Contemporary Mathematics and Applications
Vol. 5, No. 1, 2023, pp. 15-26.

Desain dan Implementasi Perangkat Lunak Untuk Abstraksi Berhingga Sistem Max-Plus-Linear dengan Tree Tanpa Fungsi Rekursif

Muhammadun[1,*], Dieky Adzkiya[2] & Imam Mukhlash[3]

[1]Universitas Airlangga
[2,3]Institut Teknologi Sepuluh Nopember

*Corresponding author: muhammadun@fst.unair.ac.id

Abstract. Max-Plus-Linear (MPL) systems are a class of discrete event systems with a continuous state space characterizing the underlying sequence of discrete events. In the literature, there are approaches to analysis based on the finite abstraction of autonomous MPL models. This procedure has been implemented in MATLAB with a list/matrix/vector data structure. The drawback of this implementation, the operation makes the transition requires a long computation time. Then made improvements to the previous implementation in JAVA with a tree data structure. This implementation succeeded in speeding up its computation time but requires a larger memory allocation because its functions are recursive. This research discusses the implementation of an autonomous MPL model finite abstraction procedure in C++ using a tree data structure without recursive functions. From several experiments conducted, the implementation in this study succeeded in significantly speeding up VeriSiMPL 2.0 computation time.

Keywords: *sistem Max-Plus-Linear; sistem transisi; model abstraksi; bisimulations; model checking.*

1 Pendahuluan

Sistem Max-Plus-Linear (MPL) [1,2] adalah sistem berbasis peristiwa untuk memodelkan fenomena sinkronisasi tanpa konkurensi. Seperti namanya, sistem MPL terdiri dari dua operasi, yaitu maksimalisasi dan penambahan. Sistem MPL telah digunakan dalam beberapa aplikasi seperti sistem manufaktur [3,4], sistem biologi [5], dan sistem transportasi [2].

Verifikasi adalah proses untuk menentukan apakah suatu model memenuhi spesifikasi [6]. Ada beberapa metode untuk verifikasi seperti pemeriksaan model dan verifikasi deduktif. Pengecekan model adalah teknik verifikasi otomatis untuk sistem state berhingga. Model checker adalah suatu perangkat lunak yang digunakan untuk proses verifikasi. Ada banyak model checker untuk sistem transisi state berhingga, misalnya NuSMV (New Symbolic Model Verifier) [7] dan SPIN (Simple Promela Interpreter) [8]. Jika model memiliki banyak state, maka dapat digunakan model checker secara langsung. Namun, jika model memiliki jumlah state yang tak berhingga, tidak dapat menggunakan

model checker secara langsung. Dalam kasus ini, dapat menerapkan abstraksi berhingga. Abstraksi berhingga adalah sebuah teknik untuk menghasilkan sistem state berhingga dari suatu sistem state yang tak berhingga sedemikian sehingga model abstrak (sistem state berhingga) mensimulasikan model asli (sistem state tak berhingga). Hal ini berarti model abstrak memiliki perilaku yang lebih kompleks daripada perilaku model yang asli. Akibatnya, jika model abstrak memenuhi spesifikasi, dapat disimpulkan bahwa model asli juga memenuhi spesifikasi. Namun, jika model abstrak tidak memenuhi spesifikasi, model asli mungkin masih memenuhi spesifikasi.

Ada banyak literatur yang membahas tentang analisis dan kontrol sistem MPL dari perspektif aljabar dan geometri [1,2]. Baru-baru ini, ada pendekatan alternatif untuk menganalisis perilaku sistem MPL dengan menggunakan abstraksi berhingga [9,10]. Abstraksi berhingga terdiri dari dua langkah. Langkah pertama adalah menentukan state abstrak dan langkah kedua adalah menentukan transisi dari model abstrak. Pendekatan tersebut telah diterapkan di VeriSiMPL (Verification via biSimulations of MPL models) [11]. Implementasi ditulis dalam bahasa pemrograman MATLAB, di mana state abstrak disimpan dalam sebuah list. Pada implementasi ini, langkah kedua yaitu komputasi abstrak transisi membutuhkan waktu komputasi yang besar. Untuk meningkatkan waktu komputasi, VeriSiMPL 2 [12] menggunakan struktur data tree untuk menyimpan state abstrak. Namun, kebutuhan memori meningkat karena menggunakan fungsi rekursif. Sebagai konsekuensinya, metode ini tidak dapat menangani sistem MPL berdimensi besar. Dalam paper ini, diusulkan sebuah algoritma untuk menghitung transisi abstrak tanpa menggunakan fungsi rekursif. Kemudian, algoritmanya diimplementasikan dalam C++ dengan memanfaatkan paradigma pemrograman berorientasi objek yang dibahas dalam [13].

