VeriSiMPL: Verification via biSimulations of MPL models

Dieky Adzkiya * and Alessandro Abate **

Abstract. VeriSiMPL ("very simple") is a software tool to obtain finite abstractions of Max-Plus-Linear (MPL) models. MPL models (Sec. 2), specified in MATLAB, are abstracted to Labeled Transition Systems (LTS). The LTS abstraction is formally put in relationship with the concrete MPL model via a (bi)simulation relation. The abstraction procedure (Sec. 3) runs in MATLAB and leverages sparse representations, fast manipulations based on vector calculus, and optimized data structures such as Difference-Bound Matrices. LTS abstractions can be exported to structures defined in the PROMELA language. This enables the verification of MPL models against temporal specifications within the SPIN model checker (Sec. 4). The toolbox is available at

`http://sourceforge.net/projects/verisimpl/`

1 Motivations and Goals

Max-Plus-Linear (MPL) models are discrete-event systems [1],[5] with continuous variables that express the timing of the underlying sequential events. MPL models are employed to describe the timing synchronization between interleaved processes, and as such are widely employed in the analysis and scheduling of infrastructure networks, such as communication and railway systems [2] and production and manufacturing lines [1],[6]. They are related to a subclass of Timed Petri Nets, namely Timed-Event Graphs [1]. MPL models are classically analyzed by algebraic [7] or geometric techniques [8] over the max-plus algebra, which allows investigating properties such as transient and periodic regimes [1], or ultimate dynamical behavior [9]. They can be simulated via the max-plus toolbox Scilab [3].

The recent work in [4] has explored a novel, alternative approach to analysis, which is based on finite-state abstractions of MPL models. The objective of this new approach is to allow a multitude of available tools that has been developed for finite-state models to be employed over MPL systems. We are in particular interested downstream in the Linear Temporal Logic (LTL) model checking of MPL models via LTS abstractions. This article presents VeriSiMPL, a software toolbox that implements and tests the abstraction technique in [4].

2 Nuts and bolts of Max-Plus-Linear models

Define $\mathbb{R}_\varepsilon$ and ε respectively as $\mathbb{R} \cup \{\varepsilon\}$ and $-\infty$. For a pair $x, y \in \mathbb{R}_\varepsilon$, we define $x \oplus y = \max\{x, y\}$ and $x \otimes y = x + y$. Max-plus algebraic operations are extended to matrices as follows: if $A, B \in \mathbb{R}_\varepsilon^{m\times n}$ and $C \in \mathbb{R}_\varepsilon^{n\times p}$, then $[A \oplus B](i,j) = A(i,j) \oplus B(i,j)$ and $[A \otimes C](i,j) = \bigoplus_{k=1}^{n} A(i,k) \otimes C(k,j)$, for all i, j. An MPL model [1, Corollary 2.82] is defined as:

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

* The authors are with the Delft Center for Systems & Control, TU Delft.
** A. Abate is also with the Department of Computer Science, University of Oxford.

where $A \in \mathbb{R}_{\varepsilon}^{n\times n}$, $B \in \mathbb{R}_{\varepsilon}^{n\times m}$, $x(k) \in \mathbb{R}_{\varepsilon}^{n}$, $u(k) \in \mathbb{R}_{\varepsilon}^{m}$, for $k \in \mathbb{N}$. In this work, the state and input spaces are taken to be $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively: the independent variable k denotes an increasing discrete-event counter, whereas the n-dimensional state variable x defines the (continuous) timing of the discrete events and the m-dimensional input u characterizes external schedules. If the input matrix B contains at least a finite (not equal to ε) element, the MPL model is called *nonautonomous*, otherwise it is called *autonomous* since it evolves under no external schedule. Nonautonomous models embed nondeterminism in the form of a controller input.

Implementation: VeriSiMPL accepts MPL models written in MATLAB [10]. For practical reasons, the state matrix A is assumed to be row-finite, namely characterized in each row with at least one element different from ε.

Example: Consider the following two-dimensional autonomous MPL model from [1, p. 4], representing the scheduling of train departures from two connected stations $i = 1, 2$ (event k denotes the k-th departure at time $x_i(k)$ for station i):

$$x(k) = \begin{bmatrix} 3 & 7 \\ 2 & 4 \end{bmatrix} \otimes x(k-1), \text{ i.e. } \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \begin{bmatrix} \max\{3 + x_1(k-1), 7 + x_2(k-1)\} \\ \max\{2 + x_1(k-1), 4 + x_2(k-1)\} \end{bmatrix}.$$

In the Appendix the model is equivalently expressed as a Timed-Event Graph.

