The Railroad Crossing Problem: Towards Semantics of Timed Algorithms and Their Model Checking in High Level Languages

Danièle Beauquier[1]
Université Paris-12 and L.I.T.P., Paris, France

Anatol Slissenko[2]
Université Paris-12 and L.I.T.P., Paris, France
and Laboratory for Theory of Algorithms,
SPIIRAN[†]*, St-Petersburg, Russia*

Abstract. The goal of this paper is to analyse semantics of algorithms with explicit continuous time with further aim to find approaches to automatize model checking in high level, easily understandable languages. We give here a general notion of timed transition system and its formula representation that are sufficient to deal with some known examples of timed algorithms. We prove that the general semantics gives the same executions as direct, more intuitive interpretations of executions of algorithms. In a way, we try to give a general treatment of considerations of Yu.Gurevich and his co-authors concerning concrete Gurevich machines (called *evolving algebras* in [Gur95]), in particular, related to Railroad Crossing Problem [GH96]. Besides that we formalize specifications of this problem in a high level language which permits to rewrite directly natural language formulations, and to give a formal proof of correctness of the railroad crossing algorithm using rather a small amount of logical means, and this leads to hypotheses how automatize inference search.

1 Introduction

The goal of this work is to make a formal analysis of model checking for a particular problem with explicit time constraints, namely, the Railroad Crossing Problem, in order to find an appropriate general notion of timed transition system to describe semantics of algorithms with continuous time. Continuous time has many intuitive and algorithmic advantages with respect to discrete time (as

[1] *Address: University Paris-12, Dept. of Informatics, 61, Av. du Gén. de Gaulle, 94010 Créteil, France.* E-mail: *beauquier@univ-paris12.fr*
[2] *Address: University Paris-12, Dept. of Informatics, 61, Av. du Gén. de Gaulle, 94010 Créteil, France.* E-mail: *slissenko@univ-paris12.fr*
[†] *St-Petersburg Inst. for Informatics and Automation of the Acad. Sci. of Russia*

well as in classical domains as mechanics or physics). The underlying question is whether one can hope to find algorithmic tools supporting model checking if easily comprehensible languages are used to describe specifications. Usually, easily comprehensible languages have no general efficient algorithms for model checking not to speak about satisfiabilty. We hope that some useful algorithmic tools can be developed for classes of problems containg practical ones, and the presented analysis leads to some hypotheses on what features of systems under consideration might assure efficiency.

Our analysis of the Railroad Crossing Problem is based on Gurevich-Huggins paper [GH96]. The profound analysis of treatment of continuous time given in [GH96] was an essential stimulus for our work.

Efficient algorithms for model checking are mostly associated with temporal logics [Eme90] as requirement specification languages, and with timed automata [AD94] or regular process algebras [Mil90] as algorithms specification languages. Whatever impressive be the achievements of research on temporal logics and their applications to model checking (e. g. [Eme90, Eme96, MP92]), some of their evident shortcomings such as hardness of understanding of temporal logic formulas inhibit their wide practical applications. Lack of explicit time is among the shortcomings of temporal logics, and it is not easy to remedy them (see, e. g. [Han94]), not speaking that the initial idea of temporal logics was to avoid explicit usage of time. On the other hand, easily understandable formalisms usually have no efficient algorithmic support even for particular interesting classes of practical problems. As two "high-level" languages for specification we take: Gurevich machines [Gur95] for specifying algorithms, and an extension of theory of real addition to specify requirements. Gurevich machines have the following advantages: they are self-explanatory and, thus, well understandable in concrete situations, they have lucid underlying theoretical ideas, in particular, concerning the semantics, and they permit to change levels of abstraction easily.

1.1 Informal Description of the Railroad Crossing Problem.

The Railroad Crossing Problem appears in various forms in papers on model checking of timed systems, we take a general version from [GH96]. An informal description of Railroad Crossing Problem is as follows. A railroad crossing has several parallel train tracks and a common gate. Each track admits in each direction 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 correctness requirements to satisfy by the controller (i. e. by the algorithm 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.

