models F$ and $\models F$ where $\mathcal{M}$ is an interpretation and F is a formula, denote respectively that $\mathcal{M}$ is a model of F, $\mathcal{M}$ is a counter-model of F and F is valid.

Remark that an interpretation f^* of a function f of type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ describes a family of temporal processes parametrized by the elements of the interpretation $\mathcal{X}^*$ of $\mathcal{X}$.

Below f will stand for a function of the type mentioned above. The function $\lambda t f(t, x)$ for a fixed x will be denoted by f_x.

2.2 Algorithm vocabulary versus verification vocabulary.

Though we do not consider here how to represent runs in FOTL (see [BS02] on this subject), we give some hints on the relation of the vocabulary of an algorithm and the vocabulary needed for requirements and verification by the example that follows.

Example 1. Consider the following algorithm in a self-explanatory notation (in fact, it is a 'basic' Gurevich ASM [Gur95, Gur00]). This algorithm consists of a set $\mathbb{B}$ of non interacting processes. Each process outputs time moments at which it detects a non zero input.

Vocabulary of the algorithm.

Sorts: $\mathcal{T}$, $\mathbb{R}$, *Bool*, $\mathbb{B}$.
Input function: $Inp : \mathbb{B} \to \mathbb{R}$.
Output function: $Out : \mathbb{B} \to \mathcal{T}$.

Proper internal function: $Flag : \mathbb{B} \to Bool$
Initial values (at time moment 0): $Flag(p) = \mathbf{true}$, $Inp(p) = 0$, $Out(p) = 0$.

```
Repeat
  ForAll $p \in \mathbb{B}$ InParallelDo
    If $Flag(p) \wedge Inp(p) = 0$ Then $Flag(p) := \mathbf{false} \,\|\, Out(p) := 0$ EndIf ||
    If $\neg Flag(p) \wedge Inp(p) \neq 0$ Then $Flag(p) := \mathbf{true} \,\|\, Out(p) := CT$ EndIf
  EndForall
EndRepeat
```

To describe the functioning of this algorithm in a FOTL we introduce timed versions of functions. For a function f of a type $\mathbb{B} \to \mathcal{Z}$ we introduce a function f° of type $\mathcal{T} \times \mathbb{B} \to \mathcal{Z}$. And in terms of these functions one can describe the runs of the algorithm.

2.3 Interpretations related to the verification problem.

In the verification setting we distinguish 3 kinds of dynamic functions: external, internal and some auxiliary functions. The latter must be described in terms of external and internal ones. Internal functions are computed by the algorithm and thus, strictly speaking, are described in a piecewise constant way. However, their 'physical' interpretation may be of other nature. For example, to represent a piece of linear function $a \cdot t + b$ on an interval σ we give two values a and b for the function and two values σ^- and σ^+ for the interval. And these values remain constant up to the moment when the algorithm calculates the next piece. But the 'physical' interpretation of this function that may be used in guards of the algorithm is not constant, however, it is described as a term of the vocabulary.

We incorporate these considerations in the following system of notions.

We assume that for every abstract function f of type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ there is fixed a term U_f with values of type $\mathcal{Z}$ constructed only from constants, variables and predefined functions. We may admit a fix number of such terms U_f for a given function (as we do in [BS02]) — the reasoning remains the same with minor technical changes. We need here to consider only terms of the form z with z being a variable or a constant, for a more general setting see [BS02].

We will write U_f also as $U_f(z)$ to make explicit the parameter. Let ζ be an interval. We say that f_x *is U_f-defined on* ζ with parameter z_0, if for $t \in \zeta$ the value $f_x(t)$ is defined as $f_x(t) = z_0$.

Define also U_{id} as t, $U_P(t)$ for $P \in V_{Pred}$ as B, where B is a Boolean variable, and thus, U_f is attributed to every $f \in V$.

A *partition* of $\mathcal{T}$ is a sequence $\pi = (\zeta_i)_{i \in \overline{N}}$ of non empty disjoint intervals where: (1) $\overline{N}$ is a prefix of $\mathbb{N}$, (2) $\bigcup_{i \in \overline{N}} \zeta_i = \mathcal{T}$, (3) $\zeta_i^+ = \zeta_{i+1}^-$ for $0 \leq i \leq |\overline{N}| - 1$, (4) $\zeta_0^- = 0$, $\zeta_k^+ = \infty$ if $\overline{N}$ is finite and k is its last element.

In the logic introduced above one can describe rather directly (see [BS02, BCS00]) the runs of basic Gurevich Abstract State Machines [BS97] or while-programs, transforming the latter into basic Abstract State Machines and applying the transformation from [BS02, BCS00].