3 From MPL models to Labeled Transition Systems

We seek to construct a finite-state Labeled Transition System (LTS) as an abstraction of an (autonomous or nonautonomous) MPL model. An LTS is comprised of a finite set of states (Sec. 3.1), of a set of transitions relating pairs of states (Sec. 3.2), and is further decorated with labels on either states or transitions (Sec. 3.3). The obtained LTS is in general nondeterministic: its determinization can be attempted by state refinement procedures [4].

3.1 LTS states: partitioning of MPL space

LTS states are obtained by partitioning the state space $\mathbb{R}^n$ based on the underlying dynamics, that is based on the state matrix A [4, Algorithms 1,2]. The partition can be further refined (in order to seek a bisimulation of the concrete model) or otherwise coarsened by merging adjacent regions (in order to reduce the cardinality of the set of abstract states).

Implementation: VeriSiMPL implements two approaches [4, Algorithms 1,2]. In order to improve the performance of the procedure, standard pruning tricks [11, Sec. 3] are applied. Each generated region is shown to be a Difference-Bound Matrix (DBM) [12, Sec. 4.1]: this allows a computationally efficient representation based on the expression $x_i - x_j \bowtie \alpha_{i,j}, \bowtie \in \{<, \leq\}$. VeriSiMPL represents a DBM as a row cell with two elements: the first element is a real-valued matrix representing the upper bound $\alpha_{i,j}$, whereas the second is a Boolean matrix representing the value of $\bowtie$. A collection of DBM is also represented as a row with two elements, where the corresponding matrices are stacked along the third dimension. Quite importantly, DBM are closed under MPL operations.

Example: The partitioning regions generated for the MPL model in Sec. 2 are $R_1 = \{x \in \mathbb{R}^2 : x_1 - x_2 > 4\}$, $R_2 = \{x \in \mathbb{R}^2 : 2 < x_1 - x_2 \leq 4\}$, and $R_3 = \{x \in \mathbb{R}^2 : x_1 - x_2 \leq 2\}$. They are shown in Fig. 1 (left).

3.2 LTS transitions: forward-reachability analysis

An LTS transition between any two abstract states R and R' is generated based on the relation between the two corresponding partitioning regions. At any given event counter k, there is a transition from R to R' if there exists an $x(k-1) \in R$ and possibly a $u(k) \in U \subseteq \mathbb{R}^m$ such that $x(k) \in R'$. Such a transition can be determined by forward-reachability computation. We assume that the set of allowed inputs $U \subseteq \mathbb{R}^m$ is characterized via a DBM: practically, this expresses upper or lower bounds on the separation between input events (schedules). Under no constraints on input events we instead define $U = \mathbb{R}^m$, which is also a DBM.

Given a pair R, R', the procedure comprises: 1) the computation of the image of R and 2) the collection of the partitioning regions that intersect the image. The image of R is defined as $[A\ B] \otimes (R \times U)$, and in general is a finite union of DBM. As a special instance, for an autonomous MPL system (trivial B or $U = \emptyset$) the image is simply a DBM.

Implementation: VeriSiMPL performs forward-reachability by mapping and manipulating DBM. It represents a transition in MATLAB as a sparse Boolean matrix. As in a precedence graph [1, Definition 2.8], the (i, j)-th element equals to 1 if there is a transition from j to i, else it is equal to 0.

Example: The transitions for the model in Sec. 2 are represented in Fig. 1 (right). In a nonautonomous version of the model, the finite-state structure in Fig. 1 will simply present additional transitions.

3.3 LTS labels: fast manipulation of DBM

LTS labels are quantities associated with states or transitions and characterize

1) the difference between the timing of a single event (k) for any two variables of the original MPL 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 labels denote the difference of the timing of a single event k over two variables x_i, x_j, are determined by the representation of the partitioning regions (a DBM in $\mathbb{R}^n$) and are quantities associated with each state of the LTS [4].