2 Kajian Pustaka

2.1 Sistem *Max-Plus-Linear*

Sistem Max-Plus-Linear (MPL) Autonomous didefinisikan sebagai berikut.

$$\mathbf{x}(k) = A \otimes \mathbf{x}(k-1) \tag{1}$$

$A \in \mathbb{R}_{\varepsilon}^{n\times n}, \mathbf{x}(k-1) = [x_1(k-1), x_2(k-1), \quad \ldots \quad , x_n(k-1)]^T \in \mathbb{R}^n$ untuk $k \in \mathbb{N}$. Ruang state dinotasikan $\mathbb{R}^n$ (daripada $\mathbb{R}_{\varepsilon}^n$), yang juga menyatakan secara tidak langsung bahwa matriks state A adalah matriks baris berhingga. Digunakan huruf bercetak tebal untuk vektor dan tupel, sedangkan nilai entri dinyatakan dengan huruf normal dengan nama dan indeks yang sama. Variabel bebas k menyatakan indeks kejadian, sedangkan variabel kejadian $\mathbf{x}(k)$ menyatakan waktu kejadian ke-k dari semua even. Secara khusus, komponen $x_i(k)$ menyatakan waktu kejadian ke-k dari even ke-i.

2.2 Sistem Transisi

Sistem transisi adalah salah satu model standar yang digunakan dalam verifikasi formal. Sebuah sistem transisi TS [6] diwakili oleh tuple $(S, \longrightarrow, I, AP, L)$,

- S adalah himpunan state
- $\longrightarrow \subseteq S \times S$ adalah relasi transisi
- $I \subseteq S$ adalah himpunan state awal
- AP adalah himpunan atomic proposition. Atomic proposition adalah proposisi yang nilai kebenarannya tidak bergantung pada proposisi lainnya (bersifat atomik).
- $L: S \rightarrow 2^{AP}$ adalah fungsi pelabelan

TS disebut berhingga jika kardinalitas dari S dan AP berhingga.

Untuk kemudahan, ditulis $s \rightarrow s'$ daripada $(s, s') \in \longrightarrow$. Perilaku dari sistem transisi digambarkan sebagai berikut. Sistem transisi berawal dari beberapa state awal $s_0 \in I$ dan berkembang menurut relasi transisi $\longrightarrow$. Jika suatu state memiliki lebih dari satu transisi keluar, transisi berikutnya dipilih dalam cara yang tidak dapat ditentukan. 2^{AP} menyatakan power set dari AP. Fungsi pelabelan memetakan tiap-tiap state ke atomic proposition yang memenuhi pada state.

2.3 *Linear Temporal Logic*

Formula LTL didefinisikan secara berulang atas suatu himpunan atomic proposition dengan operator temporal dan Boolean. Lebih formalnya, sintaks dari formula LTL didefinisikan sebagai berikut:

$$\varphi ::= true | a | \varphi_1 \wedge \varphi_2 | \neg\varphi | \mathrm{O}_\varphi | \varphi_1 \cap \varphi_2$$

untuk $a \in AP$.

Operator Boolean diantaranya ¬ (negasi), ∧ (konjungsi), dan ∨ (disjungsi), sedangkan operator temporal O (next), ∪ (until), □ (always), dan ⋄ (eventually). Operator until memungkinkan untuk memperoleh temporal modalities ⋄ dan □ [6]. Modalitas O adalah unary prefix operator dan memerlukan satu formula LTL sebagai argumennya. Formula O_ϕ memenuhi saat sekarang jika φ memenuhi step berikutnya. Modalitas ∪ adalah sebuah binary infix operator dan memerlukan dua formula LTL sebagai argumen. Formula $\varphi_1 \cup \varphi_2$ memenuhi saat sekarang jika terdapat suatu waktu di masa depan φ_2 terpenuhi dan φ_1 terpenuhi pada semua momen sampai momen masa depan tersebut.