2.4 Finiteness Properties

We consider a First Order Timed Logic (FOTL) which is an extension of the theory of reals $\mathbb{R}$ with order, addition and unary multiplications by rationals $\mathbb{Q}$ that are considered as constants in this theory. Such an extension is defined by abstract sorts and abstract functions f of the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$, where $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ is a sort of time treated as a subsort of $\mathbb{R}$, and $\mathcal{Z}$ is either $\mathcal{T}$ or an abstract sort. This abstract sort is finite but of unknown cardinality that can be arbitrary. We assume that there can be also predefined finite sorts (thus of known cardinality).

For technical simplicity we assume that in the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ of an abstract function (more general case was considered in [BS02]) $\mathcal{X}$ is *one* sort, not a direct product, though this sort may depend on the function.

2.4.1 Finite partial interpretations (FPI)

A *partial interpretation (PI)* of f_x is given as a set of disjoint intervals, that will be called the *support* of the PI, and by parameters values to put into U_f to define f_x on each such interval. A *partial interpretation (PI)* of V is a partial interpretation $\mathcal{Y}^*$ of $\mathcal{Y} \subset \mathcal{X}$ and a collection of PIs, one for each f_y, $y \in \mathcal{Y}$.

A partial interpretation $\mathcal{M}'$ of a f_x is an *extension* of a partial interpretation $\mathcal{M}$ if every interval of $\mathcal{M}$ is contained in an interval of $\mathcal{M}'$, and the restriction of the function of $\mathcal{M}'$ on intervals of $\mathcal{M}$ gives the function of $\mathcal{M}$. In a similar way we define an *extension* of a PI of V or W.

A *finite partial interpretation (FPI)* is a PI with a finite number of time intervals.

If any of the mentioned interpretations is defined on a subinterval of $\mathcal{T}$, finite or infinite, then it is *partial.*

When speaking about maximum of complexities we mean

$$\max\{(k_0,k_1),(k_0',k_1')\} = (\max\{k_0,k_0'\},\max\{k_1,k_1'\})$$

Equivalence.

To reduce the complexity of interpretations in spite of a large amount of elements in abstract sorts we introduce a notion of equivalence of interpretations, and on its basis will generalize the complexity measures for PI of individual f. Such an equivalence is defined over elements of abstract sorts for each f.

We will consider only *interval-wise* equivalences. That means that an abstract sort $\mathcal{X}$ is interpreted as an initial interval on natural numbers, and an equivalence is defined by a partition of such an interval into consecutive intervals.

For a PI $\mathcal{M}$ of $f : \mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ with $\mathcal{Y}^*$ partially interpreting $\mathcal{X}$ an equivalence E over $\mathcal{Y}^*$ means that $y_1 E y_2$ implies that the functions $\lambda t f^*(t, y_1)$ and $\lambda t f^*(t, y_2)$ are equal, i. e. have the same support and have the same values on each interval of the support. Denote by m the number of equivalence classes of E, and by K the maximal complexity of $f^*_{y^*}$ for $y^* \in \mathcal{Y}^*$. Then the complexity of the PI with respect to E is (m, K). We will write it also as a triplet (m, L, s).

Given a PI $\mathcal{M}$ of f, without any mentioning of equivalence, we can always find an equivalence E with minimal number of equivalence classes. We define the *complexity of* $\mathcal{M}$ as above with respect to this equivalence E.

Notice that the complexity of a PI $\mathcal{M}$ of $f : \mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ and $g : \mathcal{T} \times \mathcal{X} \to \mathcal{Z}'$ with the same $\mathcal{X}$ will be measured as maximum of complexities of $\mathcal{M}$ as interpretation of f and that of $\mathcal{M}$ as interpretation of g. This means, in particular, that equivalences over the same interpreted part of $\mathcal{X}$ for f and g may be different.

The *complexity of a PI of* V is the maximum of complexities of interpretations of all abstract functions.

The *strong complexity of a PI of* V is $\mathcal{K}$ if the equivalence E and the support related to a class is the same for all abstract functions.

A PI of (strong) complexity $\mathcal{K}$ will be called (strong) $\mathcal{K}$-PI.

We will define the *finiteness* properties in terms of PI contained in models or counter-models $\mathcal{M}$ of the formulas under consideration.

2.4.2 Finite Refutability and Finite Satisfiability - A decidable class

Let α be a total computable function transforming a complexity value of the form (m, s) into a complexity value of the form (m, s).

