Specifying Real-Time Finite-State Systems in Linear Logic

Max I. Kanovich* Mitsuhiro Okada† Andre Scedrov‡

Abstract

Real-time finite-state systems may be specified in linear logic by means of linear implications between conjunctions of fixed finite length. In this setting, where time is treated as a dense linear ordering, safety properties may be expressed as certain provability problems. These provability problems are shown to be in PSPACE. They are solvable, with some guidance, by finite proof search in concurrent logic programming environments based on linear logic and acting as sort of model-checkers. One advantage of our approach is that either it provides unsafe runs or it actually establishes safety.

1 Introduction

There are a number of formalisms for expressing real-time processes, including [1, 6, 7, 3, 4, 5, 50, 44, 45, 38]. Many of these real-time formalisms are based on temporal logic or its variations [46, 38, 33] or on timed process algebras [14, 42, 43, 23, 12], or on Büchi automata [52, 3]. In some cases exact complexity-theoretic information is available, such as [51, 3, 5], while other formalisms are known to be undecidable. In this context, undecidability may arise *a priori* from the undecidability of traditional predicate logic with binary predicates, or in a more subtle way from so-called punctual temporal specifications, which are known to be capable of simulating the halting problem [5].

In this work we introduce a real-time specification formalism based on *linear logic* [19, 20, 48, 49]. A clear advantage of our approach is that it provides a common user with a very easy and transparent way of writing high-level specifications without having to be concerned with operational issues. Linear logic seems a natural choice for a *logical* specification formalism in this regard because of its intrinsic ability to reflect state transitions. Indeed, the most straightforward and naive way of writing very simple propositional logic formulas that correspond to the informal natural language descriptions of state transition systems is actually rigorous and correct in linear logic, while this way of writing specifications is incorrect in classical logic. This is discussed in detail below and in the railroad-crossing example in Section 2.

*max@kanovich.dnttm.rssi.ru Department of Theoretical and Applied Linguistics, Russian State University for the Humanities, Miusskaya 6, 125267 Moscow, Russia.

†mitsu@abelard.flet.mita.keio.ac.jp Department of Philosophy, Keio University, 2-15-45 Mita, Minato-ku, Tokyo 108, Japan. Partially supported by the Grant-in-Aid for Scientific Research (Ministry of Education, Science and Culture, Japan) and the Ogata-Jyosei Grant (Keio University).

‡scedrov@cis.upenn.edu http://www.cis.upenn.edu/~scedrov Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104-6395 USA. Partially supported by DoD MURI "Semantic Consistency in Information Exchange" as ONR Grant N00014-97-1-0505, by NSF Grant CCR-9800785, and by an International Fellowship from the Japan Society for the Promotion of Science.

The best way to model, specify, and prove time-sensitive properties of real-time systems would be to use natural language. While this might be possible in the future, today it is customary to resort to various formal languages for this purpose. Among these, the formal language that has been most investigated and best understood is traditional predicate logic. In principle one could express various properties and requirements of real-time systems by means of formulas built up from certain basic, or atomic, predicates by traditional logical connectives and quantifiers. However, such a general approach in the framework of traditional predicate logic runs into difficulties, for example, the undecidability of predicate logic with binary predicates.

For purely *qualitative* time properties of real-time systems such as *sometimes, always, never*, it suffices to consider "time-closed" formulas where all time variables are bound by quantifiers. Such qualitative time properties can be handled within the *temporal logic* framework where all time variables are encapsulated by means of temporal-modal operators on the propositional level. There are a number of successful investigations in this line of research, for instance [46, 51]. However, one runs into difficulties with this approach in handling *quantitative* time properties such as *"within B time units afterwards", "never for more than B time units",* that refer to *explicit* time delays. In the case of such *quantitative* time constraints, in order to represent current states of a given real-time system temporal logic is to be equipped with first-order means so that *time* parameters cannot be handled but *explicitly,* beyond the propositional logical framework. There are other solutions, such as the *temporal logic of actions* TLA [33, 1], where real time is handled *explicitly* by introducing a variable to represent time. However, it is not easy to describe a decidable fragment of TLA suited for describing system requirements such as *safety* and *liveness.*

In this paper we introduce an approach that allows handling both qualitative and quantitative time aspects of real-time systems in purely logical terms, where the difficulties of being over-sophisticated and over-complicated are obviated within the framework of *monadic Horn fragment* of linear logic in the sense of [28]. This simple fragment of linear logic can be communicated to the common users without requiring any sophistication in logic. Let us describe the main idea of our approach. For real-time systems with their *peculiar* `time`, one of the basic primitive relations one deals with is of the form

$$P(e,t) \equiv \text{“an event } e \text{ happens in the system at moment } t\text{”}.$$

In order to circumvent the difficulties caused by binary predicates (which, for example, usually lead to undecidability of the system), one may split $P(e,t)$ into two *unary* predicates: a "timeless predicate" $\mathtt{Q}(e)$ that means "event e happens in the system" and the unique "time predicate" $\mathtt{Time}(t)$ that means "`time` is t (on the global clock)". That is,

$$P(e,t) \approx (\mathtt{Q(e)}\ \mathbf{and}\ \mathtt{Time(t)}).$$

Suppose that a given action is performed in such a way that a certain event e_1 at moment t_1 is followed by another event e_2 at moment t_2 (as a delayed effect). A naive way of formalizing this action is by a "Horn axiom" of the form $P(e_1,t_1)$ **implies** $P(e_2,t_2)$. Following our unarization procedure, this axiom is supposed to be encoded as:

$$(\mathtt{Q(e_1)}\ \mathbf{and}\ \mathtt{Time(t_1)})\ \mathbf{implies}\ (\mathtt{Q(e_2)}\ \mathbf{and}\ \mathtt{Time(t_2)}).$$

However, such a straightforward reduction to unary predicates requires certain precautions related to the exact meaning of the connectives **and** and **implies**. In particular, the traditional understanding of **and** and **implies** as boolean connectives $\wedge$ and $\Rightarrow$, respectively, yields unintended consequences such as

$$(\mathtt{Q(e_1)} \wedge \mathtt{Time(t_1)}) \Rightarrow (\mathtt{Q(e_1)} \wedge \mathtt{Q(e_2)} \wedge \mathtt{Time(t_1)} \wedge \mathtt{Time(t_2)}),$$

that is, $P(e_1, t_1) \Rightarrow (P(e_1, t_1) \wedge P(e_2, t_2))$, and furthermore

$$(\mathtt{Q}(\mathtt{e}_1) \wedge \mathtt{Time}(\mathtt{t}_1)) \Rightarrow (\mathtt{Q}(\mathtt{e}_1) \wedge \mathtt{Time}(\mathtt{t}_2)),$$

that is, $P(e_1, t_1) \Rightarrow P(e_1, t_2)$.