Modalitas $\diamond$ dan modalitas $\square$ adalah operator unary prefix dan memerlukan sebuah formula LTL sebagai argumen: formula $\diamond \varphi$ terpenuhi jika φ akan benar di masa depan, sedangkan formula $\square\varphi$ terpenuhi jika φ terpenuhi dari sekarang hingga seterusnya.

Suatu path dapat memenuhi sebuah formula LTL atau tidak. Suatu infinite path memenuhi suatu formula LTL φ jika trace dari path memenuhi φ [6]. Perhatikan bahwa trace dari infinite path adalah infinite word atas alfabet 2^{AP}. Suatu sistem transisi memenuhi sebuah formula LTL jika semua path dari sistem transisi memenuhi formula LTL.

2.4 Abstraksi Berhingga

Abstraksi adalah konsep dasar yang memungkinkan verifikasi otomatis sistem transisi besar [6, Ex. 7.53] atau bahkan tak berhingga [6, Ex. 7.54]. Abstraksi diidentifikasi oleh beberapa komponen: himpunan state abstrak yang berhingga $\hat{S}$; fungsi abstraksi f yang memetakan setiap state asli s sistem transisi TS ke state abstrak $f(s)$ yang merepresentasikan state asli s; dan suatu himpunan atomic proposition AP yang digunakan untuk melabeli state asli dan abstrak.

Sistem transisi konkret disimulasikan oleh sistem transisi abstrak. Simulasi relasi digunakan sebagai dasar untuk teknik abstraksi, di mana idenya adalah menggantikan model asli untuk diverifikasi oleh model abstrak yang lebih kecil. Simulasi relasi adalah pre order pada ruang keadaan yang mengharuskan jika s'mensimulasikan s, dapat meniru semua perilaku bertahap s, namun kebalikannya tidak dijamin. Definisi formal dari simulasi order disajikan di bawah ini.

[Simulasi Order] [6] Misal $TS_i = (S_i, Act_i \longrightarrow I_i, AP, L_i), i \in \{1,2\}$, menjadi sistem transisi terhadap AP. Sebuah simulasi untuk (TS_1, TS_2) adalah relasi biner $\mathcal{R} \subseteq S1 \times S2$ sehingga

1. untuk setiap $s_1 \in I_1$ terdapat $s_2 \in I_2$ sedemikian hingga $(s_1, s_2) \in \mathcal{R}$
2. untuk semua $(s_1, s_2) \in \mathcal{R}$ berlaku:
    a. $L_1(s_1) = L_2(s2)$
    b. jika $s_1' \in Post(s_1)$ maka terdapat $s_2' \in Post(s_2)$ dengan $(s_1', s_2') \in \mathcal{R}$.

Sebuah sistem transisi TS_1 disimulasikan oleh TS_2 (atau ekuivalen, TS_2 mensimulasikan TS_1) jika terdapat simulasi $\mathcal{R}$ untuk (TS_1, TS_2).

Ide dari abstraksi adalah memetakan himpunan-himpunan state asli ke satu state abstrak. Fungsi abstraksi pada state yang konkrit ke yang abstrak memenuhi: state yang abstrak terkait untuk state konkrit yang berlabel sama.