Note that safety alone is easy to satisfy with the gate always closed.

Some assumptions are usually done. It is assumed that a train cannot arrive on a track (in the zone of control) before the previous one has left this track. If a train is coming from the left, it leaves the crossing on the right and conversely.

The situation when a train does not leave the crossing is not 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 is not sent in between. To exclude degenerated case, it is assumed that at least $d_{close} < d_{min}$.
In our analysis of the problem we separate two concerns: declarative requirement specification and operational algorithm specification. Even if not to discuss automation of the specification analysis, formalizing the two basic poles of specifications imposes a kind of discipline and facilitates specification verification.

2 Requirement Specification Language

As specification language we take an extension of the theory of real addition with symbols of functions defined either on time domain or on finite domains specific to problem under consideration and having values also of these types. Clearly, the satisfiability problem is undecidable even for rather moderate extensions of this kind.

2.1 Continuous Time.

The interpreted part of the language consists of a domain for time. Here we take as *time* the set of reals $\mathcal{T}_0 =_{df} \boldsymbol{R}$, and, for the purposes of treatment of instantaneous actions, extend it to the set $\mathcal{T}$ by non standard numbers and, for technical reasons, by a special symbol ∞ to follow [GH96]. For basic requirement specifications, that treat user's properties of the system under consideration, we use only $\mathcal{T}_0$.
The treatment of infinitesimals will be semantical, and we will distinguish the two sets, that of standard time $\mathcal{T}_0$ and its extension $\mathcal{T}$ by non standards elements. Defining extensions of functions over standard reals to non standard ones in our case is always evident.
The symbol ∞ has the property: $\forall t \in \mathcal{T}\ (t < \infty)$.
We fix two functions giving for every $t \in \mathcal{T}$ two non standard reals t^- and t^+ such that $t^- < t < t^+$ and $StandardPart(t^-) = StandardPart(t^+) = StandardPart(t)$.
The properties of t^+ and t^- used in our proof of correctness can be easily formulated, and we omit them here as we do not discuss the proof in detail.
An initial specification of the problem is usually of declarative nature, and includes specification of the environment to control and that of requirements of control. The signature of these specifications do not include the functions representing the own identifiers of an algorithm to construct as solution for the problem of control. However, the signature of initial specifications contains functions representing inputs which will be also used by the algorithm. The identifiers of the algorithm, whose values implicitely depend on time, may not contain time parameter explicitely, contrary to the corresponding functions of the logic language. To distinguish by style the identifiers of the algorithm and the corresponding identifiers which depend on time: roman is used for time dependant

identifiers and *italic* for the identifiers of the algorithm; the constants independent of time will be in italic in both cases.

2.2 Signature of Logic Specifications for the Railroad Crossing Problem.

Specifying a problem we speak about standard time $\mathcal{T}_0$, and then to make precise semantics and to give a formal proof of correctness of an algorithm necessitates to extend the domain of time to $\mathcal{T}$. With this extension the domain of time variables and time depending functions are extended to $\mathcal{T}$ in an obvious way.
Time Variables and Constants :
• $t, \tau, \zeta, \xi, t', t_0, \ldots$ are variables for time.
• $d_{min}, d_{max}, d_{open}, d_{close}$ are non interpreted standard constants for time; their meta-meaning is the following:

- After the moment an incoming train having been detected, it takes time between d_{min} and d_{max}, $d_{min} \leq d_{max}$, for the train to reach the crossing.

- The gate closes within time d_{close} and opens within time d_{open} , more precisely: for each interval $\alpha = (t, t + d_{close}]$ (respectively $\alpha = (t, t + d_{open}]$) during which the signal to close (respectively, to open) is in force, the gate is closed (respectively, opened) at $t + d_{close}$ (respectively, at $t + d_{open}$).

