

Technical report 13-007

Finite Abstractions of Nonautonomous Max-Plus-Linear Systems*

D. Adzkiya, B. De Schutter, and A. Abate

To cite this work, please refer to the published version:

D. Adzkiya, B. De Schutter, and A. Abate, “Finite abstractions of nonautonomous max-plus-linear systems,” *Proceedings of the 2013 American Control Conference*, Washington, DC, pp. 4393–4398, June 2013.

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.24.73 (secretary)
URL: <https://www.dcsc.tudelft.nl>

* This report can also be downloaded via <https://dpub.eu/13-007>

Finite Abstractions of Nonautonomous Max-Plus-Linear Systems

Dieky Adzkiya, Bart De Schutter, and Alessandro Abate

Abstract—This work puts forward a technique to generate finite abstractions of nonautonomous Max-Plus-Linear (MPL) models, a known class of discrete-event systems characterizing the timing related to event counters. Nonautonomous models embed an external input (namely a nondeterministic choice, regarded as an exogenous control signal) in the dynamics. Abstractions are characterized as finite-state Labeled Transition Systems (LTS). LTS are obtained first by partitioning the state space of the MPL model and by associating states of the LTS to the introduced partitions, then by defining relations among the states of the LTS, corresponding to the dynamical (nonautonomous) transitions between the MPL state partitions, and finally by labeling the LTS edges according to the one-step timing properties related to the events of the original MPL model. In order to establish formal equivalences, the finite LTS abstraction is proven either to simulate or to bisimulate the original MPL model. The computational performance of the abstraction procedure is tested on a numerical benchmark. The approach enables the study of properties of the original MPL model by verifying equivalent specifications over the finite LTS abstraction.

I. INTRODUCTION

Max-Plus-Linear (MPL) systems are a class of discrete-event models [1], [2] with a continuous state space characterizing the timing of the underlying discrete events. MPL models are predisposed to easily describe synchronization between processes, under the assumption that timing events are linearly dependent (within a proper algebra) from previous event occurrences and (when nonautonomous, cf. Section II) from exogenous schedules. MPL models are widely employed in the formal study and scheduling of infrastructure networks such as communication and railway systems [3], or production and manufacturing lines [4], [5]. They can not model concurrency and are equivalent to a subclass of Petri nets, namely Timed-Event Graphs [6].

Classical dynamical analysis of MPL models is grounded on their algebraic [7] or geometric features [8]. It allows to derive conclusions on model properties, such as transient times, periodic regimes or ultimate behavior. This work investigates an alternative analytical approach aimed at checking whether trajectories of the MPL model verify dynamical properties, which is based on finite-state abstractions. It extends the recent contribution in [9], which focused on autonomous (namely, deterministic) models by embedding

a control input (that is, a nondeterministic choice) within the dynamics (cf. Section II). Furthermore, it makes use of a novel representation of the quantities into play (regions over state and control spaces, as well as dynamics), which allows for computationally agile operations.

With regards to the abstraction procedure (cf. Section III), we put forward a new technique that generates a finite-state Labeled Transition System (LTS) in a finite number of steps. The technique embeds the one-step dynamics within an “augmented space” [3], namely the cross product of state and input spaces, and furthermore leverages a region representation based on Difference-Bound Matrices (DBM) [10]. This representation allows for computationally fast operations on regions of the space, and thus for fast computation of the quantities of interest. The states of the LTS are obtained by finite partitioning of the state space of the MPL model, whereas relations between pairs of LTS states are established by checking whether a trajectory of the original MPL model is allowed to transition between the corresponding partition regions, possibly according to an input choice. Computationally, this characterization is performed by forward-reachability analysis over a piece-wise affine (PWA, i.e. linear plus constant, over regions of the state space) representation of the MPL dynamics. Finally, the labels of the LTS model are quantities defined in a number of possible ways: they either designate the difference between the timing of an event for any two variables of the original model, or represent the time difference between consecutive events of the MPL model. As argued later, this plurality of definitions allows for flexibility in the use of the LTS abstraction. We prove that in general the LTS abstraction simulates a transition system alternatively expressing the original MPL model, and raise sufficient conditions to establish a bisimulation relation between the abstract and concrete models [11].