A formula G is (strongly)-$\mathcal{K}$-*refutable* iff for every its counter-model $\mathcal{M}$ there exists a (strong) $\mathcal{K}$-PI $\mathcal{M}'$ such that $\mathcal{M}$ is an extension of $\mathcal{M}'$ and any extension of $\mathcal{M}'$ to a total interpretation is a counter-model of G.

A formula G is (strongly) $\mathcal{K}$-*satisfiable with augmentation* α iff for every (strong) $\mathcal{K}$-PI $\mathcal{M}$ extendable to a model of G there is an extension $\mathcal{M}'$ of $\mathcal{M}$ with (strong) complexity $\alpha(\mathcal{K})$ that is a model of G.

Remark 1. Finite refutability of properties of functioning of practical real time systems often (maybe almost always) takes place. For example, the safety property is usually finitely refutable and, in a way, it is a general property of safety. As for liveness, too general formulations can be not finitely refutable. For example, if we consider liveness for the mutual exclusion with unbounded waiting time, it will not be finitely refutable. But in any practical system the waiting time is always bounded. If we add such a bound the liveness becomes finitely refutable. And this shows a general principle: adding practical bounds we arrive at finiteness properties. See [BS02] for detailed examples.

Remark 2. Intuitively finite satisfiability of an algorithm means that every its run is reducible in the following sense: every interval of the run can be replaced by a piece of bounded complexity. Many control algorithms possess this property which is, in a way, a finite history property.

Denote by $VERIF_{(strong)}(\mathcal{K},\alpha)$ the class of FOTL-formulas of the form $(\Phi \to \Psi)$ such that the formula Ψ is (strongly) $\mathcal{K}$-refutable and Φ is (strongly) $\mathcal{K}$-satisfiable with augmentation α. In [BS02], it is proven that:

Proposition 1 *A formula of the class $VERIF_{(strong)}(\mathcal{K},\alpha)$ is valid iff it does not admit a counter-model of complexity $\alpha(\mathcal{K})$ and for a given complexity $\mathcal{K}$, and a computable augmentation function α this property is decidable.*

Theorem 1 *Given a complexity $\mathcal{K}$, an augmentation function α , the validity of formulas from $VERIF_{(strong)}(\mathcal{K},\alpha)$ is decidable. Moreover, if a formula of this class is false then its counter-models of complexity $\alpha(\mathcal{K})$ can be described by a quantifier-free formula.*

3 A case study : the Generalized Railroad Crossing Problem

Informal Description of GRCP.

We take the description of GRCP from [GH96]. A railroad crossing has several one-directional tracks and a common gate. Each track admits two sensors, one at some distance of the crossing in order to detect incoming of a train and another one just after the crossing in order to detect the train is leaving. An automatic controller receives the signals from the sensors and on the basis of these signals, decides to send to the gate a signal *close* or *open*. The environment of the functioning of the controller to construct is described by the following assumptions. It is assumed that a train cannot arrive on a track (i. e. in the zone of control) before the previous one has left this track. The situation when a train does not leave the crossing is not formally excluded. It takes at least time d_{min} for a train to reach the crossing after the sensor has detected its incoming. And it takes at most d_{open} (respectively d_{close}) to the gate to be really opened (respectively closed) after the reception of signal to open (respectively, to close) if the opposite signal has not been sent in between. To exclude degenerated cases, it is assumed that $0 < d_{close} < d_{min}$. The time is presumed to be continuous.

The requirements to the controller to construct are the following ones:
(Safety). *If a train is in the crossing, the gate is closed.*
(Liveness). *The gate is open as much as possible.*

Liveness in this formulation implies second order quantifiers (for discussion see [BS02]) and has never been treated in literature in this form. So we will take below a first order formulation in terms of input/output signals (one can show that it gives the liveness in the initial formulation, but it is out of the scope of this paper).

Formal Specification of the Requirements to GRCP.

FOTL permits to represent the informal requirements *directly* without any changes (modulo our remark concerning liveness).

The predefined part of vocabulary W was described above in subsection 2. So we define only the abstract part of W.

Abstract sorts consist of one sort $Tracks$ that represents the set of tracks which number is finite but not fixed. The *variables* for $Tracks$ are x and x with indices.