• **Notation:** $WaitTime =_{df} d_{min} - d_{close}$ 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 open the gate.
Variables and Constants that are used to describe railroad crossing:
• *Tracks* is the set of tracks, its cardinality $|Tracks|$ being fixed.
• $x, y, \ldots$ are variables for tracks.
• *coming*, *empty* are states of a track; they constitute the values of a function describing the gate status.
• *open* , *close* are signals of control to open, respectively to close the gate. These signals must be produced by the algorithm of control.
• *opened*, *closed* (and *udef*) are states of the gate.
Functions depending on time:
• TrackStatus : $\mathcal{T} \times Tracks \rightarrow \{coming, empty\}$ is an *input* function detecting incoming of a train on a track and, respectively, outgoing of a train out of the crossing on a track. It is constant on intervals of the form $[x, y)$.
Notation: $\text{Coming}(t, x) =_{df} \text{TrackStatus}(t, x) = coming$,
$\text{Empty}(t, x) =_{df} \text{TrackStatus}(t, x) = empty$.
• Dir: $\mathcal{T} \rightarrow \{open, close\}$ is a function representing the commands of control: to open the gate or to close the gate.
• GateStatus: $\mathcal{T} \rightarrow \{opened, closed, udef\}$ is a function representing the state of the gate: whether it is opened or closed or its status is undefined. This function serves to specify the control.
• InCrossing : $\mathcal{T} \rightarrow \{true, false\}$ is a predicate which expresses the fact that a train is in the crossing (and, thus, the gate must be closed).
• An important technical notion characterizing when the controller may open the gate is Safe To Open that we formulate in a form not equivalent to that of [GH96] which is in some sense more precise with respect to intuitive demands of dependability:

SafeToOpenSp*$(t)=_{df}$
$\forall x\,(\mathrm{Empty}(t,x)\ \vee\ \forall\tau \leq t\,(\forall\tau' \in [\tau,t]\,\mathrm{Coming}(\tau',x) \rightarrow t < \tau + \mathit{WaitTime})$.
The condition SafeToOpenSp*permits to open the gate whenever it is not dangerous.

2.3 Railroad Crossing Problem: Specification of the Environment.

(TrStInit) $\forall x\,\mathrm{Empty}(0,x)$
(At time 0 there are no trains on each track.)
(DirInit) $\mathrm{Dir}(0) = \mathit{open}$
(At the initial moment the signal controlling the gate is *open* .)
(CrCm) $\forall t\,(\mathrm{InCrossing}(t) \rightarrow \exists x\,\exists\tau \leq (t - d_{min})\,\forall\tau' \in [\tau,\ t]\,\mathrm{Coming}(\tau',x))$
(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) $\forall t(\forall\tau \in (t - d_{open},\ t]\mathrm{Dir}(\tau) = \mathit{open} \rightarrow \mathrm{GateStatus}(t) = \mathit{opened})$
(If at time t the command has been *open* for at least a duration d_{open} then the gate is opened at time t.)
(ClsClsd) $\forall t(\forall\tau \in (t - d_{close},\ t]\mathrm{Dir}(\tau) = \mathit{close} \rightarrow \mathrm{GateStatus}(t) = \mathit{closed})$
(If at time t the command has been *close* for at least a duration d_{close} then the gate is closed at time t.)
(dIneq) $d_{close} + d_{open} < d_{min} < d_{max}$
(These are trivial constraints on the durations involved, in particular, the time for closing is smaller than the minimum time of reaching the crossing by any train detected as *coming*.)
We append here a precision on the external functions that looks inessential from "physical" point of view but is indispensable for defining semantics.
(TrStIntervals) $\forall x\,\forall t\,(\mathrm{Empty}(t,x) \rightarrow \mathrm{Empty}(t_{infEmp},x)$,
$\forall x\,\forall t\,(\mathrm{Coming}(t,x) \rightarrow \mathrm{Coming}(t_{infCmg},x)$,
where $t_{infEmp} = \inf\{\tau \leq t :\ \forall\tau' \in [\tau,t]\,\mathrm{Empty}(\tau',x)\}$
and $t_{infCmg} = \inf\{\tau \leq t :\ \forall\tau' \in [\tau,t]\,\mathrm{Coming}(\tau',x)\}$.
(Intervals of the same value of TrackStatus are closed from the left and opened from the right.)
And to facilitate references we formulate the trivial property of the fact that TrackStatus has exactly two values:
(TrStValues) $\forall x\,\forall t\,(\mathrm{Empty}(t,x) \leftrightarrow \neg\mathrm{Coming}(t,x))$
(Absence of coming train means that the track is empty.)