The main reason behind these problems is that our **and** is intended to represent only the *"concurrent coexistence"*, while the traditional conjunction $\wedge$ can behave in many different ways. This is one reason why as a possible logical setting we propose *linear logic*, a resource-sensitive refinement of classical logic [19, 20, 48, 49], where the traditional conjunction $\wedge$ is split into two connectives: $\otimes$ (tensor) and & (with), and the traditional implication $\Rightarrow$ is refined as *linear implication* $\multimap$. Revealing the *"concurrent coexistence"* nature of $\otimes$, we encode the basic binary relation $P(e, t)$ by a linear logic formula of the form $\mathtt{Q}(\mathtt{e}) \otimes \mathtt{Time}(\mathtt{t})$. Accordingly, the action discussed above will be specified by a linear logic implication of the form

$$(\mathtt{Q}(\mathtt{e}_1) \otimes \mathtt{Time}(\mathtt{t}_1)) \multimap (\mathtt{Q}(\mathtt{e}_2) \otimes \mathtt{Time}(\mathtt{t}_2)).$$

It is remarkable that in linear logic this formula does not yield the undesired formula

$$(\mathtt{Q}(\mathtt{e}_1) \otimes \mathtt{Time}(\mathtt{t}_1)) \multimap (\mathtt{Q}(\mathtt{e}_1) \otimes \mathtt{Q}(\mathtt{e}_2) \otimes \mathtt{Time}(\mathtt{t}_1) \otimes \mathtt{Time}(\mathtt{t}_2)),$$

nor the undesired formula

$$(\mathtt{Q}(\mathtt{e}_1) \otimes \mathtt{Time}(\mathtt{t}_1)) \multimap (\mathtt{Q}(\mathtt{e}_1) \otimes \mathtt{Time}(\mathtt{t}_2)).$$

Concrete examples of this phenomenon are discussed in the railroad-crossing example in Section 2.

Important system properties such as *safety* are represented in our approach as certain PSPACE decision properties related to *provability* in linear logic. In terms of complexity, this indicates a good fit with the automata-theoretic approach [3, 4] and its PSPACE-complete problem of emptiness of the language associated to an automaton, in contrast with the EXPSPACE-complete properties related to *satisfiability* in metric interval temporal logic [5]. By way of comparison between our setting and the automata-theoretic approach, let us emphasize that one of the central concepts used in verification is *reachability*, in the sense that safety is seen as unreachability. Our approach provides a simple and direct correspondence between reachability and the traditional logical concept of *provability.* In contrast, the traditional concept emphasized in the automata-theoretic approach is the language emptiness problem, while reachability is treated there only as a derived, subsidiary notion. Moreover, the way reachability is derived from language emptiness in the automata-theoretic approach involves nontrivial technical operations such as language intersection and complementation. The exact nature of a relationship between our approach and the automata-theoretic approach remains to be determined.

Let us note that the method of proof of our main complexity result shows that, aside from complexity bounds, decision problems that involve temporal constraints may be dealt with by running a *finite proof search*, with some guidance, in the available concurrent logic programming environments based on linear logic [9, 10, 24, 25, 41, 30, 31, 32, 13] or in the environments supporting multiset rewriting [17, 22] or concurrent rewriting [40, 44], either of which would in this case act as sort of model-checkers. Indeed, our current work may be seen as a first step toward a larger issue of *proof-based state exploration* in contrast to model-checking, which is model-based. One advantage of our approach is that it incorporates a decision procedure, so that either it provides unsafe runs or it actually establishes safety.

Technically, our framework may be seen as a combination of *local* transitions and *global*, quantitative time correlations. In our framework transitions are instantaneous but events may have duration.

Regarding the transitions, our framework is a refinement of the work in [11, 21, 18, 39, 15], which established a direct relationship between Petri nets and linear logic axiomatizations using conjunctive formulas. Here we consider only conjunctions of fixed finite length. In linear logic this restriction suffices for a faithful simulation of finite state transitions.

We extend this underlying framework to real-time systems by using global constraints formulated by means of alarms (timers, time guards.) Our use of these devices is generally motivated by "an old-fashioned approach" in [1], although our actual technical treatment of the timers is somewhat different. Let us illustrate our combination of local transitions and global time constraints in more detail on the standard railroad-crossing example.

2 Example: Railroad-Crossing Controller

The railroad-crossing system we consider consists of a train, a signal, and a gate. The train goes from being `safe` to `approaching`, then to `crossing`, and then back to being `safe`. The signal may be set to either `raise` or `lower`. The gate has four options: `up`, `down`, `moving_up`, or `moving_down`.

The controller senses when the train starts `approaching` and sets the signal to `lower` within D time units. When the signal is set to `lower`, then the gate starts `moving_down` within G time units. Once the gate starts `moving_down`, it is `down` within L time units. When the train is `safe`, the signal is set to `raise`, and in turn, the gate starts `moving_up`, and is then `up`. For the purposes of this simple example, no time bounds are placed on this suite. In addition to that, the train is supposed to spend at least B time units going from `safe` to `crossing`.

The main *safety property* of the system is that when the train is `crossing`, then the gate is `down`. A common assumption is that $B > D + G + L$.

2.1 Specifying Timeless Transitions

In order to let the reader develop some feel for the way linear logic operates, let us first discuss in detail linear logic specifications of timeless transitions. Because of the resource-sensitive nature of linear logic, the most naive way of writing logical formulas corresponding to the informal English description above is actually rigorous and correct. For instance, changing the signal from `lower` to `raise` when the train is `safe` is specified by the formula:

$$(\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)}) \multimap (\mathtt{Tr(safe)} \otimes \mathtt{Sig(raise)}), \tag{1}$$

where $\otimes$ (tensor) is a linear logic version of conjunction and $\multimap$ is *linear implication.* The meaning of a linear implication $A \multimap B$ is not simply that A implies B but that A is *consumed* or *spent* and that B is produced. This is reflected in the linear logic rules of inference. For instance, a linear logic counterpart $A \multimap (A \otimes A)$ of the traditional propositional tautology $A \Rightarrow (A \wedge A)$ is not provable in linear logic. This expressive ability of linear logic to distinguish between *one* and *two* occurrences of a formula reflects the common sense that \$1 cannot be spent to produce a \$1 and another \$1. In our situation this expressive ability makes it possible for the linear logic specification (1) to stipulate that the signal *changes* from `lower` to `raise`. A similar formula in classical or in intuitionistic logic

$$(\mathtt{Tr(safe)} \wedge \mathtt{Sig(low)}) \Rightarrow (\mathtt{Tr(safe)} \wedge \mathtt{Sig(raise)})$$

is incorrect in this regard, because the tautology $A \Rightarrow (A \wedge A)$ allows us to infer

$$(\mathtt{Tr(safe)} \wedge \mathtt{Sig(low)}) \Rightarrow (\mathtt{Tr(safe)} \wedge \mathtt{Sig(low)} \wedge \mathtt{Sig(raise)}),$$

which does not correspond to reality, because the signal cannot be both `lower` and `raise` at once. It is for this reason that we choose linear logic. This logic allows us to represent configurations of a finite-state system in a simple-minded way as conjunctions of fixed finite length.