The part of vocabulary containing *abstract functions* consists of *abstract constants* (static functions of zero arity) d_{min}, d_{open}, d_{close}, all of the type $\to \mathcal{T}$, and of *abstract dynamic functions*. The latter are the following ones:

- $Cmg^\circ : \mathcal{T} \times Tracks \to Bool$ means a presence (coming) of a train on a track at a given time moment;
- $DirOp^\circ : \mathcal{T} \to Bool$ means that a signal to open the gate takes place at a given time moment, and $\neg DirOp^\circ$ means that the signal is to close the gate;
- $InCr^\circ : \mathcal{T} \to Bool$ says that there is a train in the crossing at a given time moment;
- $GtClsd^\circ : \mathcal{T} \to Bool$ says that the gate is closed at a given time moment;
- $GtOpnd^\circ : \mathcal{T} \to Bool$ says that the gate is opened at a given time moment. ($GtOpnd^\circ$ is not the negation of $GtClsd^\circ$ as we know only that the gate cannot be opened and closed at the same time.)

Requirements Specifications of GRCP.

Requirements consist of 2 parts: environment (formula Φ_{Env}) and demands to the functioning (formula Φ_{Func}).

We have no formal notion of train within the given syntax. We assume that for a given track a new train reaches the sensor launching Cmg° only after the previous one has left the crossing making the track status $\neg Cmg^\circ$. The alternation $\neg Cmg^\circ /\ Cmg^\circ /\ \neg Cmg^\circ \ldots$ corresponds to appearance of successive trains on a given track.

Notations:

- $WaitTime = WT =_{df} d_{min} - d_{close}$ will be used to describe a period of time when a train, though having been detected, is far enough from the crossing to permit to close the gate.
- For every function f of type $\mathcal{T} \times \mathcal{X} \to \mathcal{Y}$, every term X of type $\mathcal{X}$ and every term Y of type $\mathcal{Y}$

$LimPlus_f(t, X, Y) =_{df} \exists t_1\, (t_1 > t \wedge \forall \tau\, ((t < \tau \leq t_1) \to f(\tau, X) = Y))$

$LimMinus_f(t, X, Y) =_{df} \exists t_1\, (t_1 < t \wedge \forall \tau\, ((t_1 \leq \tau < t) \to f(\tau, X) = Y))$

- A notion describing when the controller may open the gate is stated as follows:
  $SafeToOpenSp(t) =_{df}$
  $\forall x\, \left[\, \neg Cmg^\circ(t, x) \vee \forall \tau \leq t\, \left(\forall \tau' \in [\tau, t)\, Cmg^\circ(\tau', x) \to t < \tau + WaitTime \right) \right].$

Specification of the Environment.

(TrStInit) $\forall x\, \neg Cmg^\circ(0, x)$
(At the initial moment there are no trains on any track.)

(GtStInit) $GtOpnd^\circ(0)$
(At the initial moment the gate is opened.)

(GtSt) $\forall t\, \neg(GtOpnd^\circ(t) \wedge GtClsd^\circ(t))$
(The gate cannot be closed and opened at the same time, but it can be neither opened nor closed.)

(DirInit) $DirOp^\circ(0)$
(At the initial moment the signal controlling the gate is to open the gate.)

(CrCm) $\forall t\, \left(InCr^\circ(t) \to (t \geq d_{min} \wedge \exists x\, \forall \tau \in [t - d_{min}, t]\, Cmg^\circ(\tau, x)) \right)$
(If a train is in the crossing it had been detected on one of the tracks at least d_{min} time before the current moment.)

(OpnOpnd) $\forall t\, \left(\forall \tau \in (t, t + d_{open}] DirOp^\circ(\tau) \to GtOpnd^\circ(t + d_{open}) \right)$
(If at time $t + d_{open}$ the command has been *open* for at least a duration d_{open} then the gate is opened at this time.)

(ClsClsd) $\forall t\, \left(\forall \tau \in (t, t + d_{close}] \neg DirOp^\circ(\tau) \to GtClsd^\circ(t + d_{close}) \right)$
(If at time $t + d_{close}$ the command has been *close* for at least a duration d_{close} then the gate is closed at this time.)

(Cmg)
$\forall x \forall t\, \left[\, Cmg^\circ(t, x) \to \right.$
$\left. \exists t_0\, \left(0 < t_0 \leq t \wedge \forall \tau \in [t_0, t]\, Cmg^\circ(\tau, x) \wedge LimMinus_{CmgL}(t_0, x, false) \right) \right]$

(NoCmg)
$\forall x \forall t\, \left[\, \neg Cmg^\circ(t, x) \to \right.$

$\exists t_0 \; \big(0 < t_0 \leq t \wedge \forall \tau \in [t_0, t] \neg Cmg^{\circ}(\tau, x) \wedge LimMinus_{(\neg Cmg^{\circ})}(t_0, x, false) \big) \; \big]$

(The two last properties express that the predicate Cmg° is true on intervals closed on the left and opened on the right and that the set of points where the value changes has no accumulation points.)