2.4 Railroad Crossing: Specification of the Control.

These specifications concern requirements to the control.
(Safety) $\forall t\,(\mathrm{InCrossing}(t) \rightarrow \mathrm{GateStatus}(t) = \mathit{closed})$
(When a train is in the crossing, the gate is closed).
(Dependability) $\forall t\,(\forall\tau \in [t - d_{open}, t]\,\mathrm{SafeToOpenSp}^*(\tau) \rightarrow \mathrm{GateStatus}(t) = \mathit{opened})$
(If the zone of control is safe to open for a duration of time greater than d_{open} then the gate is open).

3 Algorithm for the Railroad Crossing Controller

We start with a Gurevich machine solution of the the Railroad Crossing Problem. The solution is simular to [GH96], and just makes more precise one detail. This solution is self-explanatory, that is why we do not repeat the basic notions of Gurevich machines that can be found in [GH96] or in more detail in [Gur95].

3.1 Gurevich Machine Solution of Railroad Crossing Problem.

External (input) functions:

- CT the current time;
- *Tracks* is the set of tracks; x, y, ... are variables for tracks (this is not an input but nevertheless it is external, and no algorithm can change it);
- $TrackStatus(x)$: $Tracks \rightarrow \{coming, empty\}$ is an external function representing for every track x the track status.

Internal functions (output or strictly internal):

- *Dir* is the signal to open/close the gate generated by the algorithm; $Dir \in \{open, close\}$;
- $DeadLine$: $Tracks \rightarrow \mathcal{T}_0$ is the first moment of appearance of a train on a given track plus *WaitTime* , and this value is then used to decide on control of the gate, see *SafeToOpen* below;
- Time constants d_{min}, d_{max}, d_{open} and d_{close} are the same as in logical specifications.

Notation: $SafeToOpen^* =_{df} \forall x\,(TrackStatus(x) = empty \vee CT < DeadLine(x))$.

Remark. The corresponding time dependant function for $SafeToOpen^*(x)$ will be SafeToOpen$^*(t, x)$, and we are to prove that this function correctly represents SafeToOpenSp$^*(t, x)$ of logical specifications.

Intuitive assumption on time durations in [GH96] says that

Actions of algorithms are performed instantaneously.

This thesis needs a precision. Such a precision will be done in subsection 3.2, informal discussion concerning many interesting subtleties can be found in [GH96]. An algorithm for the Railroad Crossing Controller in terms of Gurevich machines is given on Fig. 1.

3.2 Semantics of the algorithm.

Clear, that functioning of the algorithm for a given input can be represented as a map from time to its states. As an input the algorithm has a vector function of time $(\text{TrackStatus}(t, x))_{x \in Tracks}$. Its inner state is a vector function $(\text{Dir}(t), (\text{DeadLine}(t, x))_{x \in Tracks})$. To illustrate the problem of interpreting instantaneous actions consider an execution of the operator

if $TrackStatus(x) = empty$ **and** $DeadLine(x) < \infty$
then $DeadLine(x) := \infty$ **endif.**