The computational aspects related to the abstraction procedure have been of particular concern, and have brought 1) to select DBM as a framework for representing regions of state and control spaces; and 2) to employ a PWA representation of the MPL dynamics. The overall performance of the abstraction procedure is benchmarked on a case study in Section IV.

By expressing general dynamical MPL properties as specifications in Linear Temporal Logic (LTL, a modal logic), we argue that the LTS abstraction allows for the formal verification of system properties by use of model checking techniques. Moreover, it may lead to further computational savings derived from the computation of simulation and bisimulation relations of the LTS abstraction [11].

This research is funded by the European Commission under the MoVeS project, FP7-ICT-2009-5 257005, by the European Commission under the NoE FP7-ICT-2009-5 257462, by the European Commission under Marie Curie grant MANTRAS PIRG-GA-2009-249295, and by NWO under VENI grant 016.103.020.

The authors are with the Delft Center for Systems and Control, TU Delft – Delft University of Technology, The Netherlands - {d.adzkiya,b.deschutter,a.abate}@tudelft.nl

The abstraction technique developed in this work has been implemented as a MATLAB software tool, “Verification via biSimulations of MPL models” (VeriSiMPL, as in “very simple”), which is freely available for download [12].

The article is structured as follows. Section II introduces the MPL model framework, as well as additional theoretical concepts required for the article. Section III unfolds the abstraction procedure that derives an LTS from the MPL model – it formally constructs states, transitions, and labels of the LTS from the MPL dynamics. A running example elucidates the concepts and visualizes the procedure throughout the work. Section IV tests the abstraction technique on a computational benchmark, while Section V offers a few verification instances over the MPL model that can be studied by using the abstract LTS.

II. MODELS AND PRELIMINARIES

A. Nonautonomous Max-Plus-Linear Systems

Define \mathbb{R}_ε , ε and e respectively as $\mathbb{R} \cup \{\varepsilon\}$, $-\infty$ and 0 . A vector with each component equal to 0 (or $-\infty$) is also denoted by e (resp., ε). For $x, y \in \mathbb{R}_\varepsilon$, we define $x \oplus y = \max\{x, y\}$ and $x \otimes y = x + y$. For matrices $A, B \in \mathbb{R}_\varepsilon^{m \times n}$, $C \in \mathbb{R}_\varepsilon^{n \times p}$, then $[A \oplus B](i, j) = A(i, j) \oplus B(i, j)$ and $[A \otimes C](i, j) = \bigoplus_{k=1}^n A(i, k) \otimes C(k, j)$. Notice the analogy between \oplus , \otimes and $+$, \times for matrix and vector operations in standard algebra. Given $r \in \mathbb{R}$ (or $m \in \mathbb{N}$), the max-algebraic power of $x \in \mathbb{R}$ (resp., $A \in \mathbb{R}_\varepsilon^{n \times n}$) is denoted by $x^{\otimes r}$ (or $A^{\otimes m}$) and corresponds to rx in conventional algebra (resp., $A \otimes \dots \otimes A$, m times). Notice that $A^{\otimes 0} = E_n$, where E_n is a max-plus identity matrix, i.e. $E_n(i, i) = e$ and $E_n(i, j) = \varepsilon$, for $1 \leq i \neq j \leq n$.

A nonautonomous MPL model [3, Sec. 7.3] is defined as:

$$x(k) = A \otimes x(k-1) \oplus B \otimes u(k), \quad (1)$$

where $A \in \mathbb{R}_\varepsilon^{n \times n}$, $B \in \mathbb{R}_\varepsilon^{n \times m}$, $x(k-1) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}^m$, for $k \in \mathbb{N}$. The independent variable k denotes an increasing discrete-event counter, whereas the state variable x defines the (continuous) timing related to the discrete events. Furthermore, the state space is taken to be \mathbb{R}^n , which also implies that the state matrix A has to be row-finite.