Let us also mention that simply writing $\mathtt{Sig(low)} \multimap \mathtt{Sig(raise)}$ is incorrect in linear logic for the same reason that $\mathtt{Sig(low)} \Rightarrow \mathtt{Sig(raise)}$ is incorrect in classical or in intuitionistic logic, namely the signal is supposed to be changed when the train is `safe`, not unconditionally. Furthermore, writing $(\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)}) \multimap \mathtt{Sig(raise)}$ is also incorrect because it stipulates that the train somehow ceases being `safe` as the signal is changed. In linear logic this formula is not equivalent to the correct specification mentioned above. Note that in classical or in intuitionistic logic, the analog $(\mathtt{Tr(safe)} \wedge \mathtt{Sig(low)}) \Rightarrow \mathtt{Sig(raise)}$ is equivalent to the formula $(\mathtt{Tr(safe)} \wedge \mathtt{Sig(low)}) \Rightarrow (\mathtt{Tr(safe)} \wedge \mathtt{Sig(raise)})$ discussed above.

Raising the gate is specified in a similar way by the following three formulas:

$$(\mathtt{Sig(raise)} \otimes \mathtt{Gate(moving\_down)}) \multimap (\mathtt{Sig(raise)} \otimes \mathtt{Gate(moving\_up)}), \tag{2}$$

$$(\mathtt{Sig(raise)} \otimes \mathtt{Gate(down)}) \multimap (\mathtt{Sig(raise)} \otimes \mathtt{Gate(moving\_up)}), \tag{3}$$

$$\mathtt{Gate(moving\_up)} \multimap \mathtt{Gate(up)}. \tag{4}$$

Let $\mathcal{U}$ be the set consisting of the formulas (1), (2), (3), and (4). As a first exercise, it can be readily shown that the formula
$(\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)} \otimes \mathtt{Gate(down)}) \multimap (\mathtt{Tr(safe)} \otimes \mathtt{Sig(raise)} \otimes \mathtt{Gate(up)})$
is provable in linear logic from the axioms $\mathcal{U}$, while the formula
$(\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)} \otimes \mathtt{Gate(down)}) \multimap (\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)} \otimes \mathtt{Gate(up)})$
is not provable in linear logic from the axioms $\mathcal{U}$. That is, in the available linear logic programming environments such as [9, 10, 24, 25, 41, 30, 31, 32, 13], given $\mathcal{U}$ and an *initial condition* $\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)} \otimes \mathtt{Gate(down)}$, the query on the goal $\mathtt{Tr(safe)} \otimes \mathtt{Sig(raise)} \otimes \mathtt{Gate(up)}$ will be answered positively, while the query on the goal $\mathtt{Tr(safe)} \otimes \mathtt{Sig(low)} \otimes \mathtt{Gate(up)}$ will be answered negatively.

2.2 Specifying Timed Transitions

We generally adopt "an old-fashioned recipe" from [1] but with some modifications, which are discussed below. *Time* is represented globally as a variable ranging over a dense linear ordering $\mathcal{R}$ that includes the natural numbers, for instance the non-negative reals or the non-negative rationals. On the other hand, timing *conditions* are expressed by means of mutually independent *alarms* or *alarms*. Intuitively, perhaps these are best seen as analogous to the alarm mechanism in an alarm-clock, not to the clock itself. Our alarm may be either *off* or it may be set to some finite time value. Note that the value of a alarm may be a time value, which is necessarily finite, or the value *off*, which may be technically represented as $+\infty$. Each timing condition is expressed by its own alarm that is distinct from other alarms.

Let us consider some aspects of the railroad-crossing example in order to see how alarms may be used. Initially, all alarms are off. We first describe an upper-bound alarm. Recall that when the train starts `approaching` the signal is set to `lower` within D time units. Say the train starts `approaching` at time $\mathtt{t_0}$. The alarm $\mathtt{Hi_D}$ is set to $\mathtt{t_0 + D}$, in expectation of the signal to be `lower` in the future. The information that the alarm $\mathtt{Hi_D}$ is on will be used as a precondition for setting the signal to `lower`. When that transition is completed, then the alarm $\mathtt{Hi_D}$ will be turned off (*i.e.,* set to ∞.) We also require that time t progresses below the value of $\mathtt{Hi_D}$. In this way, the signal must be set to `lower` within D time units.

Let us now consider a lower-bound alarm. The assumption that the train spends more than B time units going from `safe` to `crossing` may be described as follows. If the train starts `approaching` at time t_0, set the alarm Lo_B to $t_0 + B$. Then the condition that $t \geq t_0 + B$ is included into the preconditions for the train changing from `approaching` to `crossing`. The alarm Lo_B is turned off when the train is `crossing`.

Given this behavior of the alarms, the timed transitions of the railroad-crossing controller may be specified in linear logic in the following way, where $q_0 \in \mathcal{R}$, where the variables s and t range over $\mathcal{R}$, the variables r, x, y, and z range over $\mathcal{R}_\infty = \mathcal{R} \cup \{+\infty\}$. Let a be either `raise` or `low` and let b be either `moving_up` or `up` or `moving_down` or `down`. Let u be a variable ranging over the two states of the signal, `raise` or `low`, and let v be a variable ranging over the four states of the gate, `moving_up` or `up` or `moving_down` or `down`. An initial condition may be described as:

$$\mathtt{Tr(safe)} \otimes \mathtt{Sig(a)} \otimes \mathtt{Gate(b)} \otimes \mathtt{Time(q_0)} \otimes \mathtt{Hi_D}(\infty) \otimes \mathtt{Hi_G}(\infty) \otimes \mathtt{Hi_L}(\infty) \otimes \mathtt{Lo_B}(\infty).$$

The transitions are described by the linear implications:

$$(\mathtt{Tr(safe)} \otimes \mathtt{Hi_D}(\infty) \otimes \mathtt{Lo_B}(\infty) \otimes \mathtt{Time(t)}) \multimap$$
$$(\mathtt{Tr(appr)} \otimes \mathtt{Hi_D(t+D)} \otimes \mathtt{Lo_B(t+B)} \otimes \mathtt{Time(t)}),$$

$$(\mathtt{Tr(appr)} \otimes \mathtt{Lo_B(r)} \otimes \mathtt{Time(t)} \otimes (\mathtt{r} \leq \mathtt{t})) \multimap (\mathtt{Tr(cross)} \otimes \mathtt{Lo_B}(\infty) \otimes \mathtt{Time(t)})\ ,$$

$$\mathtt{Tr(cross)} \multimap \mathtt{Tr(safe)},$$

$$(\mathtt{Sig(u)} \otimes \mathtt{Hi_D(x)} \otimes \mathtt{Hi_G}(\infty) \otimes \mathtt{Time(t)} \otimes (\mathtt{x} < \infty)) \multimap$$
$$(\mathtt{Sig(low)} \otimes \mathtt{Hi_D}(\infty) \otimes \mathtt{Hi_G(t+G)} \otimes \mathtt{Time(t)}),$$

$$(\mathtt{Gate(v)} \otimes \mathtt{Hi_G(y)} \otimes \mathtt{Hi_L}(\infty) \otimes \mathtt{Time(t)} \otimes (\mathtt{y} < \infty)) \multimap$$
$$(\mathtt{Gate(moving\_down)} \otimes \mathtt{Hi_G}(\infty) \otimes \mathtt{Hi_L(t+L)} \otimes \mathtt{Time(t)}),$$

$$(\mathtt{Gate(moving\_down)} \otimes \mathtt{Hi_L(z)} \otimes (\mathtt{z} < \infty)) \multimap (\mathtt{Gate(down)} \otimes \mathtt{Hi_L}(\infty)).$$

