Model Checking Linear Duration Invariants of Networks of Automata ⋆

Miaomiao Zhang[1], Zhiming Liu[2], and Naijun Zhan[3]

[1] School of Software Engineering, Tongji University, Shanghai, China
miaomiao@tongji.edu.cn
[2] International Institute of Software Technology,
United Nations University, Macau, China
Z.Liu@iist.unu.edu
[3] Lab. of Computer Science, Institute of Software, CAS, Beijing, China
znj@ios.ac.cn

Abstract. Linear duration invariants (LDIs) are important safety properties of real-time systems. In this paper, we reduce the problem of verification of a network of timed automata against an LDI to an equivalent problem of model checking whether a failure state is never reached. Our approach is first to transform each component automaton $\mathcal{A}_i$ of the network $\mathcal{A}$ to an automaton $\mathcal{G}_i$. The transformation helps us to record entry and exit to critical locations that appear in the LDI. We then introduce an auxiliary checker automaton $\mathcal{S}$ and define a failure state to verify the LDI on a given interval. Since a model checker checks exhaustively, a failure of the checker automaton to find the failure state will prove that the LDI holds.

1 Introduction

The invariants constructed from linear inequalities of integrated durations of system states are important properties of real-time systems. For example, in a container unloading system, the required property has the form "for any observation interval that is longer than 60 seconds, the idle time for a device is at most one twentieth of the time".

This kind of properties are often specified by *linear duration invariants* (LDIs) [13] of the following form:

$$A \leq \ell \leq B \Rightarrow \sum_{s \in S} c_s \textstyle\int s \leq M \tag{1}$$

where $\int s$ is the duration of a state s, A, B, c_s and M are real numbers. The duration $\int s$ of a state s and the length ℓ are mappings from time intervals to reals. For an observation time interval $[b, e]$, $\int s$ defines the accumulated time for the presence of state s over $[b, e]$ and ℓ is the length $e - b$ of the interval. An LDI $\mathcal{D}$ simply says that for any observation time interval $[b, e]$, if the length ℓ of the interval satisfies the constraint $A \leq \ell \leq B$ then

⋆ The work is partly supported by the projects NSFC-60603037, NSFC-90718014, NSFC-60721061, NSFC-60573007, NSFC-90718041, NSFC-60736017, and HighQSoftD and HTTS funded by Macao S&TD Fund.

the durations of the system states over that interval should satisfy the linear constraint $\sum_{s\in S} c_s \int s \leq M$. We use $\Sigma(\mathcal{D})$ to denote the sum of the durations $\sum_{s\in S} c_s \int s$.

In this paper we consider the problem of automatic verification of a network of timed automata against an LDI, where each automaton is *closed* and *diagonal-free*. To address the issue, several algorithms have been proposed in the literature e.g. [3, 10]. Different from the existing methods, in this paper, we develop a technique to reduce the problem of verification of a network of timed automata against an LDI to an equivalent problem of model checking whether a failure state cannot be reached. This will allow us to use existing model checkers, such as UPPAAL, to check the LDI. Our approach is first to transform each automaton $\mathcal{A}_i$ of the network $\mathcal{A}$ to an automaton $\mathcal{G}_i$. The transformation helps us to record entry and exit to critical locations that appear in the LDI.

Then we introduce an auxiliary timed automaton $\mathcal{S}$ from $\mathcal{A}$ and the LDI. $\mathcal{S}$ is used to calculate the observation time and the sum $\Sigma(\mathcal{D})$. In $\mathcal{S}$ we use a variable gc to record observation time, and another variable d to calculate the durations of system states. To approach the goal, the timed automaton $\mathcal{S}$ is constructed in different ways according to whether the constant B in (1) is finite or not. Subsequently, we define a failure state in $\mathcal{S}$ from the LDI $\mathcal{D}$, and prove that $\mathcal{D}$ is satisfied by $\mathcal{A}$ iff the failure state is never reached.

The rest of the paper is organized as follows. Section 2 recalls some basic notions of timed automaton and Duration Calculus. The main technical contribution is presented in Section 3. We present algorithms on how to construct the transformed automata $\mathcal{G}_i$ from an LDI and $\mathcal{A}_i$, and the two kinds of automata $\mathcal{S}$ respectively corresponding to the cases when B is finite and when B is infinite, and prove the main theorems. A case study is given in Section 4 to illustrate our technique. Section 5 gives a comparison between our approach and related work, discusses future work and concludes the paper.

2 Preliminaries

In this section, we introduce the notions that will be used later including the modelling language of UPPAAL, and Linear Duration Invariants (LDIs) defined in DC.

2.1 The modelling language

We first recall the notion of timed automata given in [1, 2]. A timed automaton is a finite state machine equipped with a set of clocks. We use a set X of real value variables to represent the clocks and let $\Phi(X)$ be the set of clock constraints on X, which are conjunctions of the formulas of the form $x \leq c$ or $c \leq x$, where $x \in X$ and $c \in \mathbf{N}$. Formally,

Definition 1. *A timed automaton $\mathcal{A}$ is a tuple $\mathcal{A} = (L, l_0, \Sigma, X, E, I)$, where*

- *L is a finite set of locations,*
- *$l_0 \in L$ is the initial location,*
- *Σ is a finite set of actions, co-actions and the internal τ actions,*
- *X is a finite set of clocks,*

– *I is a mapping that assigns each location $l \in L$ with a clock constraint $I(l) \in \Phi(X)$ called the invariant at l.*
– *$E \subseteq L \times \Phi(X) \times \Sigma \times 2^X \times L$ is a relation among locations, whose elements are called edges and labeled with an action, a guard and a set of clocks to be reset.*

Network of timed automata A set of N timed automata $\mathcal{A}_i = (L_i, l_{i0}, \Sigma_i, X_i, E_i, I_i)$, $i = 1 \ldots N$, on the same sets of clocks and actions, are often composed into a network $\mathcal{A}$. A location for the network $\mathcal{A}$ is a vector $l = (l_1, \ldots, l_i, \ldots, l_N) \in L_1 \times \ldots \times L_N$ with the invariant $I(l) = \wedge_i I_i(l_i)$.

Binary synchronisation Channels are declared as `chan` C. An edge labelled with $C!$ is synchronized with another labelled with $C?$. A synchronized pair is chosen nondeterministically if several combinations are enabled.