Assume that at a moment t the **if**-condition is valid. At what moment $DeadLine(x)$ becomes ∞? Clear, not at t otherwise $DeadLine(x)$ will have two different values at the same moment. So, at a moment τ that is greater that t but smaller that any moment to the right of t. There is no such moment among standard reals. Thus, it is reasonable to attribute such an event to some moment t^+ which surpasses t in an infinitesimal. Sure, our construction must be independent of choices of such infinitesimals.

```
var x ranges over Tracks;
Initial values:
        DeadLine(x) := ∞ for all x ∈ Tracks;
        Dir = open;

forall x in parallel repeat
  block
    if TrackStatus(x) = coming and DeadLine(x) = ∞
      then DeadLine(x) := CT + WaitTime
    endif
    if TrackStatus(x) = empty and DeadLine(x) < ∞
      then DeadLine(x) := ∞;
    endif
    if Dir = open and ¬SafeToOpen* then Dir := close endif
    if Dir = close and SafeToOpen* then Dir := open endif
  endblock
```

Fig. 1. Railroad Crossing Controller.

Semantics of Block Algorithms. A traditional way of defining semantics of an algorithm is to look at it as at an appropriate automaton and to define its execution as a map representing the evolution of its states with time. In our case a state is a vector of values of identifiers and that of time which can be considered also as identifier CT. Thus, a *global state* is a vector constituted by values of identifiers from

$$V = \{(TrackStatus(x))_{x \in Tracks}, (DeadLine(x))_{x \in Tracks}, Dir, CT\}.$$

We distinguish *internal* and *external*, or *input* states, namely,

$$V_{Extrn} = \{(TrackStatus(x))_{x \in Tracks}, CT\},$$
$$V_{Intrn} = \{(DeadLine(x))_{x \in Tracks}, Dir\}.$$

For every identifier $v \in V$ there is a predefined range of values $Range_v$. For a set $U \subseteq V$ we denote by S_U the corresponding set of values, i. e. $S_U =_{df} \prod_{u \in U} Range_u$.
A *run* of an algorithm is an operator that for a given input, that is for given external identifiers as functions of time, defines values of internal identifiers also as functions of time, now time is $\mathcal{T}$.
The algorithm under consideration has a block structure (that is a basic construction of Gurevich machines, see [Gur95])

if $Cond_1$ **then** M_1 **endif**
if $Cond_2$ **then** M_2 **endif**
. . . . . .
if $Cond_k$ **then** M_k **endif**

where $Cond_i$ are conditions expressed by quantifier free formulas and M_i are assignments of internal identifiers, and all the **if-then**-operators of the block are executed simultaneously.
Remind that input functions, represented in algorithms by input identifiers, are constant on intervals of the form $[t, t')$, where $t, t' \in \mathcal{T}_0$. Thus, for such a function Z and a moment t when its value becomes new, the property $Z(t^-) \neq Z(t) = Z(t^+)$ holds.

We may consider that for $t < 0$ all the functions have value *udef*.
Let an input $\mathcal{E}$ be given, that is a set of functions of time representing track status for every track. We know that each such a function changes its values in isolated points of $\mathcal{T}_0$.
A *global run* of the algorithm for a given input is a vector function from time $\mathcal{T}$ to the values of identifiers. As the algorithm cannot influence the input, to define a run is to define its restriction to inner identifiers. This restriction will de denoted below by ρ and call *(internal) state trace* or *(internal) run.* The global run under definition will be denoted by $\hat{\rho}(t)$.
Let an input $\mathcal{E}$ be given. It is defined on $\mathcal{T}_0$, but can be trivially extended on $\mathcal{T}$ because it is piecewise constant.
The run ρ for this input is defined recursively, in a natural way.
To start this recursion, note that $\hat{\rho}(0)$ is defined by initial values of the algorithms that are presumed to be given.
Suppose that $\rho(\tau)$ is defined for all $\tau \in [0, t]$, $t \geq 0$.
We assume that the value of ρ does not change while all the conditions remain false.
Let t_C be infimum of $\tau \geq t$ at which at least one of the conditions becomes true.
Extend ρ slightly beyond this moment: $\rho(\tau) = \rho(t)$ for $\tau \in (t, t_C^+)$.
Two cases may appear.
Case 1. There is a condition that is true at t_C. The value $\rho(t_C^+)$ is defined as the result of execution of the assignments corresponding to all the $Cond_i$ that are valid at t_C. Sure, the assignments are taken for the values at t_C and must be consistent, otherwise the run is undefined on $[t_C^+, \infty)$.
Case 2. All the conditions are false at t_C. Then one of them is true at t_C^+ (property (TInf)). Set $\rho(t_C^+) = \rho(t)$.
It is evident that augmenting the time by infinitesimal steps infinitely says that our algorithm has no physical sense. For the algorithm under consideration one infinitesimal augmentation is sufficient, and then we have an advance of time indeed.
One can also remark that for the concrete algorithm under consideration the runs are deterministic.
For our algorithm (as well as for many others) one can represent a global run $\hat{\rho}$ in a unique way as a finite or infinite sequence $\mathcal{R} = I_0, S_0, I_1, S_1, \ldots$, where $I_0, I_1, I_2, \ldots$ is an interval sequence partitioning the time, $\hat{\rho}$ is constant on each interval I_k and has the value S_k.