The first three linear implications represent the transitions of the train and the control of the associated alarms. The first formula specifies the train transition from `safe` to `approaching`. Note that this is the only way to turn on the alarms $\mathtt{Hi_D}$ and $\mathtt{Lo_B}$. The second formula describes the train transition from `approaching` to `crossing`, and it involves the condition associated with the lower-bound train alarm. The binary predicate "$\leq$" represents the usual weak ordering on the reals extended with $+\infty$, and is treated here as in classical logic (see Section 3.) The fourth formula governs the setting of the signal to `lower`. Note that we include a precondition for this transition that the associated upper-bound alarm $\mathtt{Hi_D}$ is on, and in this way we depart from the way upper-bound alarms are used in [1].

When this transition is completed, this alarm is turned off. The fifth and sixth formulas similarly specify the behavior of the gate.

As in [1], we also specify the progress of time:

$$(\mathtt{Time(s)} \otimes \mathtt{Hi_D(x)} \otimes \mathtt{Hi_G(y)} \otimes \mathtt{Hi_L(z)} \otimes (\mathtt{s} < \mathtt{t}) \otimes (\mathtt{t} \leq \mathtt{x}) \otimes (\mathtt{t} \leq \mathtt{y}) \otimes (\mathtt{t} \leq \mathtt{z})) \multimap$$

$$(\mathtt{Time(t)} \otimes \mathtt{Hi_D(x)} \otimes \mathtt{Hi_G(y)} \otimes \mathtt{Hi_L(z)}).$$

This is the most complicated kind of specification in our approach. It expresses the requirement that as time progresses, the current time never exceeds the value of any upper-bound alarm that is turned on. No such condition is necessary for lower-bound alarms. Note that the *global* nature of time is expressed by a single formula that indicates the relationship of time to *all* upper-bound alarms at once, instead of having several formulas, each indicating a relationship of time to each upper-bound alarm, one by one. The latter would be incorrect because some alarms might get turned off by actions involving other alarms. Let us also note that in our particular example we chose to interpret all upper-bound alarms as weak upper bounds. If, for instance, the alarm $\mathtt{Hi_D}$ is to be understood as imposing a strict upper bound, then in the formula just above we would write $t < x$ instead of $t \leq x$, and similarly for other upper-bound alarms.

Let $\mathcal{V}$ be the set consisting of the untimed axioms in $\mathcal{U}$ together with all the linear implications just listed. Let *Init* be the formula describing the initial condition where the train is `safe` and all alarms are off.

The main *safety property* may be expressed in our setting as follows:

In the case where $B \leq D + G + L$, for some b that is *not* `down`, from the axioms $\mathcal{V}$ we can prove an "unsafety formula" of the form

$$Init \multimap (\mathtt{Tr(cross)} \otimes \mathtt{Sig(a)} \otimes \mathtt{Gate(b)} \otimes \mathtt{Time(q_1)} \otimes \mathtt{Hi_D(p_1)} \otimes \mathtt{Hi_G(p_2)} \otimes \mathtt{Hi_L(p_3)} \otimes \mathtt{Lo_B(p_4)}),$$

for some $q_1 \in \mathcal{R}$, and some $p_i \in \mathcal{R}_\infty$, $i = 1, 2, 3, 4$. Based on this proof, we can construct a sequence of events leading to such undesired configurations where the train is `crossing` and the gate is *not* `down`.

Whereas, in the case where $B > D + G + L$, for any $q_1 \in \mathcal{R}$, and any $p_i \in \mathcal{R}_\infty$, $i = 1, 2, 3, 4$, it is impossible to prove a formula of the form

$$Init \multimap (\mathtt{Tr(cross)} \otimes \mathtt{Sig(a)} \otimes \mathtt{Gate(b)} \otimes \mathtt{Time(q_1)} \otimes \mathtt{Hi_D(p_1)} \otimes \mathtt{Hi_G(p_2)} \otimes \mathtt{Hi_L(p_3)} \otimes \mathtt{Lo_B(p_4)}),$$

where b is not `down`. This may be interpreted as: *The situation when the train is* `crossing` *and the gate is not* `down`, *is impossible.* Note that the boundary case $B = D + G + L$ would have gone this way instead of the other way had we interpreted all upper-bound alarms as imposing strict bounds, that is, had we written the time-progress formula with all inequalities strict.

Our main result implies that this property is decidable in PSPACE in the total size of the formulas in $\mathcal{V}$ and the size of the unsafety formula. Furthermore, the proof of our main result shows that this parameterized provability problem may be resolved by running a *finite* proof search, with some guidance, on a related query in the available automated linear logic environments [9, 10, 24, 25, 41, 30, 31, 32, 13] or in environments supporting multiset rewriting [17, 22].

3 Syntax

For each $i = 1, \ldots, N$ let $\mathtt{P_i}$ be a unary predicate over a finite domain E_i, representing system states. We assume that the E_i's are mutually disjoint. Each element of E_i is represented by a separate constant. We also consider a unary predicate "$\texttt{Time}$" over a dense linear ordering $\mathcal{R}$ that includes the natural numbers, for instance the non-negative reals or the non-negative rationals. Alarms are represented as unary predicates $\texttt{Hi}_1, \ldots, \texttt{Hi}_\texttt{K}, \texttt{Lo}_1, \ldots, \texttt{Lo}_\texttt{M}$ over $\mathcal{R}_\infty = \mathcal{R} \cup \{+\infty\}$, where $M, K \geq 0$. The associated bounds are represented as *non-negative rational* constants $\texttt{hi}_1, \ldots, \texttt{hi}_\texttt{K}, \texttt{lo}_1, \ldots, \texttt{lo}_\texttt{M} \in \mathcal{R}$. We also consider a binary predicates "$<$" and "$\leq$" over $\mathcal{R}_\infty$ that reflects the usual strict and weak ordering. Each element of $\mathcal{R}_\infty$ is represented by a separate constant. $\texttt{Time}$ is a unary predicate over $\mathcal{R}$ that indicates current time.

The logical context is the so-called multiplicative fragment of quantifier-free intuitionistic linear logic [19, 20, 34, 28, 37]. (For the sake of completeness, the logical rules of inference may also be found in the Appendix.) Non-logical axioms are all true instances of $\texttt{p} < \texttt{q}$ and $\texttt{p} \leq \texttt{q}$, where $\texttt{p}, \texttt{q} \in \mathcal{R}_\infty$, as well as the axioms indicating the traditional (*i.e.,* non-linear) use of the predicates "$<$" and "$\leq$". These are $(\texttt{x} < \texttt{y}) \multimap \texttt{1}$ and $(\texttt{x} \leq \texttt{y}) \multimap \texttt{1}$, where 1 is the propositional constant *"true"* and $\texttt{x}, \texttt{y}$ are variables ranging over $\mathcal{R}_\infty$, (Another, equivalent formulation of these non-logical axioms is in the Appendix.)