Priorities on processes We follow the definition of priority of processes given in UPPAAL [15]. Process priorities are specified on the system line, using the separator $<$ to define a higher priority for processes to its right. If an instance of a template set is listed, all processes in the set will have the same priority.

The semantics of timed automata, networks of timed automata, channels are given in [15] and the help file of the UPPAAL tool.

2.2 Duration calculus and linear duration invariants

Duration calculus DC [12] is a logic for reasoning about durations of states of real-time systems. A comprehensive introduction to DC is given in the monograph by Zhou and Hansen [14]. In DC, a state s is *interpreted* as a function from the time domain $\mathbf{R}^+$ to the boolean values $1, 0$, and s is 1 at time t if the system is in state s and 0 otherwise. the duration of a state s over the time interval $[b, e]$ is defined as the integral $\int_b^e s(t)dt$, which is exactly the accumulated present time of s in the interval $[b, e]$. ℓ is used to denote the length of considered interval, which is defined by $\int 1$. We have the following measure laws on durations[4]:

1. $0 \leq \int s \leq \ell$
2. $\int \neg s = \ell - \int s$
3. $\int s_1 \vee s_2 = \int s_1 + \int s_2 - \int s_1 \wedge s_2$

We consider the set of DC models that corresponds to all the behaviors of the network of timed automata $\mathcal{A}$. A behavior ρ of $\mathcal{A}$ is of the form $(S_0, t_0)(S_1, t_1) \ldots$, where each S_i is called a state of $\mathcal{A}$ which is a subset of the state variables of $\mathcal{A}$ and t_is are incremental, i.e. $t_i \leq t_{i+1}$ for any $i \in \mathbf{N}$. Each behaviour defines an interpretation $\mathcal{I}$ of the DC formulas over the state variables of $\mathcal{A}$: for any state variable s of $\mathcal{A}$, $\mathcal{I}_s(t) = 1$ iff $\exists i \cdot (s \in S_i \wedge t \in [t_i, t_{i+1}])$. We also denote such $\mathcal{I}$ by $(\overline{s}, \overline{t})$ where $\overline{s} = (S_0, S_1, \ldots)$ and $\overline{t} = (t_0, t_1, \ldots)$ are respectively the sequence of states S_i and the sequence of time

[4] There are six axioms on durations (see [14]), but here we just list some of them which are used in this paper.

stamps t_i of ρ. Hence, $(\overline{s}, \overline{t}, [b, e])$ is a DC model representing an observation of $\mathcal{A}$ in the time interval $[b, e]$. We also call $(\overline{s}, \overline{t}, [b, e])$ an $\mathcal{A}$-model of DC.

For a given network of timed automata $\mathcal{A}$, we define the set of $\mathcal{A}$-models of DC with integral observation intervals [11] as

$$\mathcal{M}_I(\mathcal{A}) \mathrel{\widehat{=}} \{\sigma \mid \sigma = (\overline{s}, \overline{t}, [b, e]) \in \mathcal{M}(\mathcal{A}) \text{ and } b, e \in \mathbf{N}, b \leq e\}$$

Linear duration invariants A *linear duration invariant* (LDI) of a network of timed automata $\mathcal{A}$ is a DC formula $\mathcal{D}$ of the form

$$A \leq \ell \leq B \Rightarrow \sum_{s \in L} c_s \textstyle\int s \leq M$$

where A, B, c_s and M are real numbers.

An LDI $\mathcal{D}$ is evaluated in an $\mathcal{A}$-model $(\mathcal{I}, [b, e])$ to tt, denoted by $(\mathcal{I}, [b, e]) \models \mathcal{D}$, iff $A \leq e - b \leq B \Rightarrow \sum_{s \in L} c_s \int_b^e \mathcal{I}_s(t)dt \leq M$ holds. $\mathcal{D}$ is satisfied by $\mathcal{A}$, denoted by $\mathcal{A} \models \mathcal{D}$, if $(\mathcal{I}, [b, e]) \models \mathcal{D}$ holds for all $\mathcal{A}$-models $(\mathcal{I}, [b, e])$. We use $\Sigma(\mathcal{D})$ to denote the the sum of the durations $\sum_{s \in L} c_s \int s$.

Digitization of LDIs w.r.t. timed automaton Henzinger et al. [5] studied the question of which real-time properties can be verified by considering system behaviours featuring only integer durations. These results are applied to timed automata in [7], and it is shown that an approach using digital clocks is applicable to the verification of closed, diagonal-free timed automata. The digitization of duration calculus has also been studied in [6, 8, 9]. As to the digitization of an LDI, the following theorem has been proved in [11], where $\mathcal{A}$ is closed and diagonal-free.

Theorem 1. $\mathcal{M}(\mathcal{A}) \models \mathcal{D} \Leftrightarrow \mathcal{M}_I(\mathcal{A}) \models \mathcal{D}$.

Therefore only the set of integral $\mathcal{A}$ models of DC are studied in [11]. In the rest of the paper, we also only consider model $\sigma = (\overline{s}, \overline{t}, [b, e]) \in \mathcal{M}_I(\mathcal{A})$ that represents an observation of an integral behavior of $\mathcal{A}$ (i.e behavior in which transitions take place only at integer time) from an integer time point to an integer time point. So we will restrict $\mathcal{A}$ to be an integer-time model.

3 Verification of LDIs

In this section, we present our technique to reduce the verification of the satisfaction of an LDI $\mathcal{D}$ by a network of timed automata $\mathcal{A}$ to checking the property whether a failure state cannot be reached. In what follows, each automaton $\mathcal{A}_i$ is referred to an integer-time model. We start with the calculation of the duration of a location of the composite automaton of the network, i.e. a location vector.

3.1 Duration of a location vector s_j

Let $\mathcal{A}$ be a network of N timed automata $\mathcal{A}_i = (L_i, l_{i0}, \Sigma_i, X_i, E_i, I_i)$, $i = 1, \ldots, N$. A location of $\mathcal{A}$ is a vector $l = (l_1, \ldots, l_i \ldots l_N)$, where $l_i \in L_i$.