4 Timed Transition Systems and its Formula Representation.

As we remarked earlier a standard way of presentation of functioning of an algorithm is this or that notion of abstract automaton. For algorithms with time some of their features can be represented as timed automata [AD94] or various hybrid automata, e. g. [ACHH93], etc. We give here a notion to meet the demands of describing the semantics of the algorithm we consider here or intend to consider in the future.

4.1 Timed Transition Systems

Let V be a finite set of function symbols which we will call *identifiers* to refer to its further interpretation. They correspond to the signature of Gurevich machine. The set will be usually represented as a vector.
The set V is partitioned into two (disjoint) subsets V_{Extrn} and V_{Intrn} of external and internal identifiers, the set V_{Extrn} containing a symbol representing (current) time. Below we tacitly assume that whenever given a V, some its partition into internal and external subsets is also given.
As *global states* there will figure vectors representing evaluations of all identifiers from V, i. e. elements of set S_V of the type $\prod_{v \in V} Range_v$. An *internal state* is a vector of type $\prod_{v \in V_{Intrn}} Range_v$, and an *external state* is a vector of type $\prod_{v \in V_{Extrn}} Range_v$. In place of "internal state" we will often use simply "state". The set of internal states will be usually denoted by S or S_{Intrn}, and the set of external states by S_{Extrn}.
For $U \subseteq V$ and $s \in S$ denote by $U[s]$ the restriction (projection) of vector s onto components given by U. Similar notation will be used for sets: $U[E]$, where E is a set, means $\{U[s] : s \in E\}$.
A *timed transition system* is a tuple

$$(V, S_V, \sigma_0, Trans),$$

where
- V is a set of identifiers partitioned into V_{Extrn} and V_{Intrn};
- S_V is the set of global states of the type described above, and consequently, the internal and external states are defined as the corresponding projection sets $S = S_{Intrn}$ and S_{Extrn};
- $\sigma_0 \in S_V$ is the global initial state;
- $Trans \subseteq S_V \times S$ is a set of transitions (note that S_V contains time).

Let $\mathcal{S}$ be a transition system of the form described above.
An *input* is a vector function whose each component corresponding to $v \in V_{Extrn}$ has type

$$\mathcal{T}_0 \times Dom_v \to Range_v.$$

Any input is finally used in all our constructions in the context of properties. We suppose that

All properties we use which involve inputs are piecewise constant.

We assume that each input is extended on $\mathcal{T}$ preserving all the properties we use.
A given input $\mathcal{E}$, such that $V_{Extrn}[\mathcal{E}(0)] = V_{Extrn}[\sigma_0]$, determines runs of the system. A global run is a vector function of time giving for each moment the value of global state. In a run we distinguish *external trace* or simply *input*, defined as above, and *(internal) state trace* composed from the components of the run representing internal identifiers and giving the evolution of internal states in the process of execution of the transition system. We will denote a run as defined below by $\hat{\rho}(t)$, and by $\rho(t)$ its state trace.
To define a *run* for a given input is to define its state trace.
Trace ρ is defined recursively.
$\rho(0) = V_{Intrn}[\sigma_0]$.