Specifications include the formula describing the progress of time:

$$(\texttt{Time(s)} \otimes \texttt{Hi(x)} \otimes (\texttt{s} < \texttt{t}) \otimes \texttt{Ubound(t,x)}) \multimap (\texttt{Time(t)} \otimes \texttt{Hi(x)}),$$

where $\texttt{Hi(x)}$ denotes $\texttt{Hi}_1(\texttt{x}_1) \otimes \ldots \otimes \texttt{Hi}_\texttt{K}(\texttt{x}_\texttt{K})$ and $\texttt{Ubound(t,x)}$ denotes tensor product of formulas of the form $(\texttt{t} < \texttt{x}_\texttt{k})$ or $(\texttt{t} \leq \texttt{x}_\texttt{k})$ for each $1 \leq k \leq K$, where $s, t, x_1, \ldots, x_K$ are syntactically distinct individual variables. Note that as in the railroad-crossing example, this formula also involves all upper-bound alarms but no lower-bound alarms. Specifications may also include finitely many timeless changes, *i.e.,* formulas of the form $\texttt{P(a)} \multimap \texttt{P(b)}$, where $\texttt{P(a)}$ is any chosen formula of the form $\texttt{P}_{\texttt{i}_1}(\texttt{a}_{\texttt{i}_1}) \otimes \ldots \otimes \texttt{P}_{\texttt{i}_\texttt{n}}(\texttt{a}_{\texttt{i}_\texttt{n}})$, $1 \leq n \leq N$, the indices are mutually distinct and they indicate some numbers between 1 and N. Each a_{i_j} is a constant indicating an element of E_{i_j}, and $\texttt{P(b)}$ is the same formula with b_{i_j} instead of a_{i_j}. We also allow more compact expressions in which some constants a_{i_j} are replaced by *variables* u_{i_j} ranging over E_{i_j}. Specifications may also include finitely many timed changes *i.e.,* formulas of the form

$$(\texttt{P(a)} \otimes \texttt{Time(t)} \otimes \texttt{SHi(x)} \otimes \texttt{SLo(z)} \otimes \texttt{RHi(y)} \otimes \texttt{Less}(\texttt{y}, \infty) \otimes \texttt{RLo(r)} \otimes \texttt{Lbound(r,t)}) \multimap$$

$$(\texttt{P(b)} \otimes \texttt{Time(t)} \otimes \texttt{SHi}(\texttt{x}|(\texttt{t}+\texttt{hi})) \otimes \texttt{SLo}(\texttt{t}+\texttt{lo}) \otimes \texttt{RHi}(\infty) \otimes \texttt{RLo}(\infty)),$$

that indicates state changes, upper-bound alarms to be set, lower-bound alarms to be set, *other* upper-bound alarms to be released, *i.e.,* turned off, as well as *other* lower-bound alarms to be conditionally released, together with the corresponding conditions. $\texttt{SLo(z)}$ indicates the lower-bound alarms to be set. More precisely, $\texttt{SLo(z)}$ is any chosen formula of the form 1 or $\texttt{Lo}_{\texttt{j}_1}(\texttt{z}_{\texttt{j}_1}) \otimes \ldots \otimes \texttt{Lo}_{\texttt{j}_\texttt{m}}(\texttt{z}_{\texttt{j}_\texttt{m}})$, where $1 \leq m \leq M$, the indices are mutually distinct and they indicate some numbers between 1 and M. $\texttt{RLo(r)}$ indicates the lower-bound alarms to be conditionally turned off. More precisely, $\texttt{RLo(r)}$ is any chosen formula of the form 1 or $\texttt{Lo}_{\ell_1}(\texttt{r}_{\ell_1}) \otimes \ldots \otimes \texttt{Lo}_{\ell_\texttt{k}}(\texttt{r}_{\ell_\texttt{k}})$, where $1 \leq k \leq M$, the indices are mutually distinct and they indicate some numbers between 1 and M distinct from $\texttt{j}_1, \ldots, \texttt{j}_\texttt{m}$. $\texttt{Lbound(r,t)}$ is a tensor product of formulas of the form 1 or $(\texttt{r}_{\ell_\texttt{j}} < \texttt{t})$, or $(\texttt{r}_{\ell_\texttt{j}} \leq \texttt{t})$, or $\texttt{r}_{\ell_\texttt{j}} < \infty$. $\texttt{SHi(x)}$ and $\texttt{RHi(y)}$ are defined similarly. They represent upper-bound alarms to be set and other upper-bound alarms to be released, respectively. We should also observe that $\texttt{Less}(\texttt{y}, \infty)$ denotes a tensor product of formulas of the form 1 or $(\texttt{y}_{\ell_\texttt{j}} < \infty)$ and that $\texttt{t}$, the $\texttt{x}$'s, the $\texttt{z}$'s, the $\texttt{y}$'s, and the $\texttt{r}$'s are all syntactically distinct individual variables. $\texttt{x}|(\texttt{t}+\texttt{hi})$ is the expression such that for each $\texttt{q} \in \mathcal{R}$ and $\texttt{p} \in \mathcal{R}_\infty$, $\texttt{p}|(\texttt{q}+\texttt{hi}) = \texttt{q}+\texttt{hi}$ *if* $\texttt{p} = \infty$ *else* $\texttt{p}$, where $\texttt{hi}$ is the corresponding bound. (Another, equivalent way of formulating the specifications is stated in the Appendix.)

Within our framework, we express the *safety property* in the following way.

Suppose that some events $\mathtt{b_1}$, ..., $\mathtt{b_N}$ should not occur together at the same moment. In order to verify that, we have to show that for any $\mathtt{q} \in \mathcal{R}$ and $\mathtt{p_j} \in \mathcal{R}_\infty$, $1 \leq j \leq K+M$, the corresponding formula (representing an *unsafety* configuration):

$$\mathtt{P_1(b_1)} \otimes \ldots \otimes \mathtt{P_N(b_N)} \otimes \mathtt{Time(q)} \otimes \mathtt{H_1(p_1)} \otimes \ldots \otimes \mathtt{H_K(p_K)} \otimes \mathtt{Lo_1(p_{K+1})} \otimes \ldots \otimes \mathtt{Lo_M(p_{K+M})},$$

is not derivable from a set of axioms $\mathcal{S}$, specifying our system, and a certain formula $\mathtt{Init}$ describing initial conditions.

In the next section we will show that such *safety* problems are decidable, and, moreover, these problems can be resolved in *polynomial space*.