A location can be seen as a state variable. In the following, we require that each state expression s_j of a duration term $\int s_j$ in the LDI $\mathcal{D}$ is constructed by using logical connectives from the locations l_i of the component automata $\mathcal{A}_i$. For example, $l_1 \wedge l_2$ asserts that $\mathcal{A}$ is in a location where $\mathcal{A}_1$ and $\mathcal{A}_2$ are respectively in location l_1 and l_2. We are particularly interested in those state expressions s_j of the form $l_{a_1} \wedge \ldots \wedge l_{a_k}$, where $l_{a_i} \in L_{a_i}$ for $\{a_1, \ldots, a_k\} \subseteq \{1, \ldots, N\}$. We represent such a state expression by a *vector with free locations* $f \in (L_1 \cup \{\times_1\}) \times \ldots \times (L_N \cup \{\times_N\})$, such that $f[a_i] = l_{a_i}$and $f[b] = \times_b$ for $b \in \{1, \ldots, N\} \setminus \{a_1, \ldots, a_k\}$. This defines the set of vectors whose bth component $\times_b$ can be any location of $\mathcal{A}_b$.

Using the axioms in Subsection 2.2, it is easy to equivalently transform a general LDI to an LDI in which all state expressions can be represented by either a full location vector $\mathcal{A}$ or a vector with free locations. Thus, in the rest of the paper we only consider LDIs of this form.

We call each s_j that appears in an LDI a *critical* location vector of $\mathcal{A}$ and use K to denote the number of *critical* location vectors in the LDI. A location l_i of an automaton $\mathcal{A}_i$ is called *p-critical* location of s_j if l_i occurs in s_j as a non-free location and $\mathcal{A}_i$ is a *critical* automaton to s_j. A location l_i of an automaton $\mathcal{A}_i$ is called *p-critical* location if it is a *p-critical* location of some s_j. We use W to denote the number of *p-critical* locations.

Example 1. Consider the following LDI

$$c_{s_1} \int (l_{11}, l_{22}, l_{31}) + c_{s_2} \int (\times_1, \times_2, l_{31}) + c_{s_3} \int (\times_1, l_{21}, l_{32}) \ \leq M$$

Its *critical location vectors* are respectively $s_1 = (l_{11}, l_{22}, l_{31})$, $s_2 = (\times_1, \times_2, l_{31})$ and $s_3 = (\times_1, l_{21}, l_{32})$, while its *p-critical locations* are l_{11}, l_{22}, l_{31}, l_{21} and l_{32}. Obviously, in this example, $K = 3$ and $W = 5$.

For a *critical* location vector s_j of $\mathcal{A}$, with the elapse of one time unit, $\mathcal{A}$ stays in s_j for one time unit if each automaton $\mathcal{A}_i$ *critical* to s_j stays in the *p-critical* location of s_j for one time unit. This one unit delay of $\mathcal{A}$ in s_j causes an increase of c_{s_j} to the sum $\Sigma(\mathcal{D})$ of the LDI.

Main technique With the above definitions, the main idea of the technique can be sketched as follows:

- Firstly, we construct a network of automata $\mathcal{G}$ from $\mathcal{A}$ and $\mathcal{D}$ to record the entry and exit of the *p-critical* locations.
- As we need to check from any reachable state, whenever the antecedent of the LDI is true implies that its conclusion holds, we introduce $\mathcal{S}$ to count the observation time gc and the sums of the durations of the *critical* location vectors d from any reachable state.
- Finally, we construct a failure state such that $\mathcal{A} \models \mathcal{D}$ if and only if this failure state is never reached in $\mathcal{S}$.

3.2 Transformation of the network of automata

The network of automata $\mathcal{A}$ is first transformed to another network of automata to record the entry and exit of the *p-critical* locations. For this, we need to introduce a Boolean

array $active$ with size W to indicate whether the *p-critical* locations are entered. The index k of this array denotes the $(k+1)^{th}$ *p-critical* location in the LDI. Initially the value of $active[k]$ is 0. It is set to 1 when the $(k+1)^{th}$ *p-critical* location is entered, and set to 0 when this location is exited.

We transform each automaton $\mathcal{A}_i$ to an automaton $\mathcal{G}_i$. $\mathcal{G}_i$ is similar to $\mathcal{A}_i$ except that for the entry and exit of each *p-critical* location, the value of $active$ is updated by 1 and 0 respectively. Note here if the initial location of $\mathcal{A}_i$ is the $(k+1)^{th}$ *p-critical* location, then an additional urgent location is introduced in $\mathcal{G}_i$ to set the value of $active[k]$ to 1. We call $\mathcal{G} = (\mathcal{G}_1, \| \ldots, \| \mathcal{G}_N)$ constructed by this procedure the *network of transformed automata* of $\mathcal{A}$ for $\mathcal{D}$.

Example 2. Fig.1 gives a case of the transformation from $\mathcal{A}_1$ to $\mathcal{G}_1$, where `l0` is the first *p-critical* location and `l2` is the third *p-critical* location. An additional urgent location is introduced to set the value of $active[0]$ to 1.

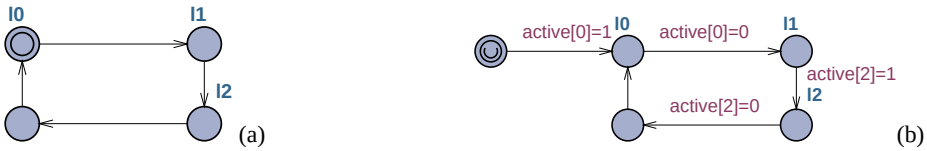

Fig. 1. Transformation from $\mathcal{A}_1$ to $\mathcal{G}_1$ a) $\mathcal{A}_1$ b) $\mathcal{G}_1$

3.3 Construction of the auxiliary automaton $\mathcal{S}$

In order to check whether the LDI is satisfied, we check for any path from a reachable state that the sum of the durations of the *critical location vectors* does not exceed M within the time interval $[A, B]$. For this, we build an auxiliary automaton $\mathcal{S}$, where the following variables are introduced and initialized to 0.

- gc is a local variable in $\mathcal{S}$ to record the length of an observation interval from s_r in a path of $\mathcal{G}$, and
- x is a local clock variable in $\mathcal{S}$ to record the elapse of one time unit, and
- d is a local variable in $\mathcal{S}$ to record the sum of the durations of the *critical* location vectors, i.e. the value $\Sigma(\mathcal{D})$.

In the construction of $\mathcal{S}$, to bound the value of d and gc, we need to use different methods dependent on the constant B in an LDI is finite or infinite. For the update of variable gc, we introduce $B+1$-normalization when B is finite and A-normalization when B is infinite. Here A is the other constant in the antecedent of the LDI.