Definition 1 (Regular (row-finite) matrix, [3, Sec. 1.2]):

A matrix is called regular (or row-finite) if it contains at least one element different from ε in each row. \square

As suggested in [6, Sec. 2.5.4], a nonautonomous MPL model (1) can be expressed as an augmented MPL model:

$$x(k) = \bar{A} \otimes \bar{x}(k-1), \quad (2)$$

where $\bar{A} = [A \ B]$, $\bar{x}(k-1) = [x(k-1)^T \ u(k)^T]^T$.

Example: Consider the modified two-dimensional MPL model from [3, Sec. 0.1]:

$$x(k) = \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix} \otimes x(k-1) \oplus \begin{bmatrix} e & \varepsilon \\ \varepsilon & e \end{bmatrix} \otimes u(k). \quad (3)$$

The augmented MPL model is simply

$$x(k) = \begin{bmatrix} 2 & 5 & e & \varepsilon \\ 3 & 3 & \varepsilon & e \end{bmatrix} \otimes \bar{x}(k-1), \quad (4)$$

where $x(k) \in \mathbb{R}^2$ and $\bar{x}(k-1) \in \mathbb{R}^4$, for $k \in \mathbb{N}$. \square

B. Piecewise-Affine Systems

This paragraph discusses the PWA system generated by a nonautonomous MPL model: each nonautonomous MPL model (1) can be expressed as a PWA system in the event domain [13] by leveraging its representation as an augmented MPL model. Each region, along with its affine dynamics, is characterized by $\bar{g} = (\bar{g}_1, \dots, \bar{g}_n) \in \{1, \dots, n+m\}^n$ or, more precisely, as:

$$\bar{R}_{\bar{g}} = \bigcap_{i=1}^n \bigcap_{j=1}^{n+m} \{\bar{x}_j - \bar{x}_{\bar{g}_i} \leq \bar{A}(i, \bar{g}_i) - \bar{A}(i, j)\}; \quad (5)$$

$$x_i(k) = \bar{x}_{\bar{g}_i}(k-1) + \bar{A}(i, \bar{g}_i), \quad 1 \leq i \leq n. \quad (6)$$

C. Difference-Bound Matrices

This section introduces the definition of a DBM [10, Sec. 4.1] and of its canonical-form representation. The DBM will be used in Section III to represent the regions, the set of inputs, as well as the labels.

Definition 2 (Difference-Bound Matrix): A DBM in the state space \mathbb{R}^n is the finite intersection of sets defined by $x_i - x_j \bowtie_{\alpha_{i,j}} \alpha_{i,j}$, where $\bowtie_{\alpha_{i,j}} \in \{<, \leq\}$, $\alpha_{i,j} \in \mathbb{R} \cup \{+\infty\}$, for $1 \leq i \neq j \leq n$. \square

Definition 2 can be customized over the augmented space \mathbb{R}^{n+m} of the MPL model. For each DBM, there exists an equivalent and unique canonical-form representation [10, Th. 2], which is a DBM with the tightest possible bounds [10, Observation 1]. As intuitive, a DBM can be represented by a matrix D (namely, a “potential graph” [14, Sec. 3]) with elements $D(i, j)$ that are intervals relating the pair (x_i, x_j) by the element $\alpha_{i,j}$ and by the operator $\bowtie_{\alpha_{i,j}}$. The Floyd-Warshall algorithm can be used over the potential graph to compute the canonical-form representation [10, Sec. 4.1].

One advantage of the canonical-form representation is that it is easy to compute orthogonal projections w.r.t. a subset of its variables: this is simply performed by deleting the rows and the columns corresponding to the complementary variables [10, Sec. 4.1].