4 Decidability in PSPACE

Let $\mathcal{S}$ be a finite set of specifications. We consider the following *parameterized provability problem*. Given any initial configuration described by the formula $\mathtt{Init}$: $\mathtt{P_1(a_{0,1})} \otimes \ldots \otimes \mathtt{P_N(a_{0,N})} \otimes \mathtt{Time(q_0)} \otimes \mathtt{H_1}(\infty) \otimes \ldots \otimes \mathtt{H_K}(\infty) \otimes \mathtt{Lo_1}(\infty) \otimes \ldots \otimes \mathtt{Lo_M}(\infty)$, and given any $\mathtt{b_i} \in \mathtt{E_i}$, $1 \leq i \leq N$, *does there exist* $\mathtt{q} \in \mathcal{R}$ and $\mathtt{p_j} \in \mathcal{R}_\infty$, $1 \leq j \leq K+M$, such that for the following formula $\mathtt{Reach}$:

$$\mathtt{P_1(b_1)} \otimes \ldots \otimes \mathtt{P_N(b_N)} \otimes \mathtt{Time(q)} \otimes \mathtt{H_1(p_1)} \otimes \ldots \otimes \mathtt{H_K(p_K)} \otimes \mathtt{Lo_1(p_{K+1})} \otimes \ldots \otimes \mathtt{Lo_M(p_{K+M})},$$

the formula $\mathtt{Init} \multimap \mathtt{Reach}$ is provable in quantifier-free multiplicative linear logic from the axioms $\mathcal{S}$ and the non-logical axioms for the predicates "$<$" and "$\leq$" given in Section 3.

It should be pointed out that given an initial configuration, the positive answer means that our system is *unsafe* with respect to $\mathtt{b_1}$, ..., $\mathtt{b_N}$. At the same time, this problem may be viewed as a kind of *satisfiability* problem, where $\mathtt{q}$, the $\mathtt{p_i}$'s, and choices of rules of inference in a proof search constitute a kind of assignment or model.

Theorem 4.1 *The parametric provability problem is decidable in polynomial space in the total size of the input that consists only of $\mathcal{S}$ and the bounds $\mathtt{hi_i}$ and $\mathtt{lo_j}$.*

We may assume without loss of generality that the bounds $\mathtt{hi_i}$, $\mathtt{lo_j}$ are natural numbers: the rational case easily follows by multiplication by the least common multiple of all denominators.

The reduced problem is again a parametric provability problem, but the existential problem involved is now a finite one. More precisely, the reduced setting involves an additional finite domain D, a unary predicate $\mathtt{Q}$ over D, finitely many new specifications $\mathcal{S}^+$ of the form $\mathtt{P_1(a_1)} \otimes \ldots \otimes \mathtt{P_N(a_N)} \otimes \mathtt{Q(d)} \multimap \mathtt{P_1(b_1)} \otimes \ldots \otimes \mathtt{P_N(b_N)} \otimes \mathtt{Q(e)}$ and a new initial configuration $\mathtt{Init}^+ \equiv \mathtt{P_1(a_{0,1})} \otimes \ldots \otimes \mathtt{P_N(a_{0,N})} \otimes \mathtt{Q(d_0)}$. Observe that the new specifications are timeless and that they involve full information about the configurations, which was not necessarily the case with the old specifications.

Given $\mathtt{b_i} \in \mathtt{E_i}$, $1 \leq i \leq N$, the new parameterized provability problem is whether there exists $d \in D$ such that the formula $\mathtt{Init}^+ \multimap (\mathtt{P_1(b_1)} \otimes \ldots \otimes \mathtt{P_N(b_N)} \otimes \mathtt{Q(d)})$ is provable in quantifier-free multiplicative linear logic from the axioms $\mathcal{S}^+$. This problem does not involve the reals or the inequalities. This new problem may be solved on-line, with some guidance, in the available automated environments based on linear logic [9, 10, 24, 25, 41, 30, 31, 32, 13], or in environments supporting multiset rewriting [17, 22]. In addition, note that the new decision problem is related to the standard reachability problem in a finite directed graph.

The construction of the additional domain D involves an equivalence relation on the configurations of the original, timed system with the specifications $\mathcal{S}$. For instance, let us assume that the original system has two upper-bound and two lower-bound timers, *i.e.,* $K = M = 2$. Then a configuration is typically indicated by a tuple $\gamma = (\mathtt{a}, \mathtt{t}_0, \mathtt{p}_1, \infty, \infty, \mathtt{q}_2)$, where $\mathtt{a}$ denotes $(\mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{N})$ and where $\mathtt{t}_0, \mathtt{p}_1, \mathtt{q}_2 \in \mathcal{R}$. This information indicates that at time $\mathtt{t}_0$, the state is $\mathtt{a}$, the timers $\mathtt{Hi}_2$ and $\mathtt{Lo}_1$ are off, the timer $\mathtt{Hi}_1$ is set to $\mathtt{p}_1 + \mathtt{hi}_1$, and the timer $\mathtt{Lo}_2$ is set to $\mathtt{q}_2 + \mathtt{lo}_2$.

Two such tuples γ and γ' are deemed equivalent iff (1) $\mathtt{a} = \mathtt{a}'$ and exactly the same timers are off, (2) for those timers $\mathtt{Hi}_\mathtt{i}$ that are on, it is the case that $\mathtt{t}_0 < \mathtt{p}_\mathtt{i} + \mathtt{m}$ iff $\mathtt{t}'_0 < \mathtt{p}'_\mathtt{i} + \mathtt{m}$ and that $\mathtt{t}_0 = \mathtt{p}_\mathtt{i} + \mathtt{m}$ iff $\mathtt{t}'_0 = \mathtt{p}'_\mathtt{i} + \mathtt{m}$ for each $\mathtt{m} = 1, \ldots, \mathtt{hi}_\mathtt{i} + 2$, (3) for those timers $\mathtt{Lo}_\mathtt{j}$ that are on, it is the case that $\mathtt{t}_0 < \mathtt{q}_\mathtt{j} + \mathtt{m}$ iff $\mathtt{t}'_0 < \mathtt{q}'_\mathtt{j} + \mathtt{m}$ and that $\mathtt{t}_0 = \mathtt{q}_\mathtt{j} + \mathtt{m}$ iff $\mathtt{t}'_0 = \mathtt{q}'_\mathtt{j} + \mathtt{m}$ for each $\mathtt{m} = 1, \ldots, \mathtt{lo}_\mathtt{j} + 2$, and (4) if $\mathtt{Hi}_{\mathtt{i}_1}, \ldots, \mathtt{Hi}_{\mathtt{i}_\mathtt{k}}$ and $\mathtt{Lo}_{\mathtt{j}_1}, \ldots, \mathtt{Lo}_{\mathtt{j}_\mathtt{m}}$ are exactly those timers that are on, then tuples $(\mathtt{t}_0, \mathtt{p}_{\mathtt{i}_1}, \ldots, \mathtt{p}_{\mathtt{i}_\mathtt{k}}, \mathtt{q}_{\mathtt{j}_1}, \ldots, \mathtt{q}_{\mathtt{j}_\mathtt{m}})$ and $(\mathtt{t}'_0, \mathtt{p}'_{\mathtt{i}_1}, \ldots, \mathtt{p}'_{\mathtt{i}_\mathtt{k}}, \mathtt{q}'_{\mathtt{j}_1}, \ldots, \mathtt{q}'_{\mathtt{j}_\mathtt{m}})$ are order-isomorphic in the sense of the ordering on the unit circle.