Definition 2. *($B+1$-normalization)*

$$norm_{B+1}(gc) = \begin{cases} gc+1, & \text{if } gc \leq B \\ B+1, & \text{if } gc > B \end{cases}$$

The intuitive intention is that gc records the length of the current observation interval, and the LDI $\mathcal{D}$ is satisfied trivially when it exceeds the constant B in the antecedent of $\mathcal{D}$. Hence, we do not need to record all the values of gc that are bigger than B. It is sufficient to record $B+1$ when the length of the observation time exceeds B.

Definition 3. *(A-normalization)*

$$norm_A(gc) = \begin{cases} gc+1, & \text{if } gc < A \\ A, & \text{if } gc \geq A \end{cases}$$

Intuitively, the A normalization is dual to the B-normalization. With this normalization, for checking LDI $\mathcal{D}$ when gc equals A, we only need to check whether there exists a path along which the value of $\Sigma(\mathcal{D})$ is bigger than M.

In both cases (when B is finite and B is infinite), we require that the process $\mathcal{S}$ has higher priority than any other process. This is declared by `system` $\mathcal{G}_1, \ldots, \mathcal{G}_N < \mathcal{S}$, which means that at a given time-point, a transition in $\mathcal{G}_i$ is enabled only if all transitions of $\mathcal{S}$ are disabled.

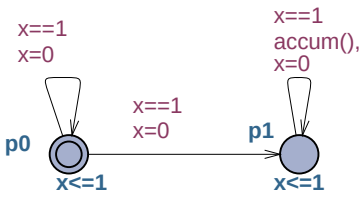

Fig. 2. Auxiliary automaton $\mathcal{S}$

When B is finite Fig. 2 shows the automaton $\mathcal{S}$ when B is finite. There are two locations `p0` and `p1` and initially $\mathcal{S}$ stays in `p0`. The trick that $\mathcal{S}$ will nondeterministically stay in `p0` for any number of time units before moving to `p1` ensures that gc and d will start to count from any reachable state of $\mathcal{A}$.

In location `p1` of $\mathcal{S}$, with the elapse of one time unit, the values of gc and d are updated. These are implemented by the function `accum`() given in Fig 3, where we still use s_j to denote a *critical* location vector and K to denote the number of *critical* location vectors. In `accum`(), the first assignment assigns gc the value of $gc+1$ if $gc \leq B$ and the value $B+1$ otherwise, i.e. it is the implementation of the $B+1$ normalization.

Note that with one time unit elapsed, $\mathcal{A}$ may not stay in any of the *critical* location vectors or may stay in several *critical* location vectors during the time unit. Therefore, function `accum`() uses a "for" loop the latter case so that the update of d is correct. By the definitions of Subsection 3.1, checking if the duration of a *critical* location vector

s_j is 1 over the one time unit interval is equivalent to checking if the duration of each *p-critical* location of this vector is 1. The following theorem is used to decide if the duration of a *p-critical* location is 1.

Theorem 2. *Let $\mathcal{G}$ be the network of transformed automata of $\mathcal{A}$ and $\mathcal{S}$ be the auxiliary automaton of $\mathcal{A}$ defined above. With one time unit elapsed when function accum() is executed, for the k^{th} p-critical location l, if $active[k-1] = 1$ then the duration of l is 1 over this one time unit interval, otherwise the duration of l is 0.*

Proof. Since each automaton $\mathcal{A}_i$ we consider is an integer-time model, each discrete transition of the transformed automaton $\mathcal{G}_i$ is therefore taken at integer time point. As observed from $\mathcal{S}$, in location `p1` function `accum()` is enabled each one time unit. By the declaration of process priority: `system` $\mathcal{G}_1, \ldots, \mathcal{G}_N < \mathcal{S}$, we have that any transition that is enabled to exit or enter a *p-critical* location must be executed after the execution of `accum()`.

Let l be the k^{th} *p-critical* location of $\mathcal{G}_i$. Let τ_1 be the transition that enters the *p-critical* location l with the assignment $active[k-1] = 1$ and τ_2 be the transition that exits location l with the assignment $active[k-1] = 0$. In location `p1`, consider the one time unit interval $II = [\mathcal{S}.x = 0, \mathcal{S}.x = 1]$, where x is the local clock in $\mathcal{S}$. At time point $\mathcal{S}.x = 1$, function `accum()` is executed before any other enabled transition to check the duration of l over the interval II. To do this, it checks the value of $active[k-1]$.

- When $active[k-1] = 1$, suppose the duration of l is not 1 over the interval II, that is, it is either 0 or lies in the interval $(0, 1)$. In the former case, it implies that at the time point $\mathcal{S}.x = 1$, τ_1 is taken before `accum()`, which violates the assumption of process priority. In the latter case, it means that the time point that the action τ_1 takes place lies in the interval $II' = (\mathcal{S}.x = 0, \mathcal{S}.x = 1)$. This also contradicts the fact that $\mathcal{G}_i$ is an integer-time model. Therefore, when $active[k-1] = 1$, the duration of l is 1 over II.
- When $active[k-1] = 0$, suppose the duration of l is not 0 over the interval II, that is, it is either 1 or lies in the interval $(0, 1)$. If it is 1, then it must be the case that $\mathcal{G}_i$ stays in l for one time unit and τ_2 is executed before `accum()`. This also violates the assumption of process priority. If the duration of l is in the interval $(0, 1)$, then the time point τ_2 takes place lies in the interval $II' = (\mathcal{S}.x = 0, \mathcal{S}.x = 1)$. However this kind of transition is not allowed in $\mathcal{G}_i$. So when $active[k-1] = 1$, the duration of l is 0 over II.

We conclude the proof. □

The above theorem allows the calculation of the duration of a *critical* location vector in terms of the information of entry or exit of its *p-critical* locations. Obviously, if the duration of a *critical* location vector s_j is 1 over one time unit interval, the value of d is increased by the value of the coefficient of this vector, i.e, c_{s_j}. So the construction of $\mathcal{S}$ correctly records the durations of the *critical* location vectors from any reachable state of $\mathcal{A}$. In addition, all the variables introduced are bounded. This is because by assigning 0 to d when $gc > B$, the value of variable d is finite, also gc is bounded by $B + 1$.

```
void accum()
{     gc = (gc ≤ B?gc + 1 : B + 1)
      for (j = 1, j ≤ K, j++)
      { if each p-critical location of s_j is entered
            d = (gc ≤ B?d + c_{s_j} : 0)
      }
}
```

Fig. 3. Function `accum`() when B is finite

The corresponding failure state The failure state $\mathcal{F}$ is $A \leq gc \leq B \wedge d > M$. We check $\mathcal{F}$ cannot be reached in $\mathcal{S}$. This property can be expressed in CTL [4] as

$$\psi_1 \mathrel{\widehat{=}} \texttt{A[] not } \mathcal{F} \tag{2}$$

We call $\mathcal{F}$ the *failure state* of $\mathcal{D}$ for $\mathcal{A}$.