Definition 3 (Orthogonal projection): The orthogonal projection w.r.t. state variables (input variables) is defined as $\Pi_X : \mathbb{R}^{n+m} \rightarrow \mathbb{R}^n$ ($\Pi_U : \mathbb{R}^{n+m} \rightarrow \mathbb{R}^m$), where $\Pi_X : [x_1 \dots x_n]^T \mapsto [x_1 \dots x_n]^T$ ($\Pi_U : [x_1 \dots x_n]^T \mapsto [u_1 \dots u_m]^T$). \square

Notice that from (5) and (6), each region and corresponding affine dynamics of the PWA system generated by a nonautonomous MPL model can be characterized by a DBM in \mathbb{R}^{n+m} and \mathbb{R}^{2n+m} , respectively. The following holds.

Theorem 1: The image and the inverse image of a DBM w.r.t. the affine dynamics (6) is a DBM. \square

Proof: The general procedure to compute the image and the inverse image of a DBM w.r.t. affine dynamics involves: 1) computing the intersection of the given DBM with a DBM generated by the expression of the affine dynamics; then 2) calculating the canonical form of the intersection; finally 3) projecting the canonical-form representation over a subset of its variables. The claim follows by noticing that the intersection of two DBM is a DBM, that the canonical

form of a DBM is a DBM and that the orthogonal projection of a DBM is again a DBM. \square

D. Labeled Transition Systems

Definition 4 (Labeled Transition System, [11, Def. 2.1]):

An LTS (S, L, δ, I, AP) consists of a set S of states, a set L of labels, a transition relation $\delta \subseteq S \times L \times S$, a set $I \subseteq S$ of initial states, and a set AP of atomic propositions. \square

Denote with $\mathcal{P}(S)$ the power set of a given set S . This work considers a special class of LTS. To begin with, often we will assume that $I = S$ and $AP = \mathbb{R}^n$, where the dimensions correspond to the size of the state of the MPL model. Moreover the set of labels, customarily taken to be discrete and finite, is here assumed to be $L = \mathcal{P}(AP)$ — in other words, the labels will be finite unions of n -dimensional vectors of real numbers or of real-valued intervals.

III. LTS ABSTRACTIONS OF NONAUTONOMOUS MPL MODELS

Given a set of inputs, we abstract a nonautonomous MPL model into a finite-state LTS. The states of the LTS are obtained by partitioning the continuous state space, whereas its transitions depend also on the set of input signals. The labels of the LTS are defined in a few distinguished manners and depend on the delay (timing) features of the MPL model. Technically, the abstraction procedure makes use of the augmented model, of its dynamics represented via PWA equations, and of the DBM representation for regions over the state, input and augmented spaces.

A. States: Partitioning Procedure

1) *Partitioning via Autonomous Model:* We use the partitioning procedure in [9, Sec. III-A2], presented in Algorithm 1. The partitions are based on the underlying dynamics of the autonomous part of the MPL model (notice that the algorithm does not use matrix B), and each partitioning region is a DBM, which allows exploiting specific computational results (cf. Section II-C).

Example: Consider the autonomous version of (3). The regions generated by the scheme in Algorithm 1 are $R_1 = \{x \in \mathbb{R}^2 : x_1 - x_2 < e\}$, $R_2 = \{x \in \mathbb{R}^2 : e < x_1 - x_2 < 3\}$, $R_3 = \{x \in \mathbb{R}^2 : x_1 - x_2 = e\}$, $R_4 = \{x \in \mathbb{R}^2 : x_1 - x_2 > 3\}$, $R_5 = \{x \in \mathbb{R}^2 : x_1 - x_2 = 3\}$. The regions are shown in Fig. 1. \square

In order to exploit additional steps of the partitioning procedure and to streamline the whole process, we introduce next an alternative approach to generate a partitioning of the state space. We then argue that the partitioning generated by the alternative approach is finer than that obtained from Algorithm 1, which leads to conclude that Algorithm 1 is computationally more attractive than the alternative approach.