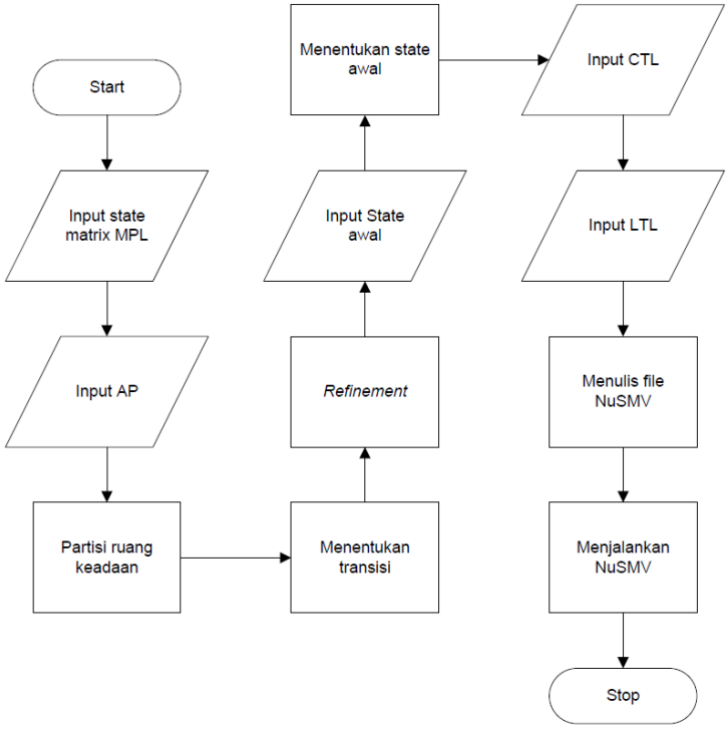
Prosedur untuk mengkontruksi sistem transisi abstrak adalah sebagai berikut. Sistem Transisi abstrak TS_f berasal dari TS dengan mengidentifikasi semua state yang direpresentasikan oleh state abstrak yang sama berdasar fungsi abstraksi f. State abstrak awal merepresentasikan state konkrit awal. Dengan cara yang sama, terdapat transisi dari state abstrak $f(s)$ ke state $f(s')$ jika terdapat transisi dari s ke s'.

Proposisi 2.1 [6] Misalkan $TS = (S, \longrightarrow, I, AP, L)$ sistem transisi yang konkrit, S' himpunan state yang abstrak, dan $f: S \rightarrow S'$ fungsi abstraksi. Maka TS_f mensimulasikan TS.

Proposisi 2.2 [6] Misalkan TS_2 mensimulasikan TS_1, diasumsikan TS_1 tidak memiliki terminal state, dan φ adalah properti linear-time. Jika TS_2 memenuhi φ, maka TS1 juga memenuhi φ.

3 Hasil dan Pembahasan

Diagram alir proses pada perangkat lunak abstraksi berhingga sistem MPL autonomous yang dikembangkan pada penelitian ini secara garis besar bisa dilihat pada Gambar 1.



Gambar 1. Diagram alir proses abstraksi

Pertama user menginputkan state matrix model MPL autonomous. Matriks ini harus merupakan matriks regular (baris berhingga). Kemudian user diminta untuk menginputkan banyaknya himpunan Atomic Proposition (AP). Tiap-tiap AP berkorespondensi dengan sebuah region yang dinyatakan dalam bentuk DBM. Kemudian sistem perangkat lunak akan melakukan partisi ruang keadaan. Partisi yang digunakan adalah partisi pi0. Partisi pi0 melakukan partisi ruang keadaan berbasis tree, yaitu pertama dilakukan partisi AP, kemudian dilanjutkan partisi AD. Setelah diperoleh pi0 partition tree, kemudian dilanjutkan dengan proses Refinement apabila sistem stransisi abstrak yang dihasilkan memiliki satu atau lebih state yang mempunyai transisi keluar lebih dari 1. Proses Refinement ini mempunyai batas atas (upper bound), yaitu suatu stopping condition apabila perulangan proses refinement mencapai suatu kardinalitas tertentu untuk sistem transisi abstrak yang dihasilkan. Kemudian user diminta untuk menginputkan banyaknya state inisial yang berkorespondensi dengan suatu region. Region ini juga direpresentasikan dalam bentuk DBM. Setelah itu, dilakukan proses menentukan state awal untuk sistem transisi abstrak. Maka, sistem transisi abstrak telah terbentuk. Kemudian user diminta untuk menginputkan formula CTL dan LTL, yang kemudian dilakukan penulisan file NuSMV, karena model checker yang digunakan adalah NuSMV. Akhirnya, perangkat lunak akan menjalankan NuSMV untuk mengeksekusi file NuSMV yang telah dibuat. Success return dari kegiatan ini adalah nilai true atau false terhadap verifikasi. Penjelasan dari proses-proses abstraksi berhingga Sistem Max-Plus-Linear Autonomous bisa dilihat pada penelitian sebelumnya oleh Adzikya et al [12].