Lemma 1. *Let $\mathcal{D}$ be an LDI of the network of timed automata $\mathcal{A}$, $\mathcal{G}$ the network of transformed automata of $\mathcal{A}$ for $\mathcal{D}$, $\mathcal{S}$ the auxiliary automaton of $\mathcal{A}$ for $\mathcal{D}$, $\mathcal{G} \parallel \mathcal{S}$ the parallel composition of G and S, $\mathcal{P}(\mathcal{G} \parallel \mathcal{S})$ the set of all paths of $\mathcal{G} \parallel \mathcal{S}$ and ψ_1 the failure state property. Then there exists a path $\rho_g \in \mathcal{P}(\mathcal{G} \parallel \mathcal{S})$ such that $\rho_g \not\models \psi_1$ iff there exists a path $\rho \in \mathcal{P}(\mathcal{A})$ such that $\rho \not\models \mathcal{D}$.*

Proof. From the construction procedure for $\mathcal{G}$ and $\mathcal{S}$, there is an obvious correspondence between a path ρ of $\mathcal{A}$ and a path ρ_g of $\mathcal{G} \parallel \mathcal{S}$ starting from the initial locations, that represents an observation of the system in the two models. Let $\ell(\rho)$ be the length of ρ, which represents the time of the observation, and $last(\rho_g)$ be the last node of ρ_g.

1. When $\ell(\rho) \leq B$, the value of gc at $last(\rho_g)$ equals $\ell(\rho)$, and the value of d at $last(\rho_g)$ is the value of the sum $\Sigma(\mathcal{D})$.
2. When $\ell(\rho) > B$, the value of gc at $last(\rho_g)$ is $B + 1$.

Consequently, the lemma follows immediately from the definition of the satisfaction relations $\models$ for LDIs and the definition of the failure state. □

Theorem 3. *Let $\mathcal{F}$ be the failure state of $\mathcal{D}$ for $\mathcal{A}$. When B is finite, $\mathcal{A} \models \mathcal{D}$ if and only if state $\mathcal{F}$ is never reached in $\mathcal{G} \parallel \mathcal{S}$.*

Construction example Fig. 4 and Fig.5 give a case of the construction of the network of transformed automata $\mathcal{G}$ from $\mathcal{A}$ and the correspondent automaton $\mathcal{S}$. The LDI is of the form:

$$A \leq \ell \leq B \Rightarrow\ c_{s_1} \textstyle\int (l_j, l_m) + c_{s_2} \int (\times_1, l_n) \leq M$$

The *critical* location vectors are $s_1 = (l_j, l_m)$ and $s_2 = (\times_1, l_n)$. The first *p-critical* location is l_j, the second *p-critical* location is l_m and the third *p-critical* location is l_n.

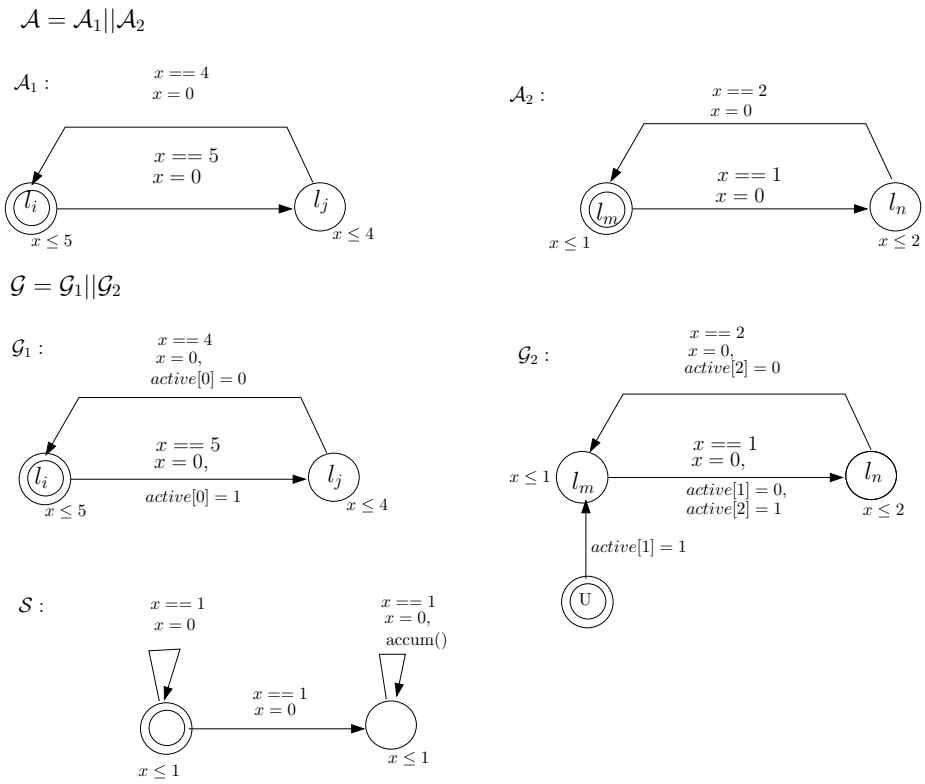

Fig. 4. Case example of the network of transformed automata $\mathcal{G}$ and $\mathcal{S}$ when B is finite

```
void accum()
{
    gc = (gc ≤ B?gc + 1 : B + 1)
    if active[0] × active[1] == 1
        d = (gc ≤ B?d + c_{s_1} : 0)
    if active[2] == 1
        d = (gc ≤ B?d + c_{s_2} : 0)
}
```

Fig. 5. Function `accum`() of $\mathcal{S}$

When B is infinite In terms of the automaton $\mathcal{S}$ constructed in the previous subsection, gc can increase infinitely and d can take an arbitrary value if B is infinite. The above theorem does not apply anymore. To bound the value of gc we will use "A-normalization" introduced before. Now we introduce a number to bound the value of d.