2) *Partitioning via Nonautonomous Model:* It is possible to leverage the nonautonomous MPL model to obtain a partition of the state space: this is achieved by computing the orthogonal projection w.r.t. the state variables of each region in the PWA system generated by the nonautonomous

Algorithm 1: Generation of state-space partition via autonomous model

input : $A \in \mathbb{R}_\varepsilon^{n \times n}$, a row-finite max-plus matrix
output: R , a collection of partitioning regions

```

1  $R \leftarrow \emptyset$ ;
2 foreach  $(f_1, \dots, f_n) \in (\mathcal{P}(\{1, \dots, n\}) \setminus \emptyset)^n$  do
3    $R_f \leftarrow \mathbb{R}^n$ ;
4   foreach  $1 \leq r \leq n$  do
5     foreach  $1 \leq i \neq j \leq n$  do
6       if  $i, j \in f_r$  then
7          $R_f \leftarrow R_f \cap \{x_i - x_j =$ 
8            $A(r, j) - A(r, i)\}$ ;
9       else if  $i \in f_r, j \notin f_r$  then
10         $R_f \leftarrow R_f \cap \{x_i - x_j >$ 
11           $A(r, j) - A(r, i)\}$ ;
12      end
13    end
14  if  $R_f$  is not empty then  $R \leftarrow R \cup \{R_f\}$ ;
15 end

```

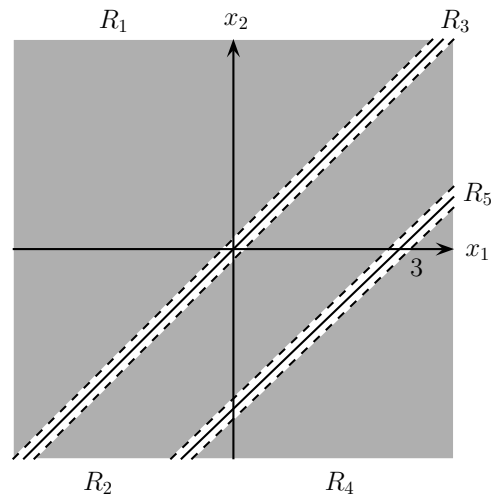


Fig. 1. Partition of \mathbb{R}^2 for the model in (3), as generated by Algorithm 1

Function Refine(R'): Generation of state-space partitioning via refinement of state-space cover

input : R' , a cover of \mathbb{R}^n where $|R'| < \infty$
output: R , a partition of \mathbb{R}^n

```

1  $R \leftarrow \emptyset$ ;
2 foreach  $R'' \in \mathcal{P}(R')$  do
3    $R \leftarrow R \cup \left\{ \bigcap_{R'' \in R'} R'' \setminus \bigcup_{R'' \in R \setminus R'} R'' \right\}$ ;
4 end
5 remove the empty region from  $R$ ;

```

MPL model, thereafter refining the projection as needed to obtain a partition. The approach is detailed in Algorithm 2.

It can be guaranteed that the number of obtained partitioning regions over the state space is finite and that each

Algorithm 2: Generation of state-space partition via nonautonomous model

input : $\bar{A} \in \mathbb{R}_\varepsilon^{n \times (n+m)}$, an augmented MPL model
output: R , a partition of \mathbb{R}^n

- 1 $\bar{R} \leftarrow$ regions of the PWA system generated by \bar{A} ;
 - 2 $R \leftarrow \emptyset$;
 - 3 **foreach** $\bar{R} \in \bar{R}$ **do** $R \leftarrow R \cup \{\Pi_X(\bar{R})\}$;
 - 4 $R \leftarrow \text{Refine}(R)$;
-

partitioning region is a DBM in \mathbb{R}^n . However, we next argue that the state-space partition obtained from Algorithm 2 is in general finer (hence, more expensive) than that resulting from Algorithm 1.