Untuk menyelesaikan permasalahan abstraksi berhingga ini, akan dibentuk kelas-kelas untuk menyatakan objek-objek yang ada dalam sistem. Hal ini, belum dilakukan pada implementasi sebelumnya yaitu VeriSiMPL 2.0. Secara garis besar, kelas-kelas yang dibentuk digolongkan menjadi 3 bagian, yaitu:

1. Kelas abstraksi

   Kelas ini merupakan kelas utama dalam merepresentasikan proses abstraksi berhingga sistem Max-Plus-Linear autonomous. Yang digolongkan kelas ini adalah kelas Node, Tree, dan AbstractionTree.

2. Kelas Difference-Bound Matrices

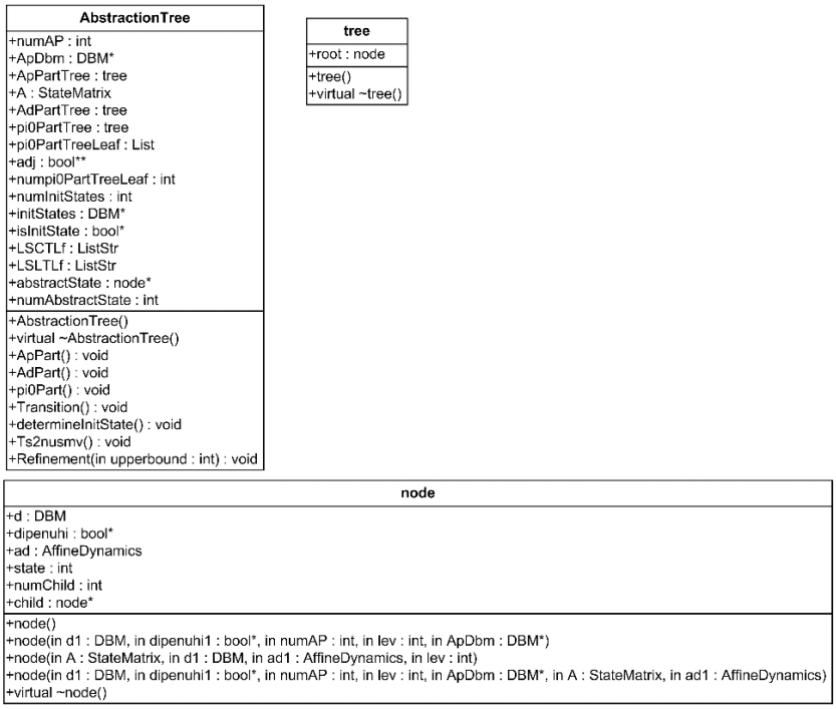
   Yang termasuk di dalam golongan kelas ini adalah kelas DbmInterval, AffineDynamics, dan DBM.

3. Kelas Tambahan

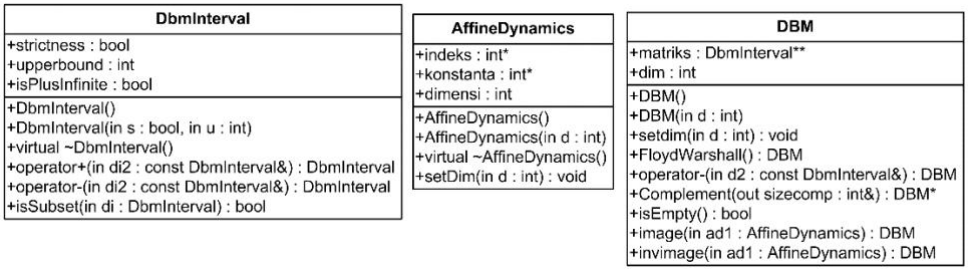
   Kelas-kelas tambahan berupa kelas-kelas yang digunakan untuk melengkapi dua kelas di atas. Kelas-kelas di dalam golongan ini: kelas ListEl, List,

ListStateMatrixElemEl, ListStateMatrixElem, ListStrEl, ListStr, StateMatrixElem, dan StateMatrix.