It may be shown that transitions and time advances respect this equivalence relation.

There are only finitely many equivalence classes, say N_0, at most $|E| \cdot (2\mathtt{hi}_1 + 5) \cdot \ldots \cdot (2\mathtt{lo}_{\mathtt{K}+\mathtt{M}} + 5) \cdot 5^{(\mathtt{K}+\mathtt{M}+1)^2}$, where $|E|$ is the product of the sizes of the domains E_i.

Let the new domain D consist of N_0 names of the form $(\mathtt{a}, \mathtt{v})$, where again $\mathtt{a}$ denotes $(\mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{N})$ and where $\mathtt{v}$ represents a complete list of answers to the questions indicated in conditions (2) $-$ (4) in the definition of equivalence. Note that in regard to each of the two lists of questions in (2) and likewise in (3) it suffices to specify the least m for which the answer is "yes", if any. Let $\mathtt{Q}$ be a formal unary predicate on D.

A formula of the form $\mathtt{P}_1(\mathtt{a}_1) \otimes \ldots \otimes \mathtt{P}_\mathtt{N}(\mathtt{a}_\mathtt{N}) \otimes \mathtt{Q}(\mathtt{a}, \mathtt{v}) \multimap \mathtt{P}_1(\mathtt{b}_1) \otimes \ldots \otimes \mathtt{P}_\mathtt{N}(\mathtt{b}_\mathtt{N}) \otimes \mathtt{Q}(\mathtt{b}, \mathtt{w})$ belongs to $\mathcal{S}^+$ iff there exists an instance of a transition formula τ in $\mathcal{S}$ other than time advance, applicable to the equivalence class named by $(\mathtt{a}, \mathtt{v})$, where that instance of τ possibly followed by a time advance results in the equivalence class named by $(\mathtt{b}, \mathtt{w})$. Note that the information about applicability of any formula instance is easily obtained from $(\mathtt{a}, \mathtt{v})$.

Regarding the polynomial space bound, indeed, we will take advantage of the fact that each of the names $(\mathtt{a}, \mathtt{v})$ and, hence, each of the new timeless transitions can be encoded in *polynomial space*. Furthermore, although there are in general exponentially many new transitions, we can manage our *exptime* decision procedure so that those transitions do not have to be written down all at once (which requires *exponential space*). We will generate the new transitions as needed in our decision procedure.

5 Conclusions and Future Work

We have investigated linear logic specifications of real-time, finite-state systems. A clear advantage of our approach is that it provides a common user with a very easy and transparent way of writing high-level specifications without having to be concerned with operational issues. In our approach, the runs of the system that satisfies the given specifications are exactly the linear logic proofs from the axioms given by the specifications, where proofs are presented in a certain normal form. In this way, an important system *safety* property, which means that certain system configurations are *unreachable,* is directly related to the logical notion of provability, in the sense that the formulas representing certain system configurations are *unprovable* from the specification axioms, and vice versa.

We show that provability is in PSPACE relative to the size of the specification axioms. (PSPACE-hardness is likely, because we may use punctual time specifications to simulate linear-bounded machines.) Our method of proof actually shows that syntactic properties involving temporal constraints may be decided, with some guidance, by *finite proof search* in the available logic programming environments based on linear logic [9, 10, 24, 25, 41, 30, 31, 32, 13] or in environments supporting multiset rewriting [17, 22] or concurrent rewriting [40, 44], either of which would in this case act as sort of model-checkers. In this sense, our work is a first step toward the study of *proof-based state exploration.* One advantage of our approach over model-based state exploration such as model-checking, is that our approach incorporates a decision procedure, so that either it provides unsafe runs or it actually establishes safety.

Our approach based on proof search represents concurrency by *interleaving* of proof rules, which take turns in being applied as atomic transitions. As with other interleaving models of concurrency such as [38], it is important to reconcile the formal approach by means of interleaved applications of proof rules with the notion of overlapped or even simultaneous executions of independent processors in actual concurrent systems. We are currently investigating several techniques of achieving this reconciliation, including the modelling within our approach of standard techniques from [38] such as introduction of additional states and events. However, our linear logic approach may provide other possible solutions as well, perhaps by means of so-called additive connectives.

The distinctions between multiplicative and additive connectives in linear logic might also be involved in expressing system properties such as *fairness* and *liveness.* This might also involve nontrivial combinations of first-order quantifiers. We plan to investigate this in the future, aided by the known EXPTIME decidability of the *pure* first-order multiplicative-additive linear logic [36, 35]. Because our complexity bound argument involves a further exponential reduction of timed transitions to a timeless setting, it would be at this level that one might expect the linear logic provability analogue of the known EXPSPACE-complete satisfiability in metric interval temporal logic [5]. Beyond the real-time systems, it would be quite interesting to see how the more general *hybrid systems* [2, 8] fit into our framework.

After the completion of this work, we became aware of the work of Fages *et al.* [16] on using *phase models* of linear logic for the verification of concurrent constraint programs. While their approach is model-based, it would be interesting to understand common points of their approach and ours and to investigate possible gains of combining them.

Acknowledgements: Kanovich and Scedrov would like to thank Keio University for hosting them during their visit in the fall of 1997, when the initial results were obtained. Scedrov would also like to thank Insup Lee and Patrick Lincoln for initial introduction to the literature on real-time systems, and Dale Miller for his advice on automated linear logic environments. The authors would also like to thank Rajeev Alur and Iliano Cervesato for their comments on an early draft of this paper.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.

[2] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

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

[4] R. Alur and D.L. Dill. Automata-theoretic verification of real-time systems. In *Formal Methods for Real-Time Computing*, pages 55–82. Trends in Software Series, John Wiley & Sons Publishers, 1996.

[5] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[6] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker et al., editor, *Real Time: Theory in Practice*, pages 74–106. Springer LNCS 600, 1992.

[7] R. Alur and T.A. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104:35–77, 1993.

[8] R. Alur, T.A. Henzinger, and H. Wong-Toi. Symbolic analysis of hybrid systems. In *Proc. 37-th IEEE Conference on Decision and Control*, 1997.

[9] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[10] J.-M. Andreoli, T. Castagnetti, and R. Pareschi. Static analysis of linear logic programming. *New Generation Computing*, 15(4), 1997.

[11] A. Asperti. A logic for concurrency. Technical report, Dipartimento di Informatica, Universitá di Pisa, 1987.

[12] P. Brémond-Grégoire, J.-Y. Choi, and I. Lee. A complete axiomatization of finite-state ACSR processes. *Information and Computation*, 138:124–159, 1997.

[13] I. Cervesato and F. Pfenning. A linear logical framework. In *Proc. 11-th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, NJ*, pages 264–275, July 1996.