Let us characterize the orthogonal projection w.r.t. the state variables of $\bar{R}_{\bar{g}}$, which we denote by $\Pi_X(\bar{R}_{\bar{g}})$, under two special cases: 1) each inequality of $\bar{R}_{\bar{g}}$ contains at least one state variable, i.e. $\bar{g} \in \{1, \dots, n\}^n$; and 2) each inequality of $\bar{R}_{\bar{g}}$ contains at least one input variable, i.e. $\bar{g} \in \{n+1, \dots, n+m\}^n$. The following two lemmas hold.

Lemma 1: For each $\bar{g} \in \{1, \dots, n\}^n$,

$$\Pi_X(\bar{R}_{\bar{g}}) = \bigcap_{i=1}^n \bigcap_{j=1}^n \{x_j - x_{\bar{g}_i} \leq A(i, \bar{g}_i) - A(i, j)\} = R_{\bar{g}}. \quad \square$$

Lemma 2: For each $\bar{g} \in \{n+1, \dots, n+m\}^n$, if $\bar{R}_{\bar{g}}$ is not empty, then $\Pi_X(\bar{R}_{\bar{g}}) = \mathbb{R}^n$. \square

Lemma 1 implies that each region of the PWA system generated by the autonomous MPL model [9, Algorithm 1] is included in the orthogonal projection w.r.t. the state variables of regions in the PWA system generated by the nonautonomous MPL model (Algorithm 2 without refinement). This leads directly to the following conclusion.

Proposition 1: The state space partition generated by Algorithm 2 is finer than that obtained from Algorithm 1. \square

It is only for special instances that the two approaches lead to equivalent outcomes, as argued in the following theorem. In this case, Lemmas 1 and 2 can be used to show that the projection of each region in the PWA system generated by the nonautonomous MPL model is a finite union of regions of the PWA system generated by its autonomous version.

Theorem 2: If there is at most one finite element in each column of the matrix B , the partitions constructed by Algorithm 2 coincide with those obtained by Algorithm 1. \square

B. Transitions: Forward Reachability

Next, we investigate a technique to determine the transition relations δ between two partitioning regions. At any given event step k , given a pair of partitioning regions R and R' and a set of inputs $U \subseteq \mathbb{R}^m$, there exists a transition from R to R' iff there exists an $x(k-1) \in R$ and a $u(k) \in U$, such that $x(k) \in R'$. Such a transition can be determined either by the forward- or by the backward-reachability computation, that is calculating either $R' \cap \{x(k) : x(k-1) \in R, u(k) \in U\}$, or $R \cap \{x(k-1) : x(k) \in R', u(k) \in U\}$, and by checking the non-emptiness of the resulting set.

Algorithm 3: Computation of the transitions via forward-reachability analysis

input : $\bar{A} \in \mathbb{R}_\varepsilon^{n \times (n+m)}$, a row-finite augmented matrix;

R , a partition of \mathbb{R}^n ;

\bar{R} , a cover of \mathbb{R}^{n+m} where $|\bar{R}| < \infty$;

U , the input set

output: $\delta \subseteq R \times R$, a transition relation

- 1 $\delta \leftarrow \emptyset$;
 - 2 **foreach** $R, R' \in R$ **do**
 - 3 **if** $\exists R'' \in \mathcal{I}(R)$ s.t. $R'' \cap R'$ is not empty **then**
 - 4 $\delta \leftarrow \delta \cup \{(R, R')\}$;
 - 5 **end**
 - 6 **end**
-

We assume that the set of inputs $U \subseteq \mathbb{R}^m$ is represented via a DBM. Practically, this enables expressing upper or lower bounds on the separation between input events (schedules) (cf. Definition 2). If on the other hand there are no constraints on input events, we define $U = \mathbb{R}^m$, which is also a DBM.

We opt to focus on the forward-reachability approach, since it is computationally more attractive than the backward one: more precisely, the number of image computations in the forward-reachability approach is in general less than the number of inverse-image computations in the backward-reachability approach.