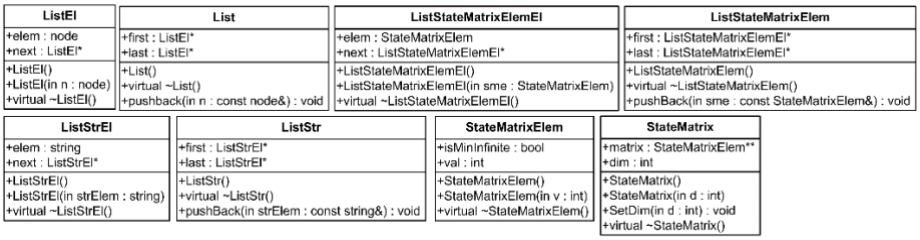
Penjelasan mengenai kelas-kelas di atas dapat dibaca pada paper sebelumnya yang membahas proses desain kelas sistem abstraksi berhingga ini [13]. Desain UML kelas-kelas di atas dapat dilihat pada Gambar 2, Gambar 3, dan Gambar 4.



Gambar 2. Diagram kelas abstraksi



Gambar 3. Digram kelas Difference-Bound Matrices



Gambar 4. Diagram kelas-kelas Tambahan

Implementasi perangkat lunak keseluruhan, yang diberi nama VeriSiMPL 3, bisa diunduh pada laman https://sourceforge.net/projects/verisimpl/postdownload. Berikutnya akan dijelaskan kontribusi utama penelitian, serta uji coba yang dilakukan yang dibandingkan dengan penelitian sebelumnya.

3.1 *Improved Algorithms*

Proses penentuan transisi adalah sebagai berikut. Pertama, untuk setiap node leaf, dihitung image dari DBM yang bersesuaian berdasar affine dynamics nya. Kemudian dicari node-node leaf yang beririsan dengan image menggunakan backtracking, tanpa menggunakan fungsi rekursif.

Algoritma backtracking untuk menentukan node leaf yang beririsan dengan suatu image adalah sebagai berikut:

1. Inisialisasi $\mathcal{L}$ sebagai himpunan kosong. Himpunan $\mathcal{L}$ merepresentasikan himpunan node leaf sedemikian sehingga DBM yang bersesuaian beririsan dengan image.

2. Definisikan node root sebagai current node.

3. Tentukan irisan antara image dan DBM yang bersesuaian dengan current node:

    a. Jika irisannya tidak kosong dan current node memiliki setidaknya satu anak, definisikan anak pertama sebagai current node dan lanjutkan ke langkah 3.

    b. Jika irisannya tidak kosong dan current node adalah node leaf, tambahkan node ini ke $\mathcal{L}$. Jika current node memiliki saudara kanan, definisikan saudara kanan sebagai current node dan lanjutkan ke langkah 3. Jika current node adalah anak yang terakhir, cari ancestor terdekat yang memiliki saudara kanan. Jika ancestor tersebut ada, definisikan saudara

kanan dari ancestor sebagai current node dan lanjutkan ke langkah 3. Jika semua ancestor adalah anak terakhir, proses selesai.

c. Jika irisannya kosong dan current node bukan anak terakhir, definisikan anak berikutnya sebagai current node dan lanjutkan ke langkah 3.

d. Jika irisannya kosong dan current node adalah anak terakhir, cari ancestor terdekat yang memiliki saudara kanan. Jika ancestor tersebut ada, definisikan saudara kanan dari ancestor sebagai current node dan lanjutkan ke langkah 3. Jika semua ancestor adalah anak terakhir, proses selesai.

3.2 Uji Coba

Pada bagian ini akan dijelaskan hasil perbandingan dengan implementasi yang telah dilakukan sebelumnya, yaitu VeriSiMPL 2.0. Uji coba ini dilakukan dengan menggunakan komputer yang berspesifikasi sebagai berikut:

1. Operating System: Windows 7 Home Premium 64-bit (6.1, Build 7601) Service Pack 1 (7601.win7sp1_gdr.140303-2144)
2. Processor: Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz (4 CPUs), ~2.5GHz
3. Memory: 4096MB RAM

Uji coba perbandingan perangkat lunak dilakukan dengan memberikan input yang sama pada kedua perangkat lunak, kemudian dihitung *running time* dari dua implementasi tersebut. Desain input yang diberikan pada uji perangkat lunak adalah sebagai berikut:

A. *State matrix*
   Matriks State (keadaan) dari Sistem Max-Plus-Linier pada masing-masing percobaan yang dilakukan adalah sebagai berikut:
   1. 2,5;3,3;
   2. 1,2,3;4,5,6;7,8,9;
   3. 1,2,3,4;5,6,7,8;9,10,11,12;13,14,15,16;
   4. 1,2,3,4,5;6,7,8,9,10;11,12,13,14,15;16,17,18,19,20;21,22,23,24,25;
   5. 1,2,3,4,5,6;7,8,9,10,11,12;13,14,15,16,17,18;19,20,21,22,23,24;25,26,27,28, 29,30;31,32,33,34,35,36;

B. *Atomic proposition* pada setiap percobaan adalah sama, yaitu sebanyak satu yang berkorespondensi dengan DBM $0 \leq x_1 - x_2 < 3$.
   Jadi, matriks yang diinputkan pada perangkat lunak untuk masing-masing percobaan adalah sebagai berikut:

Percobaan	AP		
	Number	Upper bound matrix	Strictness matrix
1	1	0,inf,inf;inf,0,0;inf,3,0;	1,0,0;0,1,1;0,0,1;
2	1	0,inf,inf,inf;inf,0,0,inf;inf,3,0,inf;inf,inf,inf,0;	1,0,0,0;0,1,1,0;0,0,1,0;0,0,0,1;
3	1	0,inf,inf,inf,inf;inf,0,0,inf,inf;inf,3,0,inf,inf;inf,inf,inf,0,inf;inf,inf,inf,inf,0;	1,0,0,0,0;0,1,1,0,0;0,0,1,0,0;0,0,0,1,0;0,0,0,0,1;
4	1	0,inf,inf,inf,inf,inf;inf,0,0,inf,inf,inf;inf,3,0,inf,inf,inf;inf,inf,inf,0,inf,inf;inf,inf,inf,inf,0,inf;inf,inf,inf,inf,inf,0;	1,0,0,0,0,0;0,1,1,0,0,0;0,0,1,0,0,0;0,0,0,1,0,0;0,0,0,0,0,1,0;0,0,0,0,0,1;

C. DBM yang merepresentasikan *state* awal pada setiap percobaan adalah sama yaitu sebanyak 1 yaitu $x_1 - x_2 = 1$.
Berikut adalah input untuk masing-masing percobaan.

Percobaan	Initial State		
	Number DBM	Upper bound difference matrix	Upper bound strictness matrix
1	1	0,inf,inf;inf,0,-1;inf,1,0;	1,0,0;0,1,1;0,1,1;
2	1	0,inf,inf,inf;inf,0,-1,inf;inf,1,0,inf;inf,inf,inf,0;	1,0,0,0;0,1,1,0;0,1,1,0;0,0,0,1;
3	1	0,inf,inf,inf,inf;inf,0,-1,inf,inf;inf,1,0,inf,inf;inf,inf,inf,0,inf;inf,inf,inf,inf,0;	1,0,0,0,0;0,1,1,0,0;0,1,1,0,0;0,0,0,1,0;0,0,0,0,1;
4	1	0,inf,inf,inf,inf,inf;inf,0,-1,inf,inf,inf;inf,1,0,inf,inf,inf;inf,inf,inf,0,inf,inf;inf,inf,inf,inf,inf,0;	1,0,0,0,0,0;0,1,1,0,0,0;0,1,1,0,0,0;0,0,0,1,0,0;0,0,0,0,1,0;0,0,0,0,0,1;

D. Formula CTL
Untuk semua percobaan, CTL Formulae yang diinputkan adalah sama, yaitu yang diinputkan pada perangkat lunak adalah sebagai berikut:
AX(AX(a))
EG(AF(!a))

E. Formula LTL
Untuk semua percobaan, LTL Formulae yang diinputkan adalah sama, yaitu yang diinputkan pada perangkat lunak adalah sebagai berikut:

G(a)
F(a)

Output dari setiap uji coba berupa sistem transisi hasil abstraksi, waktu proses uji coba, serta hasil verifikasi oleh NuSMV. Dari uji coba yang telah dilakukan, kedua perangkat lunak menghasilkan sistem transisi abstrak dan nilai verifikasi yang sama. Beberapa attribut dari sistem transisi abstrak dan nilai verifikasi dari uji coba yang telah dilakukan dapat diliihat pada Tabel 1.