The second labels represent time difference between consecutive events $(k-1$ and $k)$. They are quantities associated with transitions in the LTS. In order to compute the labels of a transition from R to R', first we calculate $\{[x^T\ u^T]^T \in R \times U : [A\ B] \otimes [x^T\ u^T]^T \in R'\}$ by employing a backward-reachability analysis of R' w.r.t. the system generated by $[A\ B]$, which yields a finite union of DBM in $\mathbb{R}^{m+n}$. For each obtained DBM, we substitute its dynamics into $x_i(k) - x_i(k-1)$, for $1 \leq i \leq n$, obtaining a finite union of vectors of real-valued intervals. As a special case, for an autonomous MPL system we compute $\{x \in R : A \otimes x \in R'\}$, which is a DBM: in this case each label is a vector of real-valued intervals.

Implementation: Practically, in both cases VeriSiMPL stores the labels as (unions of) vectors of real-valued intervals in MATLAB. In the second case the labels are computed by fast DBM manipulations.

Example: The obtained LTS can be expressed as a simple text file and parsed by Graphviz [13] for plotting, as displayed in Fig. 1 (right).

4 Computational Benchmark and Case Study

We have computed the runtime required to abstract an autonomous MPL system as a finite-state LTS, for increasing dimensions n of the MPL model, and

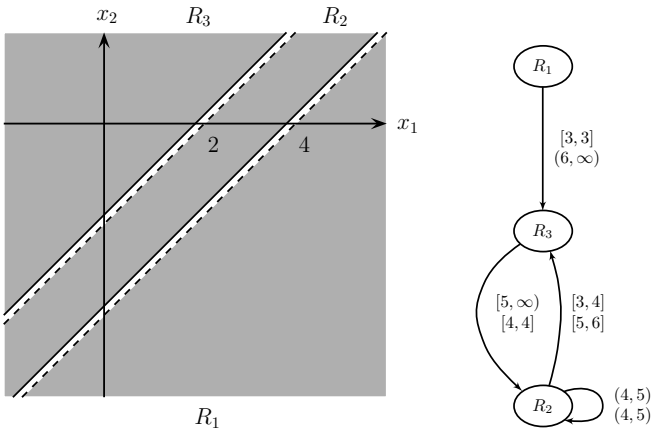

Fig. 1. LTS abstraction (right graph) of the MPL model in Sec. 2, inclusive of abstract states (associated to the partitioning regions on the left plot), transitions, and labels.

kept track of the number of states and of transitions of the obtained LTS (memory requirement). Compared to partition-based abstraction procedures in the literature for other classes of dynamical systems [14], the presented abstraction procedure comfortably manages MPL models with significant size.

Implementation: For any given n, we have generated row-finite matrices A with 2 finite elements (random integers taking values between 1 and 100) placed randomly in each row. The algorithms have been implemented in MATLAB 7.13 (R2011b) and the experiments have been run on a 12-core Intel Xeon 3.47 GHz PC with 24 GB of memory. For a 15-dimensional model, VeriSiMPL generates an LTS with about 10^4 states and 10^6 transitions, with a runtime limited within a few hours. The complete outcomes are reported in Table 1 in the Appendix.

Example: The obtained LTS can be exported to PROMELA (a PROcess MEta LAnguage), to be later used by the SPIN model checker [15]. We employ SPIN to modelcheck LTL properties over MPL models. Model checking can be performed over both state and transition labels, which adapts to the LTS formats we can generate with the abstraction.

As an example of verification of properties over transitions, consider the specification $\Psi : \forall k \in \mathbb{N}, \psi(k)$, where $\psi(k) := \{x_2(k+1) - x_2(k) \leq 6\}$. After a proper finite labeling of the LTS, Ψ can be expressed as $\Box\psi$. We obtain the satisfiability set $\mathrm{Sat}(\Psi) = \{R_2, R_3\}$.

Bibliography

1. Baccelli, F., Cohen, G., Olsder, G., Quadrat, J.P.: Synchronization and Linearity, An Algebra for Discrete Event Systems. John Wiley and Sons (1992)
2. Heidergott, B., Olsder, G., van der Woude, J.: Max Plus at Work–Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications. Princeton University Press (2006)
3. Plus, M.: Max-plus toolbox of Scilab [Online] (1998) Available at `www.cmap.polytechnique.fr/~gaubert/MaxplusToolbox.html`.
4. Adzkiya, D., De Schutter, B., Abate, A.: Abstraction and verification of autonomous max-plus-linear systems. In: Proc. 31st American Control Conf., Montreal, CA (Jun. 2012) 721–726 Available at `www.dcsc.tudelft.nl/~aabate/publications/AdzkiyaDeSchutterAbateACC12.pdf`.