(dIneq) $0 < d_{close} < d_{min} \wedge 0 < d_{open}$
(These are trivial constraints on the durations involved, the time for closing is smaller than the minimum time of reaching the crossing by any train detected as coming.)

Specification of the Control.

These specifications concern requirements on the functioning.
(Safety): $\forall t \, (InCr^{\circ}(t) \rightarrow GtClsd^{\circ}(t))$.
(When a train is in the crossing, the gate is closed).

(Liveness) or (Utility): $\forall t \, (SafeToOpenSp(t) \rightarrow DirOp^{\circ}(t))$.
(If the zone of control is safe to open at time t then the control signal must be to open the gate).

One can notice that using FOTL permits us to rewrite almost *directly* the environment and requirements specifications without any modeling which could introduce a lot of errors.

Railroad Crossing Controller.

The part of the ASM vocabulary specific to this example consists of
Static functions:
- d_{min}, d_{open}, d_{close}; as above in the logic signature.

External functions:
- CT the current time has type $\rightarrow \mathcal{T}$.
- $Cmg : Tracks \rightarrow Bool$ is an input function giving for every track its status (coming or empty).

Internal functions:
- $DirOP$ says that the signal to open the gate is being generated by the algorithm, its type is $\rightarrow Bool$.
- $DL : Tracks \rightarrow \mathcal{T}$ is the first moment of appearance of a train on a given track plus $WaitTime$, and this value is then used as a *DeadLine* to decide on control of the gate, see $SafeToOpen$ condition below.
- $NoDL : Tracks \rightarrow Bool$ says that there is no deadline on a given track.

Notation:
$SafeToOpen =_{df} \forall x \; \big(\neg Cmg(x) \vee NoDL(x) \vee CT < DL(x) \big)$.

Remark that this $SafeToOpen$ is presumed to represent adequately the $SafeToOpenSp$ condition, but it is to be proved.

```
Repeat
  ForAll $x \in Tracks$
  InParallelDo
    If $Cmg(x)$ and $NoDL(x)$ Then $NoDL(x) := false$;
                 $DL(x) := CT + WT$ EndIf
    If $\neg Cmg(x)$ and $\neg NoDL(x)$ Then $NoDL(x) := true$ EndIf
    If $DirOp$ and $\neg SafeToOpen$ Then $DirOp := false$ EndIf
    If $\neg DirOp$ and $SafeToOpen$ Then $DirOp := true$ EndIf
  EndInParallelDo
  EndForall
EndRepeat
```

Figure 1: Railroad Crossing Controller

An algorithm to control the railroad crossing is given in Figure 3. To distinguish it from that of [GH96] we will name it Symmetric Controller as it uses our version of $SafeToOpen$ condition in a symmetric way. Below we will refer to this algorithm simply as Controller.

The initial values of internal functions are defined by the condition

$$Init =_{df} \forall x \; \big(\; NoDL(x) \wedge DL(x) = 0 \; \big) \wedge DirOp$$

3.1 A decidable class for the GRCP

Finite Refutability of Safety in the Railroad Crossing Problem.

Theorem 2 *(Safety) is a strongly* $(1,1)$*-refutable.*

Proof. Show that the formula $\forall t \, (InCr^{\circ}(t) \to GtClsd^{\circ}(t))$ for (Safety) is strongly $(1,1)$-refutable. Let $\mathcal{M}$ be a counter model for it. Then there is a train in the crossing and the barrier is not closed at some time t_0. We take $k = 1$ and $n = 1$. The FPI $\mathcal{M}'$ extendable in $\mathcal{M}$ is chosen as follows. An arbitrary track x is taken, and for this class, the interval of time is $\{t_0\}$. So $\mathcal{M}'$ is a $(1,1)$-FPI and every interpretation which extends $\mathcal{M}'$ is a counter-model of (Safety). ■

Finite Refutability of Utility in the Railroad Crossing Problem.

The formula for (Utility) is $\forall t \, (SafeToOpenSp(t) \to DirOp^{\circ}(t))$, where

$$SafeToOpenSp(t) =_{df}$$
$$\forall x \; \big[\; Emp^{\circ}(t,x) \vee \forall \tau \le t \; \big(\forall \tau' \in [\tau, t) \, Cmg^{\circ}(\tau', x) \to t < \tau + WaitTime \; \big) \; \big].$$

Let $\mathcal{M}$ be a counter-model for (Utility). Then for some t we have the situation shown on Figure 2 where the tracks x with $Emp^{\circ}(t,x)$ are not shown.