[14] J. Davies and S. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138:243–271, 1995. See also Technical Report PRG-75, Oxford University Computing Laboratory, August 1989.

[15] U. Engberg and G. Winskel. Petri nets as models of linear logic. In A. Arnold, editor, *Proceedings of CAAP '90*. Springer LNCS 431, 1990.

[16] F. Fages, P. Ruet, and S. Soliman. Phase semantics and verification of concurrent constraint programs. In *Proc. 13-th Annual IEEE Symposium on Logic in Computer Science*, Indianapolis, 1998.

[17] P. Fradet and D. Le Métayer. Structured Gamma. Science of Computer Programming, to appear.

[18] V. Gehlot and C.A. Gunter. Normal process representatives. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[19] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[20] J.-Y. Girard. Linear logic: its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in linear logic*. London Mathematical Society Lecture Note Series, Cambridge University Press, 1995. Available by anonymous ftp from lmd.univ-mrs.fr as /pub/girard/Synsem.ps.Z.

[21] C.A. Gunter and V. Gehlot. Nets as tensor theories. In G. De Michelis, editor, *Proc. 10-th International Conference on Application and Theory of Petri Nets, Bonn*, pages 174–191, 1989.

[22] C. Hankin, D. Le Métayer, and D. Sands. Refining multiset transformers. Theoretical Computer Science, to appear.

[23] M. Hennessy and T. Reagan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[24] J.S. Hodas. Logic programming in intuitionistic linear logic: theory, design, and implementation. Dissertation, University of Pennsylvania, 1994.

[25] J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110:327–365, 1994.

[26] M.I. Kanovich. Effective program synthesis in computational models. *Journal of Logic Programming*, 9:159–177, 1990.

[27] M.I. Kanovich. The relational knowledge-base interpretation and feasible theorem-proving for intuitionistic propositional logic. University of Amsterdam, Institute for Logic, Language and Computation, ILLC Prepublication Series ML-93-21, December 1993.

[28] M.I. Kanovich. The complexity of Horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69:195–241, 1994.

[29] M.I. Kanovich. Effective calculi as a technique for search reduction. *American Mathematical Society Translations*, 178:133–148, 1996. Russian original in: Voprosy Kibernetiki, Moscow, No. 131, pp.148-168, 1987.

[30] N. Kobayashi. Concurrent linear logic programming. Dissertation, University of Tokyo, 1996.

[31] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, 7:113–149, 1995.

[32] N. Kobayashi and A. Yonezawa. Higher-order concurrent linear logic programming. In *Theory and Practice of Parallel Programming*, pages 137–166. Springer LNCS 907, 1995.

[33] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[34] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992.

[35] P. Lincoln and A. Scedrov. First order linear logic without modalities is NEXPTIME-hard. *Theoretical Computer Science*, 135:139–154, 1994.

[36] P. Lincoln and N. Shankar. Proof search in first order linear logic and other cut free sequent calculi. In *Proc. 9-th Annual IEEE Symposium on Logic in Computer Science, Paris*, pages 282–291, July 1994.

[37] P. Lincoln and T. Winkler. Constant-Only Multiplicative Linear Logic is NP-Complete. *Theoretical Computer Science*, 135:155–169, 1994.

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

[39] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. *Mathematical Structures in Computer Science*, 1:66–101, 1991. Revised version of paper in Springer LNCS 389.

[40] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey. *Internat. J. Foundations Comp. Sci.*, 2:297–399, 1991.

[41] D. Miller. A multiple-conclusion specification logic. *Theoretical Computer Science*, 165:201–232, 1996.

[42] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR '90*, pages 401–415. Springer LNCS 458, 1990.

[43] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[44] P.C. Olveczky and J. Meseguer. Specifying real-time systems in rewriting logic. *Electronic Notes in Theoretical Computer Science*, 4, 1996.

[45] J.S. Ostroff. *Temporal Logic of Real-Time Systems.* Research Studies Press, 1990.

[46] A. Pnueli. The temporal logic of programs. In *Proc. 18-th Annual Symposium on the Foundations of Computer Science*, pages 46–57, 1977.

[47] D. Miller R. McDowell and C. Palamidessi. Encoding transition systems in sequent calculus: Preliminary report. *Electronic Notes in Theoretical Computer Science*, 3, 1996.

[48] A. Scedrov. A brief guide to linear logic. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 377–394. World Scientific Publishing Co., 1993.

[49] A. Scedrov. Linear logic and computation: a survey. In H. Schwichtenberg, editor, *Proof and Computation, Proceedings Marktoberdorf Summer School 1993*, pages 281–298. NATO Advanced Science Institutes, Series F, Springer-Verlag, Berlin, 1994.

[50] N. Shankar. Mechanical verification of real-time systems using PVS. In C. Courcoubetis, editor, *5-th International Conference on Computer-Aided Verification*. Springer LNCS 697, 1993. Expanded version available as a Technical Report SRI-CSL-92-12, November, 1992.

[51] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[52] A.P. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

A Logical axioms and inference rules

The formal logical framework may be conveniently presented by means of Gentzen-style sequent calculus. We consider sequents of the form $\Gamma \vdash A$, where A is a formula obtained from atomic formulas described in Section 3 and the propositional constant 1 by the connectives $\otimes$ (tensor) and $\multimap$ (linear implication), and where Γ is a finite multiset of formulas. The English names for the rules given below are identity, tensor, linear implication, one, instantiation, and cut, respectively. In the rules, H denotes an atomic formula other than the propositional constant 1. A, B and C denote formulas, Σ and Δ denote finite multisets of formulas, and $A[c/x]$ indicates the result of substituting a constant c for all occurrences of an individual variable x in a formula A and carrying out the indicated additions, if any. For instance, $P(x+3)[2/x]$ indicates $P(5)$.

$$\frac{}{H \vdash H}\ \mathbf{I}$$

$$\otimes\mathbf{L}\quad \frac{\Gamma, A, B \vdash C}{\Gamma, (A \otimes B) \vdash C} \qquad\qquad \frac{\Sigma \vdash A \qquad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}\quad \otimes\mathbf{R}$$

$$\multimap\mathbf{L}\quad \frac{\Sigma \vdash A \qquad \Gamma, B \vdash C}{\Sigma, \Gamma, (A \multimap B) \vdash C} \qquad\qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash (A \multimap B)}\quad \multimap\mathbf{R}$$

$$1\mathbf{L}\quad \frac{\Gamma \vdash A}{\Gamma, 1 \vdash A} \qquad\qquad \frac{}{\vdash 1}\quad 1\mathbf{R}$$

$$\mathbf{Inst}\quad \frac{\Gamma, A[c/x] \vdash C}{\Gamma, A \vdash C} \qquad\qquad \frac{\Sigma \vdash A \qquad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}\quad \mathbf{Cut}$$

In this sequent calculus setting, the simplest formulation of all our non-logical axioms is of the form $A \vdash B$ instead of the form $\vdash A \multimap B$ indicated in Sections 3 and 4.