Appendix

Testing VeriSiMPL

The toolbox is available at

`http://sourceforge.net/projects/verisimpl/`

The most updated version is the 1.2 (dated Jan 14, 2013). This toolbox has been successfully tested with Matlab R2009b, R2010b, R2011a, R2011b, R2012b. Please open the `README` file with your favorite text editor to proceed with the main instructions.

In order to ease the job of reviewers, we suggest testing a run of VeriSiMPL by executing the following commands, which correspond to the numerical example presented in the article.

1. Define the system matrix

```
Ampl = [3 7;2 4]
```

2. Generate the LTS states (`D`) and the dynamical system associated to the MPL model and characterized via `A,B,sysD`

```
[A,B,D] = maxpl2pwa(Ampl)
[A,B,D] = maxpl2pwa_refine(Ampl,A,B,D)
sysD = 1:size(A,2)
```

3. Determine the LTS transitions

```
adj = maxpl2ts_trans(A,B,D,sysD)
```

4. Compute the (transition) LTS labels

```
L = maxpl2ts_label(A,B,D,sysD,adj)
```

The computation of state labels is similar. The obtained TS can be plotted in Graphviz by first generating a text file in dot format (`ts.dot`) by running

```
ts2graphviz(adj,'ts','graph')
```

where `graph` is a name selected for the TS.

The LTS can be exported into PROMELA for verification over transition labels by running the following commands.

1. Define the set of atomic propositions

```
AP = [-Inf 0 0 Inf;-Inf 0 1 6]
```

2. Define the transition from R_1 to R_3 as the initial transition

```
t0 = 1
```

3. Generate a text file (`ts1.pml`) in PROMELA

```
ts2spin_trans(L,adj,'ts1',AP,t0)
```

The export aimed at verification over state labels is similar. Finally, the PROMELA code can be fed to SPIN in order to check whether the LTS model satisfies an LTL formula made up of `AP`, as follows.

1. Generate a C code (`pan.c`) by feeding the negation of the LTL formula to SPIN

```
spin -a -f '![] p1' ts1.pml
```

2. Compile the C code with GCC

```
gcc -o pan pan.c
```

3. Run the executable binary file

```
./pan
```

MPL Models as Timed-Event Graphs

Consider the following two-dimensional autonomous MPL model from [1, p. 4]:

$$x(k) = \begin{bmatrix} 3 & 7 \\ 2 & 4 \end{bmatrix} \otimes x(k-1), \text{ i.e. } \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \begin{bmatrix} \max\{3 + x_1(k-1), 7 + x_2(k-1)\} \\ \max\{2 + x_1(k-1), 4 + x_2(k-1)\} \end{bmatrix}.$$

As discussed in [1, Sec. 2.5], the MPL model can be equivalently expressed as a Timed-Event Graph as follows:

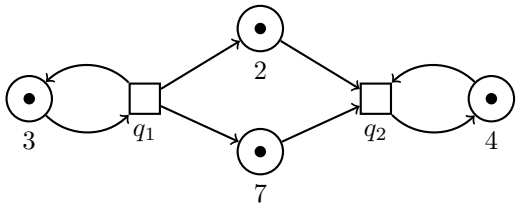

Fig. 2. Timed-Event Graph representation of the autonomous MPL model in in Sec. 2.

Complete Output of the Computational Benchmark

Table 1 reports the mean and maximum values, obtained over 10 independent experiments. Similar outcomes have been generated for nonautonomous MPL models (with non-trivial matrix B and input set U) and for models with full matrix A and have displayed similar trends. The computational bottlenecks in Table 1 conform to a complexity analysis of the abstraction steps in Section 3 and are guiding current upgrades.

Downstream Applications and Outlook

With focus on case studies over the short term, we plan to exploit the capabilities of VeriSiMPL in applications spanning biology and communication systems.

Computationally, VeriSiMPL will be further optimized by employing symbolic structures and by developing a full implementation in C language.