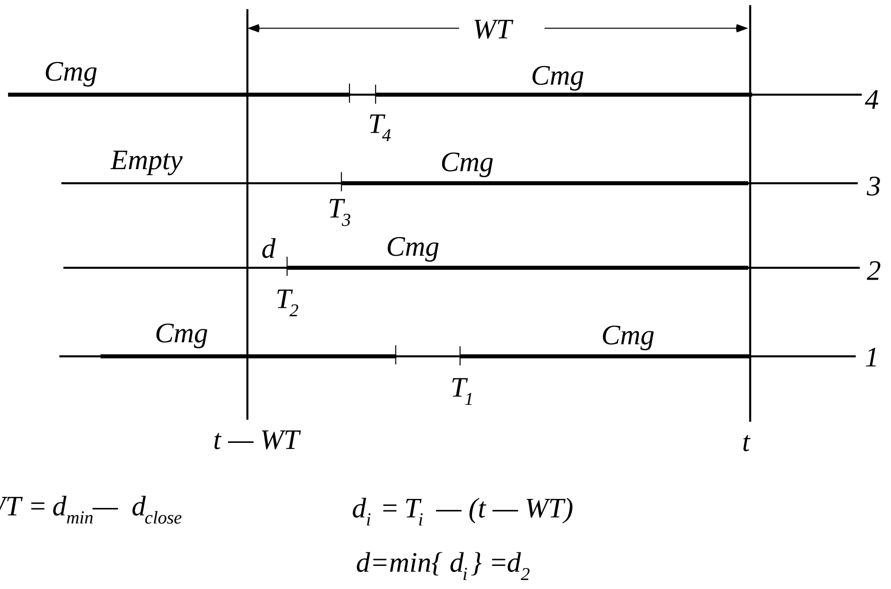

Figure 2: Utility counter-model.

Denote by X the set of tracks x such that $Cmg^{\circ}(t,x)$.

For $x \in X$ let T_x be the moment of coming of this train on the track x, i. e. $T_x = \inf\{T \le t : \forall \tau \in [T,t] \; Cmg(t,x)\}$. Clearly, $\mathcal{M}$ is a counter-model iff $T_x > t - WT$ for all $x \in X$ and $DirCl^{\circ}(t)$.

Consider the intervals $\zeta_x = [t - WT, T_x)$, $x \in X$. If all of them had the track status *Empty* then a FPI, that ensures that the extensions are counter-models, would be $Emp^{\circ}(\tau, x)$ for $\tau \in [t - WT, \min_{x \in X}\{T_x\})$, for $x \in X$, $Emp^{\circ}(t,x)$, for $x \notin X$, and $DirCl^{\circ}(t)$. Hence, we have defined a $(2,1)$-FPI restriction of $\mathcal{M}$ which every extension is a counter-model of (Utility).

But in the general case, if there is not a lower bound sufficiently large between the moment when a train leaves some track and the moment when the next train arrives on this same track there is no possibility to have a FPI extendable into counter-models, independent of the number of tracks. Thus, with an additional reasonable requirement (that always takes place in practice) that after a train having left the track at some

t the next one cannot appear before $t+\theta$ on this track, with θ such that $\frac{WT}{\theta} \leq K$ for some constant $K \in \mathbb{N}$, (Utility) is finitely refutable.

Indeed, divide $(t-WT, t)$ into $2K$ intervals of equal length. Since $\mathcal{M}$ is a counter-model, for every track $x \in X$ there is such an interval where $Emp^{\circ}(\tau, x)$ holds. Take this interval as $\mathcal{T}$-support for $\lambda\tau Emp^{\circ}(\tau, x)$. For $x \notin X$, take $\{t\}$ as $\mathcal{T}$-support for $\lambda\tau Emp^{\circ}(\tau, x)$. For $DirCl^{\circ}$ take $\{t\}$ as $\mathcal{T}$-support. This defines a $(2K+1, 1)$-FPI restriction of $\mathcal{M}$ which every extension is a counter-model of (Utility).

Thus, we have :

Theorem 3 *(Utility) is strongly* $(2K+1, 1)$*-refutable (under the above mentioned hypothesis* $\frac{WT}{\theta} \leq K$*).*

$\mathcal{F}$-satisfiability of $\Phi_{Runs} \wedge \Phi_{Env}$ in the Generalized Railroad Crossing Problem.