Definition 4. *A critical location vector s_p in $\mathcal{A}$ is said* positive *if $c_{s_p} > 0$. Let L_+ be the set of all positive critical location vectors in $\mathcal{A}$, l_i the ith p-critical location of a positive critical vector s_p and $\mathtt{u}(l_i)$ the maximum time units that $\mathcal{A}_i$ stays in l_i. We define $\mathtt{u}(s_p) = \mathtt{min}\{\mathtt{u}(l_1), \ldots, \mathtt{u}(l_i), \ldots, \mathtt{u}(l_N)\}$, and call $Q = \sum_{s_p \in L_+}(c_{s_p} \times \mathtt{u}(s_p))$ the maximum increment of $\mathcal{A}$.*

Note that $\mathtt{u}(s_p)$ is the *maximum* time that $\mathcal{A}$ stays in s_p because of the clock synchronization. This value is used in the calculation of Q, and Q is used to detect if a path of $\mathcal{A}$ contains a *positive loop* that takes non-zero time. If there is no *positive loop* in a path of $\mathcal{A}$, the value of d along that path can increase at most Q. In other words, if the value of d along a path increases more than Q, then there must be a positive loop in the path. It is in general difficult to calculate the actual value of $\mathtt{u}(s_p)$ as it requires all the $\mathtt{u}(l_i)$'s. So usually we calculate a value that is bigger than $\mathtt{u}(s_p)$ and assign it to $\mathtt{u}(s_p)$ when calculating Q.

Example Suppose for the case shown in Fig. 4, $c_{s_1} > 0$ and $c_{s_2} > 0$. Then we can let $Q = 4 \times c_{s_1} + 2 \times c_{s_2}$.

The auxiliary automaton $\mathcal{S}^+$ is similar to the one shown in Fig.2 except that the updates of the value of gc and d are different. In other words, the function $\mathtt{accum}()$ is different. For any *critical* location vector s_j, we make the variable d bounded by updating it in different ways depending on whether the coefficient c_{s_j} of $\int s_j$ in $\Sigma(\mathcal{D})$ is negative or not.

1. The update of gc is done by the *A-normalization*.
2. For the accumulation of d,
   - if s_j has a non-negative coefficient c_{s_j},
     $d = (gc \geq A \wedge d > M ? M + 1 : d + c_{s_j})$.
   - if s_j has a negative coefficient c_{s_j},
     $d = (gc \geq A \wedge d < M - Q ? d : d + c_{s_j})$

When c_{s_j} is non-negative, if $gc \geq A$ and $d > M$, by setting d to $M + 1$, the value of d is finite. Moreover, when $gc \geq A$, gc remains as A, so gc is a bounded variable. Since the states that satisfy $gc \geq A \wedge d = M + 1$ imply $\mathcal{G} \parallel \mathcal{S}^+ \not\models \mathcal{D}$, it is obvious that the update does not change the verification result.

If c_{s_j} is negative, the value update of d is $d = (gc \geq A \wedge d < M - Q ? d : d + c_{s_i})$. It is not hard to see why we set d to $d + c_{s_i}$ if $\neg(gc \geq A \wedge d < M - Q)$: we have to evaluate the value of d precisely when we do not have enough information for verifying if $\mathcal{D}$ is satisfied.

Now we prove that if $gc \geq A \wedge d < M - Q$, the value of d remaining unchanged does not change the checking result of the LDI. To do so, we define another graph $\mathcal{S}^\bullet$

that is the same as $\mathcal{S}^+$ except that if $gc \geq A \wedge d < M - Q$ the assignment for d is $d = d + c_{s_j}$ in function `accum()`.

Similar to the case when B is finite, we define the *failure state* $\mathcal{F}'$: $gc \geq A \wedge d > M$. Still we use CTL to express that $\mathcal{F}'$ is never reached.

$$\psi_2 \mathrel{\widehat{=}} \texttt{A[] not } \mathcal{F}' \tag{3}$$

Lemma 2. *Let $\mathcal{P}(\mathcal{G} \parallel \mathcal{S}^+)$ be the set of pathes of $\mathcal{G} \parallel \mathcal{S}^+$. There exists a path $\rho \in \mathcal{P}(\mathcal{G} \parallel \mathcal{S}^+)$ such that $\rho \not\models \psi_2$ if and only if there exists a path $\rho' \in \mathcal{P}(\mathcal{G} \parallel \mathcal{S}^\bullet)$ such that $\rho' \not\models \psi_2$.*

Proof. Notice that the topological structure of $\mathcal{S}^+$ and $\mathcal{S}^\bullet$ are the same. Each path $\rho = s_0^+, \ldots, s_m^+$ in $\mathcal{G} \parallel \mathcal{S}^+$ corresponds to exactly one path $\rho^\bullet = s_0^\bullet, \ldots, s_m^\bullet$ in $\mathcal{G}^\bullet \parallel \mathcal{S}$. Let s_i^+ and $s_i^\bullet$ be any two corresponding nodes respectively in ρ and $\rho^\bullet$. Then the value of gc at vertex s_i^+ is the same as the value of that at vertex $s_i^\bullet$. Due to the different updates of d in ρ and $\rho^\bullet$ for the negative coefficient of a vertex, we know that at vertex s_i^+, the value of d is bigger than or equal to the value of d at $s_i^\bullet$. Hence, if a path $\rho' = \rho^\bullet$ in $\mathcal{G} \parallel \mathcal{S}^\bullet$ does not satisfy ψ_2, then its corresponding path ρ in $\mathcal{G} \parallel \mathcal{S}^+$ does not satisfy ψ_2.

To prove the other direction, let ρ in $\mathcal{G} \parallel \mathcal{S}^+$ be such that $\rho \not\models \psi_2$ and ρ starts from the initial location. If $\rho^\bullet \not\models \psi_2$, we are done. Otherwise, we need to show that there will be a "positive cycle" in ρ, i.e. there is a cycle such that going along the cycle will increase the value of d properly by at least 1. We now give the illustration for the case $\rho \not\models \psi_2 \wedge \rho^\bullet \models \psi_2$. This case denotes that the values of d on ρ and on $\rho^\bullet$ are different and there should be a first node s_j^+ along ρ where the condition $gc \geq A \wedge d < M - Q \wedge c_{s_j} < 0$ holds. Thus, from s_j^+, the value of d is increased by at least $Q + 1$ to make $\rho \not\models \psi_2$.