Table 1. Computational benchmark for autonomous MPL model – {mean;max} values

MPL size	generation of states	generation of transitions	generation of labels	number of states	number of transitions
3	{0.1;0.2} [s]	{0.4;0.9} [s]	{0.1;0.1} [s]	{3.6;6.0}	{4.3;13.0}
4	{0.2;0.3} [s]	{0.5;0.9} [s]	{0.1;0.2} [s]	{6.2;12.0}	{11.4;35.0}
5	{0.2;0.3} [s]	{0.4;1.0} [s]	{0.1;0.2} [s]	{8.6;24.0}	{13.8;90.0}
6	{0.4;0.5} [s]	{0.4;0.9} [s]	{0.1;0.2} [s]	{19.4;36.0}	{68.5;191.0}
7	{0.9;1.0} [s]	{0.5;0.9} [s]	{0.3;0.8} [s]	{37.2;84.0}	{289.3;1278.0}
8	{1.5;1.8} [s]	{0.5;0.9} [s]	{0.6;1.7} [s]	{58.0;160.0}	{512.3;1927.0}
9	{4.1;4.8} [s]	{0.8;1.4} [s]	{1.6;3.1} [s]	{120.0;208.0}	$\{1.7;4.3\}\cdot 10^3$
10	{9.5;12.8} [s]	{3.1;15.4} [s]	{7.8;39.3} [s]	{283.6;768.0}	$\{1.3;8.3\}\cdot 10^4$
11	{24.8;32.1} [s]	{15.2;46.6} [s]	{16.1;33.6} [s]	{613.2;1104.0}	$\{1.9;4.8\}\cdot 10^4$
12	{1.2;1.9} [m]	{1.5;3.6} [m]	{42.9;106.1} [s]	$\{1.2;2.0\}\cdot 10^3$	$\{4.7;14.1\}\cdot 10^4$
13	{3.5;5.0} [m]	{5.5;15.5} [m]	{2.8;11.0} [m]	$\{1.9;3.8\}\cdot 10^3$	$\{1.9;8.5\}\cdot 10^5$
14	{12.0;29.6} [m]	{28.2;86.3} [m]	{12.6;54.7} [m]	$\{4.1;8.1\}\cdot 10^3$	$\{7.8;34.5\}\cdot 10^5$
15	{53.6;78.3} [m]	{2.0;9.4} [h]	{39.4;219.6} [m]	$\{7.4;19.7\}\cdot 10^3$	$\{2.0;11.6\}\cdot 10^6$

Additional Bibliography

5. Hillion, H., Proth, J.: Performance evaluation of job-shop systems using timed event-graphs. IEEE Trans. Autom. Control **34**(1) (January 1989) 3–9
6. Roset, B., Nijmeijer, H., van Eekelen, J., Lefeber, E., Rooda, J.: Event driven manufacturing systems as time domain control systems. In: Proc. 44th IEEE Conf. Decision and Control. (December 2005) 446–451
7. Gaubert, S., Katz, R.: Reachability and invariance problems in max-plus algebra. In Benvenuti, L., De Santis, A., Farina, L., eds.: Positive Systems. Volume 294 of Lecture Notes in Control and Information Sciences. Berlin: Springer (2003) 791–793
8. Katz, R.: Max-plus (A,B)-invariant spaces and control of timed discrete-event systems. IEEE Trans. Autom. Control **52**(2) (February 2007) 229–241
9. De Schutter, B.: On the ultimate behavior of the sequence of consecutive powers of a matrix in the max-plus algebra. Linear Algebra and Its Applications **307**(1-3) (2000) 103–117
10. MATLAB: version 7.13.0 (R2011b). The MathWorks Inc., Natick, Massachusetts (2011)
11. Land, A., Doig, A.: An automatic method of solving discrete programming problems. Econometrica **28**(3) (July 1960) 497–520
12. Dill, D.: Timing assumptions and verification of finite-state concurrent systems. In Sifakis, J., ed.: Automatic Verification Methods for Finite State Systems. Volume 407 of Lecture Notes in Computer Science. Berlin: Springer (1990) 197–212
13. Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G.: Graphviz open source graph drawing tools. In Mutzel, P., Jnger, M., Leipert, S., eds.: Graph Drawing. Volume 2265 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 483–484
14. Yordanov, B., Belta, C.: Formal analysis of discrete-time piecewise affine systems. IEEE Trans. Autom. Control **55**(12) (December 2010) 2834–2840
15. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley (2003)