Let $\mathcal{M}$ be a model of $(\Phi_{Runs} \wedge \Phi_{Env})$ and $\mathcal{M}_1$ be a strong (k, n)-FPI, restriction of $\mathcal{M}$. Let X be one of the k classes of tracks, and two consecutive intervals I and J of the support related to this class. First one can suppress on each of these tracks the occurrence of a train which arrives and leaves the zone after I and before J. The new model of $(\Phi_{Runs} \wedge \Phi_{Env})$ admits the same FPI $\mathcal{M}_1$. If on some track of X a train is coming during I and leaves the zone between I and J, it is the same for all the tracks in X, and we can take as a time to leave the zone for all the trains on these tracks the latest time. The new model of $(\Phi_{Runs} \wedge \Phi_{Env})$ admits again the same FPI $\mathcal{M}_1$. Secondly, if a train arrives in the zone after I and before J, and is coming during J, it is the same for all the tracks in X, and we can take as a time to arrive in the zone for all the trains on these tracks the earliest time. The new model of $(\Phi_{Runs} \wedge \Phi_{Env})$ admits again the same FPI $\mathcal{M}_1$. We repeat this transformation for each pair of consecutive intervals (a similar transformation is done before the first interval and after the last one). We get a model of $(\Phi_{Runs} \wedge \Phi_{Env})$ which restriction to X has a complexity equal to $6n$ (a complete phase for one train involves 6 intervals). Thus, we apply this transformation to each class. If the complement of the k classes is not empty, it is enough to take one more track, and on this track an interpretation of complexity $6n$ to ensure the correct value of functions which do not depend on tracks like $InCr^{\circ}$ and $GtClsd^{\circ}$. We get finally a model of $(\Phi_{Runs} \wedge \Phi_{Env})$ which is a strong $(k+1, 6n)$-FI which is an extension of $\mathcal{M}_1$. Therefore we have:

Theorem 4 $(\Phi_{Runs} \wedge \Phi_{Env})$ *is strongly* (k, n)*-satisfiable with augmentation* $\alpha(k, n) = (k+1, 6n)$.

Verification of $(Safety)$ in the Generalized Railroad Crossing Problem.

Using Theorems 2, 4, and Proposition 1 we deduce that $(\Phi_{Runs} \wedge \Phi_{Env}) \rightarrow (Safety)$ is in the class $VERIF_{strong}((1, 1, \alpha)$ where $\alpha(k, n) = (k+1, 6n)$, and so has a counter model iff it has a counter model of complexity $(2, 6)$.

4 The tool

The tool consists into three modules.
• The first one, the ASM2FOTL translator is a tool for translating a Timed ASM (Gurevich Abstract State Machine) into a FOTL-formula. The translator is written in Objective Caml. It permits to compute the FOTL-formula associated to a basic Gurevich machine (http://www.univ-paris12.fr/lacl/).
• The second one, FOTL2Reduce which is the new part in this paper, implements the algorithm associated to Theorem 1 and permits to decide whether a given FOTL-formula φ admits a model of a given complexity (k, n). The algorithm transforms a FOTL-formula Ψ into a formula $\hat{\Psi}_{k,n}$ such that Ψ admits a model of complexity (k, n) iff $\hat{\Psi}_{k,n}$ is valid (in the case when Ψ is closed).

The formula $\hat{\Psi}_{k,n}$ contains real and integer variables and addition, but one can remark that we can eliminate quantifiers by applying consecutively quantifier elimination for theory of real addition or Presburger arithmetic. These two sorts of variables are well separated: an atomic formula contains either only real variables or only integer variables (and even no addition in this case).

Remark 3. If Ψ is in a strong class, since the supposed counter-model is a strong (k, n)-FI, the partition of the abstract sort is the same for all abstract functions, and then one can suppose that this abstract sort has cardinality k, and each class has exactly one element. In that case, integer existential (universal) quantifiers can be replaced by finite disjunctions (conjunctions) and there is no problem of elimination of integer quantifiers.