Suppose that ρ is defined on $[0,t]$, $t \geq 0$.
Let $\sigma(\tau)$ be the global state composed of $\rho(t)$ and $\mathcal{E}(\tau)$ (the latter contains τ). Let

$$t_0 = \inf\{\tau \geq t \,:\, \exists s \in S\ ((\sigma(\tau), s) \in Trans)\}.$$

If t_0 is undefined (i. e. the defining set is empty) then the trace $\rho(\tau)$ is undefined for $\tau > t$.
Assume that t_0 is defined.
Extend ρ up to t_0^+: $\rho(\tau) = \rho(t)$ for $\tau \in (t, t_0^+)$.
Consider the set $D = \{s \in S \,:\, ((\sigma(t_0), s) \in Trans)\}$. Two cases are possible.
Case 1: $D = \emptyset$. Set $\rho(t_0^+) = \rho(t)$.
Case 2: $D \neq \emptyset$. Choose any $s \in D$, and set $\rho(t_0^+) = s$.
Remark. To model transitions with time delay it is sufficient to consider transitions as a subset of $S_V \times \boldsymbol{R}_{\geq 0} \times S$, and to adapt the definition of run.

4.2 Formula Timed Transition System.

We can effectively treat only finitely represented transition systems. To arrive at such a notion we are to coarse the states into finite number of sets (see e. g. [ACHH93]). Rather a general way of such representation is representation in terms of logic formulas.
To construct formulas we use variables for elements of S_V and S and the notation for projections introduced above. A *formula timed transition system* is a tuple

$$(V, S_V, \sigma_0, Q, q_0, \psi),$$

where
- V, S_V, σ_0 are as in timed transition systems above;
- Q is a finite set of *formula states*, briefly *F-states* each one being a formula of the form $q(s)$ where list of variables s consists of variables of all types $S_{V_{Intrn}}$ (thus, each formula represents a set of internal states);
- $q_0 \in Q$ is the initial F-state, and it satisfies $q_0(V_{Intrn}[\sigma_0])$;
- ψ gives for each pair (p,q) of F-states a finite set $\psi(p,q)$ of formulas of the form $F(\sigma, s)$, where σ is of type S_V and s is of type S (the variables of a same type are different in σ and s), such that whatever be $F \in \psi(p,q)$, a global state σ and an internal state s,

$$F(\sigma, s) \rightarrow (p(V_{Intrn}[\sigma]) \wedge q(s)),$$

that is ψ respects F-states.
We define the set of transitions $Trans$ of the formula transition system as

$$\{(\sigma, s) : \bigvee_{p,q \in Q} \bigvee_{F \in \psi(p,q)} F(\sigma, s)\}.$$

For the Railroad crossing Controller we take as the set of identifiers the identifiers of the algorithm:

$$V = \{(TrackStatus(x), DeadLine(x))_{x \in Tracks}, Dir\}$$

with $V_{Extrn} = \{(TrackStatus(x))_{x \in Tracks}\}$.
The set of basic states is then:

$$S_V = (\{coming, empty\} \times \mathcal{T})^{|Tracks|} \times \{open, close\}.$$

Construct a set Q of F-states. For each track x let:
$q_x(s) =_{df} (DeadLine(x)[s] = \infty)$ and $\bar{q}_x(s) =_{df} (DeadLine(x)[s] < \infty)$.
Then define $d(s) =_{df} (Dir[s] = open)$ and $\bar{d} =_{df} (Dir[s] = close)$.