Given a partitioning region $R \in R$, we employ the PWA representation over the augmented space to compute its image:

$$\mathcal{I}(R) = \{\bar{A} \otimes \bar{x} : \bar{x} \in R \times U\},$$

where $R \times U$ denotes the cross product of the sets R and U . Since both R and U are DBM, $R \times U$ is also a DBM in the augmented space. Computing the image of this DBM can be done by applying the procedure in Theorem 1, while using the PWA representation of \bar{A} . Algorithm 3 details the complete approach.

The outcome of Algorithm 3 is in general a nondeterministic transition system. Its relationship with the nonautonomous MPL model is clear: the transition system obtained by Algorithm 3 simulates a transition system alternatively expressing the nonautonomous MPL model [11], whereas in general the opposite direction does not hold. In order to provide sufficient conditions to obtain a bisimulation relation [11], we employ a backward-reachability analysis over the augmented MPL model:

$$\mathcal{I}^{-1}(R') = \{\bar{x} \in \mathbb{R}^n \times U : \bar{A} \otimes \bar{x} \in R'\}. \quad (7)$$

The general procedure for obtaining $\mathcal{I}^{-1}(R')$ is applying the procedure in Theorem 1 by using the PWA representation of \bar{A} . Notice that $\Pi_X(\mathcal{I}^{-1}(R'))$ represents the set of states that are able to transition into R' or, more formally $\{x \in \mathbb{R}^n : \exists u \in U \text{ s.t. } A \otimes x \oplus B \otimes u \in R'\}$. This leads to the following result.

Proposition 2: The transition system obtained by Algorithm 3 bisimulates the nonautonomous MPL model if for each pair $(R, R') \in \delta$, the following holds: for each $x \in R$, there exists an input $u \in U$ such that $A \otimes x \oplus B \otimes u \in R'$. Equivalently, for each $(R, R') \in \delta$, $R \subseteq \Pi_X(\mathcal{I}^{-1}(R'))$. \square

Example: We introduce an input set U equals to R_2 . We are going to explicitly check the existence of a transition from R_4 to R_3 . First, we determine the PWA regions that intersect with $R_4 \times U = \{\bar{x} \in \mathbb{R}^4 : x_1 - x_2 > 3, 0 < u_1 - u_2 < 3\}$: after performing the checking of emptiness of intersections, we are left with the regions $\bar{R}_{(1,1)}$, $\bar{R}_{(3,1)}$, and $\bar{R}_{(3,3)}$. Thereafter, we determine whether the image of $\bar{R}_{(1,1)} \cap (R_4 \times U)$, $\bar{R}_{(3,1)} \cap (R_4 \times U)$, and $\bar{R}_{(3,3)} \cap (R_4 \times U)$ intersect with R_3 : after computing the sets, we derive that only the image of $\bar{R}_{(3,1)} \cap (R_4 \times U)$ intersects with R_3 . Thus, we conclude that there is a transition from R_4 to R_3 . The overall transition system is shown in Fig. 2, where transitions, for the moment, are defined with no labels. It can be checked that the LTS only simulates the nonautonomous MPL model: for instance, the pair (R_2, R_1) does not satisfy the condition in Proposition 2. \square

C. Labels: Backward Reachability

We now introduce labels on the transition system, thus obtaining a full LTS. Labels are quantities defined in two different possible ways: they either characterize 1) the difference between the timing of an event for any two variables of the original model, i.e. $x_i(k) - x_j(k)$, where $1 \leq i < j \leq n$; or 2) the time difference between consecutive events of the MPL model, i.e. $x_i(k) - x_i(k-1)$, for $1 \leq i \leq n$.

The first labeling characterizes the explicit representation of partitioning regions and is associated to each state in the LTS. Since each partitioning region is represented by a DBM in its canonical-form representation (cf. Section II-C), the bounds on the state labels are the tightest possible and the representation is unique [10, Th. 2]. A label is a finite vector of real-valued intervals.