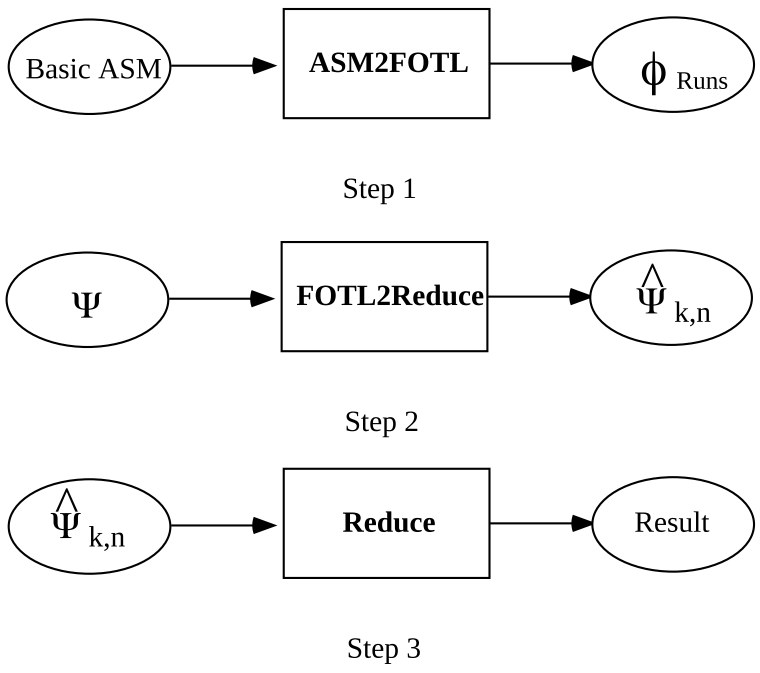

Figure 3: Steps of Verification.

Remark 4. In the case of a formula which is not in a strong class, since the signature for integers contains only the order (no addition), we have developped and implemented an algorithm of elimination of integer quantifiers which is simpler than in Presburger arithmetic.

• In the last step, the software Reduce (http://www.rrz.uni-koeln.de/REDUCE/) is used to eliminate the real quantifiers in $\hat{\Psi}_{k,n}$ and to get the necessary and sufficient condition for Ψ to admit a model of complexity (k, n).

References

[AH98] M. Archer and C. Heitmeyer. Mechanical verification of timed automata: A case study. Technical Report 5546–98–8180, University Paris-12, Department of Informatics, Naval Reserach Laboratory, Washington, 1998. NRL Memorandum Report.

[BCS00] D. Beauquier, T. Crolard, and A. Slissenko. A predicate logic framework for mechanical verification of real-time Gurevich Abstract State Machines: A case study with PVS. Technical Report 00–25, University Paris 12, Department of Informatics, 2000. Available at http://www.univ-paris12.fr/lacl/.

[BMSU98] N. Bjørner, Z. Manna, H. Sipma, and T. Uribe. Deductive verification of real-time systems using STeP. Technical Report Technical Report STAN–CS–TR–98–1616, Computer Sci. Dept., Stanford Univ., December 1998. Submitted to *Elsevier Science*.

[BS97] D. Beauquier and A. Slissenko. On semantics of algorithms with continuous time. Technical Report 97–15, Revised version., University Paris 12, Department of Informatics, 1997. Available at http://www.eecs.umich.edu/gasm/ and at http://www.univ-paris12.fr/lacl/.

[BS02] D. Beauquier and A. Slissenko. A first order logic for specification of timed algorithms: Basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113(1–3):13–52, 2002.

[GH96] Y. Gurevich and J. Huggins. The railroad crossing problem: an experiment with instantaneous actions and immediate reactions. In H. K. Buening, editor, *Computer Science Logics, Selected papers from CSL'95*, pages 266–290. Springer-Verlag, 1996. Lect. Notes in Comput. Sci., vol. 1092.

[Gur95] Y. Gurevich. Evolving algebra 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–93. Oxford University Press, 1995.

[Gur00] Y. Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.

[HJL93] C. Heitmeyer, R. Jeffords, and B. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *Proc. of the 10th IEEE Workshop on Real-Time Operating Systems and Software, New York*. IEEE, 1993.

[HL94] C. Heitmeyer and N. Lynch. The generalized railroad crossing: a case study in formal verification of real-time systems. In *Proc. of Real-Time Systems Symp., San Juan, Puerto Rico*. IEEE, 1994.

[HL96] C. Heitmeyer and N. Lynch. Formal verification of real-time systems using timed automata. In C. Heitmeyer and D. Mandrioli, editors, *Formal Methods for Real-Time Computing*, pages 83–106. John Wiley & Sons, 1996. In series: "Trends in Software", vol. 5, Series Editor: B. Krishnamurthy.

[Sha93] N. Shankar. Verification of real-time systems using PVS. In *Proc. 5th International Computer Aided Verification Conference*, pages 280–291, 1993.

[Wei99] V. Weispfenning. Mixed real-integer linear quantifier elimination. In *Proc. of the 1999 Int. Symp. on Symbolic and Algebraic Computations (ISSAC'99)*, pages 129–136. ACM Press, 1999.