From the definition of Q, in ρ there must be a "positive cycle" along which d will be increased by at least 1. From the correspondence relation between ρ and $\rho^\bullet$, $\rho^\bullet$ must also have a positive cycle $\mathcal{C}$. Thus ρ' is formed by increasing the number of repetition of the cycle $\mathcal{C}$ in $\rho^\bullet$, such that $\rho' \not\models \psi_2$. □

Therefore we conclude that d is a bounded integer variable. We now have another main theorem.

Theorem 4. *When B is infinite, $\mathcal{A} \models \mathcal{D}$ if and only state $\mathcal{F}'$ cannot be reached in $\mathcal{G} \parallel \mathcal{S}^+$.*

4 Case Study

We now use a simplified automated container system to illustrate our techniques. We assume there are infinite number of containers to be transported from a ship to a yard. One quay crane (QC) and two track cars (TC) are used to unload these containers. As to a container in the ship, the QC first transports it to an idle TC, then the TC delivers it to the yard. It takes 5 time units for a QC to move down and pick up a container, then it will wait until one of the TCs is idle. If either of the TCs is idle, the QC spends 3

time units unloading the container to the idle TC, and 10 time units to get back to its initial position to handle the next container. Once a TC receives a container, it needs 15 time units to finish the delivery to the yard. Since the QC is a heavy equipment, it is expected that the utilization of the QC is higher. We thus have the requirement that the accumulated time of the QC waiting for an idle TC is at most one twentieth of the time in any interval.

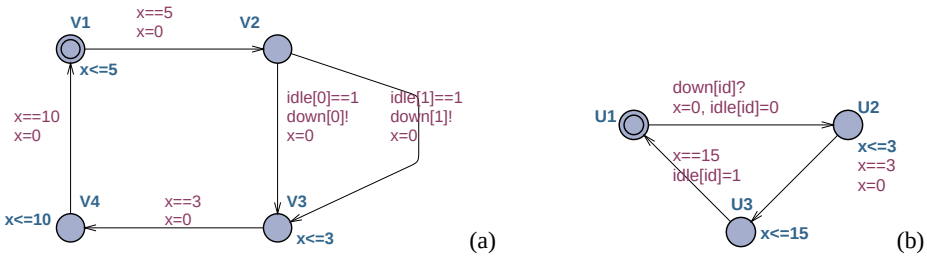

Fig. 6. Automated container system: (a) QC automaton (b)TC automaton

The automata QC and TC are shown in Fig 6. V2 is the location at which QC waits for an idle TC. The boolean variable `idle`[`i`] is used to indicate whether TC[`i`] is idle (it takes value 1 when idle) or not with an initial value 1. The urgent channel `down`[`i`] ensures that if the QC is at location V2, and as soon as TC[`i`] is idle, then the container unloading is done immediately. The whole system is $\mathcal{A}$ =QC$\|$ TC[0]$\|$ TC[1]. The above requirement can be easily specified by the following DC formula:

$$\ell > 0 \Rightarrow \int(V2, \times_2, \times_3) \leq 0.05\ell \tag{4}$$

The above formula can be easily transformed to the following LDI:

$$D : \ell > 0 \Rightarrow 19\int(V2, \times_2, \times_3) - \int(\times_1, \times_2, \times_3) \leq 0 \tag{5}$$

The *critical* location vector is $s_1 = (V2, \times_2, \times_3)$ and $Q = 2000$. We also declare `system` QC0,TC0,TC1 $< \mathcal{S}$. Following the techniques in Section 3, the transformed QC, the auxiliary automaton $\mathcal{S}$ and the function `accum`() are given in Fig.7 and Fig.8.

The failure state is specified as C : `A[] not` $d > 0$. We checked whether the failure state is never reached with UPPAAL and got $\mathcal{G} \parallel \mathcal{S} \models C$. Therefore, we have $\mathcal{A} \models D$. If the unloading time from a TC to the yard is changed to a big value, e.g, 35, C does not hold any more and UPPAAL can generate a counter-example.

This case can also be extended to a more complicated system with more QCs and TCs. However, the transformed automata and $\mathcal{S}$ does not change if the LDI remains unchanged.

5 Conclusion

This paper studies the problem of automatic verification of a timed automaton against an LDI. To solve this problem, several algorithms have been proposed in the literature

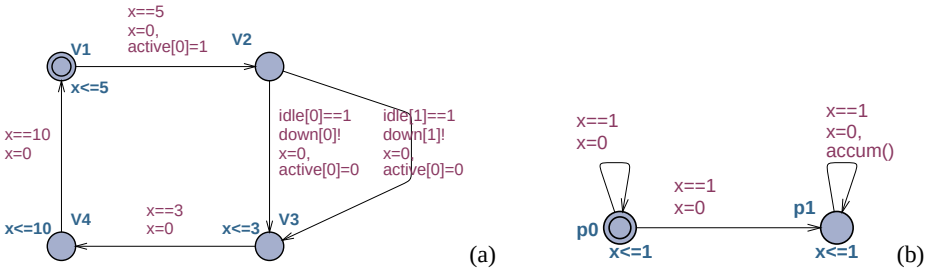

Fig. 7. Automated container system: (a) Transformed QC automaton (b) $\mathcal{S}$ automaton

```
void accum()
{    gc = (gc < A?gc + 1 : A)
     if active[0] == 1
          d = (gc ≥ A ∧ d > M?M + 1 : d + 19)
     d = (gc ≥ A ∧ d < M − Q?d : d − 1)
}
```

Fig. 8. Function `accum`()

[3, 10]. For improving the complexity the algorithm proposed in [11] is restricted to the class of the so-called *digitalized properties*. However, these model checking algorithms can only apply to one automaton, and cannot deal with the case when B is infinite in an efficient way. In addition, there is no available tool to support these algorithms. Recently some works have been done [17–19] for developing model checking tools for Duration Calculus. However, to our knowledge, compared with the model checkers of other temporal logics, the tools are still not widely applicable in industrial fields.