The second labeling is associated to each transition in the LTS. In order to calculate the labels of a transition, first we employ a backward-reachability analysis (7) to the incoming partitioning regions, which yields a finite union of DBM. Then for each DBM, we collect information from its explicit representation and its affine dynamics (2). Thus, it is a finite union of vectors of real-valued intervals.

Example: We are going to determine the labels of the transition from R_4 to R_3 . First we calculate $(R_4 \times U) \cap \mathcal{I}^{-1}(R_3)$: after iterating through all PWA regions, we obtain a region described by $\{\bar{x} \in \mathbb{R}^4 : x_1 - x_2 > 3, x_1 - u_1 = -3, x_1 - u_2 \geq -3, x_2 - u_1 \leq -5, 0 < u_1 - u_2 < 3\} \subseteq \bar{R}_{(3,1)}$. The second label is obtained by using the explicit representation of the region and its affine dynamics: $x_1(k) - x_1(k-1) = u_1 - x_1 = 3$, and $x_2(k) - x_2(k-1) = x_1 - x_2 + 3 > 6$. Fig. 2 depicts the full LTS endowed with the second labeling. \square

IV. COMPUTATIONAL BENCHMARK

In order to test the practical efficiency of the proposed algorithms we compute the runtime needed to perform the

abstraction of an MPL system into an LTS, for an increasing dimension n of the MPL model. We also keep track of the number of states and of transitions of the obtained LTS. We assume that the control is a scalar, namely that $m = 1$. For any given n , we generate matrices A with 2 finite elements (in a max-plus sense) that are randomly placed in each row, as well as matrices B as column vectors where all the elements are finite. The finite elements are randomly generated integers with value in between 1 and 100. The input space is selected as $U = \mathbb{R}$.

The experiments have been run on MATLAB code, over a dual-core AMD Opteron 2.8 GHz PC with 8 GB of memory. Over 10 independent experiments, Table I (dealing with maximum and mean values) reports the time needed to construct the LTS, broken down over the three successive procedures for states, transitions, and label generation. The number of states and of transitions in the LTS are also computed.

Recall that the first step (generation of states) consists of the partitioning of the state space (Algorithm 1) and constructing the PWA system over the augmented space (cf. Section II-B), whereas the second (generation of transitions) uses the forward-reachability analysis. The generation of labels is limited to computing (via backward reachability) the labels over the transitions, since those over the states can be derived from the first step and as such require no extra computational overhead. The labeling procedure is computationally heavier than the generation of transitions – this is due to the computation step based on the backward reachability, which as discussed in Section III is in general more expensive than the forward reachability one.

The worst-case complexity of the abstraction procedure is exponential w.r.t. the dimension of the MPL model [9], however the benchmark displays that this conservative estimation in practice does not occur.

V. FORMAL VERIFICATION OF NONAUTONOMOUS MPL MODELS

The obtained LTS model can be employed to verify certain properties over the original MPL model. If the LTS bisimulates the original MPL model, we can verify a number of specifications, for instance those expressed in LTL logic [11]. Otherwise, if the LTS simulates the original MPL model, we can verify at least safety properties.

To specify properties for trajectories of the MPL model, we use Linear Temporal Logic (LTL) [11, Ch. 5]. LTL formulas are recursively defined over a set of atomic propositions (AP), by Boolean operators and temporal operators. Boolean operators are \neg (negation), \wedge (conjunction), and \vee (disjunction), whereas temporal operators are \bigcirc (next), \mathcal{U} (until), \square (always), and \diamond (eventually). A formula ϕ , which in general is (recursively) determined by application of the above operators, is interpreted over traces (trajectories) generated by the LTS. In particular it is of interest to check if (the trajectories of) an LTS satisfies a given formula (or “specification”) – this procedure is known as “model checking”.

Computer Science, B. Cook and A. Podelski, Eds. Springer Berlin
Heidelberg, 2007, vol. 4349, ch. 20, pp. 268–282.