The set Q of F-states of the system is the set of all formulas:

$$(\bigwedge_{x \in Tracks} \xi_x \wedge \delta), \text{ where } \xi_x \in \{q_x, \bar{q}_x\} \text{ and } \delta \in \{d, \bar{d}\}.$$

The initial F-state is the state $(\bigwedge_{x \in Tracks} q_x(s) \wedge d(s))$.
It is easy to write formula transitions in the succinct form we discussed above. Syntactically they almost repeat the description of the algorithm:

(R1) $((TrackStatus(x)[\sigma] = coming \wedge DeadLine(x)[\sigma] = \infty) \rightarrow DeadLine(x)[s] = CT[\sigma] + WaitTime)$,
(R2) $((TrackStatus(x)[\sigma] = empty \wedge DeadLine(x)[\sigma] < \infty) \rightarrow DeadLine(x)[s] = \infty)$,
(R3) $((Dir[s] = open \wedge \neg SafeToOpen^*[\sigma]) \rightarrow Dir[s] = close)$,
(R4) $((Dir[\sigma] = close \wedge SafeToOpen^*[\sigma]) \rightarrow Dir[s] = open)$,
where $SafeToOpen^*[\sigma]$ is $SafeToOpen^*$ with each identifier v replaced by $v[\sigma]$.
This representation can be easily timed explicitely, that gives the following logical description of runs used in model checking proof:

$$((\text{DeadLine}(t, x) = \infty \wedge \text{Coming}(t, x)) \rightarrow \text{DeadLine}(t^+, x) = t + WaitTime) \quad (1)$$

$$((\text{DeadLine}(t, x) < \infty \wedge \text{Empty}(t, x)) \rightarrow \text{DeadLine}(t^+, x) = \infty) \quad (2)$$

$$((\text{Dir}(t) = open \wedge \neg\text{SafeToOpen}^*(t)) \rightarrow \text{Dir}(t^+) = close) \quad (3)$$

$$((\text{Dir}(t) = close \wedge \text{SafeToOpen}^*(t)) \rightarrow \text{Dir}(t^+) = open) \quad (4)$$

Sure we must add to these formulas obvious default conventions.

Proposition 1 *The formula transition system for the Railroad Crossing Controller defines the same run as the semantics of block algorithms, and the same run as the formulas given above.*

On Model Checking Proof.

Theorem 1 *The Railroad Crossing Algorithm satisfies* (Safety) *and* (Dependability) *properties.*

The proof of theorem 1 [BS96] shows that the only non trivial inference search rule is to take inf when eliminating positive quantifiers. I. e. if we use a premise $\exists t \Phi(t, X)$ we take $t_0 = \inf\{t : \Phi(t, X)\}$, and get information on the behavior at t_0 and t_0^-.
One general observation concerns the fact that the system under consideration is finite memory in the following sense: there is a constant C such that if there exists a counter-model for the verification problem then its complexity can be bounded by C. Such a property permits to reduce the problem to theory of real addition.

References

[ACHH93] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Workshop on Theory of Hybrid Systems, 1992*, pages 209–229. Springer Verlag, 1993. Lect. Notes in Comput. Sci, vol. 736.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BS96] D. Beauquier and A. Slissenko. The railroad crossing problem: Towards semantics of timed algorithms and their model checking in high level languages. *TR-96-10, Dept. of Informatics, Univ. Paris-12*, 24p., 1996.

[Eme90] A. Emerson. Temporal and model logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Vol. B: Formal Models and Sematics*, pages 995–1072. Elsevier Science Publishers B.V., 1990.

[Eme96] A. Emerson. Automated temporal reasoning about reactive systems. In F. Moller and G. Birtwistle, editors, *Logic for Concurrency. Structure versus Automata*, pages 41–101. Springer-Verlag, 1996. Series: "Lecture notes in Computer Science (Tutorial)", Vol. 1043.

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

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

[Han94] H. A. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Elsevier, 1994. Series: "Real Time Safety Critical System", vol. 1. H. Zedan, Series Ed.

[Mil90] R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Vol. B: Formal Models and Sematics*, pages 1201–1242. Elsevier Science Publishers B.V., 1990.

[MP92] Z. Manna and A. Pnueli. *Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer Verlag, 1992.