In [16], we give an algorithm to reduce model checking an LDI to model checking a CTL formula. The basic idea of that algorithm is: Instead of checking an automaton $\mathcal{A}$ against the LDI directly, we first construct an untimed model $\mathcal{H}$ from $\mathcal{A}$ and the LDI, and then construct a CTL formula ϕ from the LDI and then use a popular model checker, such as UPPAAL or SPIN, to check if $\mathcal{H} \models \phi$. This technique is simple and works well for one automaton. However, in order to apply the approach to real-time systems modelled as a network of automata with a common set of clocks and actions, we have to construct a composite automaton of the network explicitly. Obviously, it is not always feasible to manually construct such a composite automaton because of the high complexity and mistakes may be inevitable.

To avoid the construction of composite automata, in this paper, we have presented a different approach to this problem. We first construct a network of automata $\mathcal{G}$ to record the entry end exit of the *p-critical* locations. The construction of each $\mathcal{G}_i$ is very similar to the automaton $\mathcal{A}_i$ itself, and is simpler than the transformed model $\mathcal{H}$ proposed in [16]. As we need to check from any reachable state, when the antecedent of the LDI is true, whether or not the consequent is true, we then introduce $\mathcal{S}$ to count the observation time gc and the sum of the durations of the critical locations d from any reachable state.

The trick that $\mathcal{S}$ can stay in the initial state for arbitrary time units ensures that $\mathcal{S}$ starts the calculation of gc and d from any reachable state of $\mathcal{G}$. Also, the introduction of $\mathcal{S}$ is convenient for the user to simplify the specification. Without this, more extra variables and channels need to be introduced in the transformed network $\mathcal{G}$ and more complex expression of temporal logic needs to be defined. Finally, we define a failure state such that $\mathcal{A} \models \mathcal{D}$ if and only if this failure state is never reached in $\mathcal{S}$. Such checking can be done by some popular model checkers like UPPAAL.

In our future work, we will implement a tool that integrates the construction of $\mathcal{S}$ and $\mathcal{G}$. This tool is able to transform a `xml` file that has been constructed in UPPAAL to describe the original automata $\mathcal{A}$ to two `xml` files that describe respectively transformed $\mathcal{S}$ and $\mathcal{G}$. Then UPPAAL uses these two files as the input to do the checking. In this way, the checking of an LDI can be done without manually constructing the transformed automata. These will help to make Duration Calculus more applicable in practical applications.

The other direction of the future work is to apply the technique and tool to schedulability analysis and scheduler synthesis. There have been other approaches to formalising real-time scheduling, for instance, in [20] TLA is used to specify a system and analyze the schedulability of the system by proving that the system and the scheduler satisfy the given scheduling constraint. Using the Duration Calculus, Zhou Chaochen et al formalized a well-established scheduler EDF [22] and defined the semantics of scheduled programs [21]. We believe that these techniques will be useful in our future work on real-time scheduling analysis and synthesis, based on model checking DC properties.

Acknowledgment We are grateful to Anders P. Ravn for his comments on how to improve the presentation, and in particular for his suggestion on the algorithm of the calculation of the sum of the durations that simplified the algorithm in an earlier version of the paper. Without his help, this paper would not have become its present form.

References

1. R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*. 126(2): 183-235. 1994.
2. R. Alur. Timed Automata. In *Proc. CAV'99*, LNCS 1633, pp. 8-22. 1999.
3. V. A. Braberman and Dang Van Hung. On Checking Timed Automata for Linear Duration Invariants. In *Proc. RTSS'98*, pp. 264-273. IEEE Computer Society Press. 1998.
4. E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *Journal of the ACM* 33(1): pp. 151-178. 1986.
5. T. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. ICALP'92*, LNCS 623, pp. 545-558. 1992.
6. G. Chakravorty and P. K. Pandya. Digitizing Interval Duration Logic. In *Proc. CAV'03*, LNCS 2725, pp:167-179. 2003.
7. D. Bosnacki. Digitization of timed automata. In *Proc. FMICS'99*, pp 283-302. 1999.
8. Dang Van Hung and Phan Hong Giang. Sampling Semantics of Duration Calculus.In *FTRTFT 1996*, LNCS 1135, pp 188-207. 1996.
9. M. Franzle. Model-checking dense-time Duration Calculus. *Formal Asp. Comput. 16(2)*, pp 121-139. 2004.

10. X. Li and Dang Van Hung. Checking Linear Duration Invariants by Linear Programming. In *Proc. ASIAN'96*, LNCS 1179, pp. 321-332. 1996.
11. Pham Hong Thai and Dang Van Hung. Verifying Linear Duration Constraints of Timed Automata. In *Proc. ICTAC 2004*, LNCS 3407, pp.295-309. 2004.
12. Zhou Chaochen, C.A.R. Hoare and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276. 1991.
13. C. Zhou, J. Zhang, L. Yang and X. Li. Linear Duration Invariants. In *FTRTFT 1994*, LNCS 863, pp. 86-109. 1994.
14. C. Zhou and M. R. Hansen. *Duration Calculus. A Formal Approach to Real-Time Systems*. 2004.
15. G. Behrmann, A. David and K.G. Larsen. A tutorial on Uppaal. In *Proc. SFM-RT'04*, LNCS 3185, pp. 200-236. 2004.
16. M. Zhang, Dang Van Hung and Z. Liu. Verification of Linear Duration Invariants by Model Checking CTL Properties. In *Proc. ICTAC 2008*, LNCS 5160, pp. 395-410. Springer, Heidelberg (2008).
17. P. K. Pandya. Interval Duration Logic: Expressiveness and Decidability. ENTCS 65(6). 2002.
18. R. Meyer, J. Faber and A. Rybalchenko. Model Checking Duration Calculus: A Practical Approach. In *Proc. ICTAC 2006*, LNCS 4281, pp.332-346. 2006.
19. M. Fränzle and M. R.Hansen. Deciding an Interval Logic with Accumulated Durations. In *Proc. TACAS 2007*, LNCS 4424, pp. 201-215. 2007.
20. Z. Liu and M. Joseph. Specification and Verification of Fault-Tolerance, Timing, and Scheduling. *ACM Trans. Program. Lang. Syst*. 21(1): 46-89. 1999.
21. C. Zhou, M. R. Hansen, A. P. Ravn and H. Rischel. Duration Specifications for Shared Processors. In *Proc. FTRTFT'92*, LNCS 571, pp. 21-32. 1991.
22. Y. Zheng and C. Zhou. A Formal Proof of the Deadline Driven Scheduler. In *Proc. FTRTFT 1994*, LNCS 863, pp.756-775. 1994.