Tabel 1 Beberapa atribut sistem transisi abstrak dan nilai verifikasi hasil uji coba.

Uji Coba ke-	**Ukuran Sistem Transisi**	**State Awal**		**Nilai Verifikasi Formula**	
		VeriSiMPL2	**Paper**	**CTL**	**LTL**
1	7	s5	s1	true, false	true, true
2	7	s3, s5	s2, s3	false, true	false, true
3	10	s3, s5, s8	s2, s3, s4	false, true	false, true
4	13	s3, s5, s8, s11	s2, s3, s4, s5	false, true	false, true
5	16	s3, s5, s8, s11, s14	s2, s3, s4, s5, s6	false, true	false, true

Perbedaan output state inisial sistem transisi abstrak pada dua perangkat lunak hanya merupakan perbedaan cara melabeli state-statenya saja. Untuk running time uji coba dan rasio (running time VeriSiMPL 2.0: running time program paper) bisa dilihat pada Tabel 2.

Tabel 2 *Running time* uji coba.

Percobaan ke-	***Running time* (detik)**		**Rasio**
	VeriSiMPL 2.0	**Paper**	
1	0.048257009	0.008	6.032126125
2	0.057136887	0.008	7.142110875
3	0.144904857	0.012	12.07540475
4	5.122412259	0.02	256.12061295
5	1477.64639893	0.037	39936.38916027027

4 Kesimpulan

Dari beberapa hasil uji coba yang telah dilakukan, perangkat lunak yang dihasilkan pada paper ini memberikan *output* yang sama dengan *output* pada VeriSiMPL 2.0. Maka dapat dianggap bahwa perangkat lunak telah dirancang dengan benar. Kemudian dilihat dari *running time* uji coba, implementasi pada penelitian ini berhasil mempercepat waktu komputasi VeriSiMPL 2.0 secara signifikan.

5 Referensi

[1] Baccelli, F., Cohen, G., Olsder, G. & Quadrat, J.-P., 1992, *Synchronization and Linearity, An Algebra for Discrete Event Systems*, John Wiley and Sons.

[2] Heidergott, B., Olsder, G. & van der Woude, J., 2006, *Max Plus at Work–Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications*, Princeton University Press.

[3] Roset, B., Nijmeijer, H., van Eekelen, J., Lefeber, E. & Rooda, J., 2005, *Event driven manufacturing systems as time domain control systems*, Conference on Decision and Control, pp. 446-451.

[4] van Eekelen, J., Lefeber, E. & Rooda, J., 2006, *Coupling event domain and time domain models of manufacturing systems*, Conference on Decision and Control, pp. 6068-6073.

[5] Brackley, C. A., Broomhead, D. S., Romano, M. C. & Thiel, M., 2012, *A max-plus model of ribosome dynamics during mRNA translation*, Journal of Theoretical Biology, 128-140.

[6] Baier, C. & Katoen, J.-P., 2008, *Principles of Model Checking*, The MIT Press.

[7] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. & Tacchella, A., 2002, *Nusmv 2: An opensource tool for symbolic model checking*, International Conference on Computer Aided Verification, Springer, pp. 359-364.

[8] Holzmann, G. J., 1997, *The model checker spin*, IEEE Transactions on software engineering, 279-295.

[9] Adzkiya, D., De Schutter, B. & Abate, A., 2013, *Finite abstractions of max-plus-linear systems*, IEEE Transactions on Automatic Control, 3039-3053.

[10] Adzkiya, D., De Schutter, B. & Abate, A., 2015, *Computational techniques for reachability analysis of max-plus-linear systems*, Automatica, 293-302.

[11] Adzkiya, D. & Abate, A., 2013, *VeriSiMPL: Verification via biSimulations of MPL models*, Quantitative Evaluation of Systems, 253-256.

[12] Adzkiya, D., Zhang, Y. & Abate, A., 2016, *VeriSiMPL 2: An open-source software for the verification of max-plus-linear systems*, Discrete Event Dynamic Systems, 109-145.

[13] Muhammadun, M., Adzkiya, D. & Mukhlash, I., 2017, *Object Oriented Design of Software Tool for Finite Abstractions of Max-Plus-Linear Systems using Unified Modeling Language*, International Journal of Computing Science and Applied Mathematics, 32-39.