TESIS-SM 142501

DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK UNTUK ABSTRAKSI BERHINGGA SISTEM *MAX-PLUS-LINEAR* DENGAN *TREE* TANPA FUNGSI REKURSIF

Muhammadun
1214 201 008

DOSEN PEMBIMBING
Dr. Imam Mukhlash, M.T.
Dr. Dieky Adzkiya, S.Si., M.Si.

PROGRAM MAGISTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

THESIS-SM 142501

DESIGN AND IMPLEMENTATION OF SOFTWARE FOR FINITE ABSTRACTIONS OF MAX-PLUS-LINEAR SYSTEMS USING TREE WITHOUT RECURSIVE FUNCTIONS

Muhammadun
1214 201 008

SUPERVISORS
Dr. Imam Mukhlash, M.T.
Dr. Dieky Adzkiya, S.Si., M.Si.

MASTER'S DEGREE
MATHEMATICS DEPARTMENT
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2017

DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK UNTUK ABSTRAKSI BERHINGGA SISTEM MAX-PLUS-LINEAR DENGAN TREE TANPA FUNGSI REKURSIF

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Sains (M.Si.)
di
Institut Teknologi Sepuluh Nopember

Oleh:
MUHAMMADUN
NRP. 1214 201 008

Tanggal Ujian : 11 Januari 2017
Periode Wisuda : Maret 2017

Disetujui oleh:

Dr. Imam Mukhlash, M.T.
NIP. 19700831 199403 1 003 (Pembimbing)

Dr. Dieky Adzkiya, S.Si., M.Si.
NIP. 19830517 200812 1 003 (Pembimbing)

Dr. Subiono, M.S.
NIP. 19570411 198403 1 001 (Penguji)

Dr. Chairul Imron, MI.Komp.
NIP. 19611115 198703 1 003 (Penguji)

a.n. Direktur Program Pascasarjana,
Asisten Direktur

Prof. Dr. Ir. Tri Widjaja, M.Eng.
NIP. 19611021 198603 1 001

DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK UNTUK ABSTRAKSI BERHINGGA SISTEM *MAX-PLUS-LINEAR* DENGAN *TREE* TANPA FUNGSI REKURSIF

Nama mahasiswa : Muhammadun
NRP : 1214201008
Pembimbing : 1. Dr. Imam Mukhlash, M.T.
2. Dr. Dieky Adzkiya, S.Si., M.Si.

ABSTRAK

Sistem *Max-Plus-Linear* (MPL) adalah suatu kelas sistem *event* diskrit dengan ruang keadaan kontinu mengkarakterisasi sekuensial kejadian diskrit yang mendasari. Di literatur, ada pendekatan untuk analisis yang didasarkan pada abstraksi berhingga model MPL yang *autonomous*. Prosedur ini telah diiimplementasikan dalam MATLAB dengan struktur data *list*/matrik/vektor. Kekurangan dari implementasi ini, operasi membuat transisinya membutuhkan waktu komputasi yang lama. Kemudian dilakukan perbaikan terhadap implementasi sebelumnya dalam JAVA dengan struktur data *tree*. Implementasi ini berhasil mempercepat waktu komputasinya tetapi membutuhkan alokasi memori yang lebih besar karena fungsi-fungsinya bersifat rekursif. Tesis ini membahas implementasi prosedur abstraksi berhingga model MPL *autonomous* dalam C++ dengan menggunakan struktur data *tree* tanpa fungsi rekursif, dengan harapan dapat memperbaiki hasil yang diperoleh dari dua implementasi sebelumnya.

Kata kunci: sistem *Max-Plus-Linear*, sistem transisi, model abstraksi, *bisimulations*, model *checking*

DESIGN AND IMPLEMENTATION OF SOFTWARE FOR FINITE ABSTRACTIONS OF MAX-PLUS-LINEAR SYSTEMS USING TREE WITHOUT RECURSIVE FUNCTIONS

By	: Muhammadun
NRP	: 1214201008
Supervisor	: 1. Dr. Imam Mukhlash, M.T.
	2. Dr. Dieky Adzkiya, S.Si., M.Si.

ABSTRACT

Max-Plus-Linear (MPL) systems are a class of discrete-event systems with a continuous state space characterizing the timing of the underlying sequential discrete events. In the literature, there is an approach to analyze these systems based on finite abstractions. This algorithms have been implemented in MATLAB using list/matrix/vector data structure. Disadvantages of this implementation, operation makes the transition requires a long computation time. Then improvement against previous implementation in JAVA using tree data structure. This implementation successfully accelerate computation time but requires a larger memory allocation because its functions are recursive. This thesis discuss implementation in C++ using tree data structure without recursive functions in the hope of improving the results obtained by the two previous implementations.

Key words: Max-Plus-Linear systems, transition systems, model abstractions, bisimulations, model checking

KATA PENGANTAR

Maha Suci Allah, dzat yang Maha Kuasa atas segala sesuatu. Sebesar apapun suatu masalah adalah kecil dihadapan-Nya. Segala puji hanyalah bagi Allah yang atas pertolongan-Nya sehingga penulis dapat menyelesaikan laporan Tesis yang berjudul:

" DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK UNTUK ABSTRAKSI BERHINGGA SISTEM *MAX-PLUS-LINEAR* DENGAN TREE TANPA FUNGSI REKURSIF".

Atas bantuan dan dukungan dari berbagai pihak, maka berbagai kendala dan hambatan selama mengerjakan tesis menjadi terasa indah dan menyenangkan untuk dilalui. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada berbagai pihak yang telah membantu terwujudnya laporan tesis ini, antara lain kepada:

1. Ibu dan ayahku. Semoga Allah SWT memuliakan mereka.
2. Bapak Dr. Dieky Adzkiya, S.Si., M.Si. yang telah banyak membantu penulis dalam penyusunan tesis ini, baik dalam hal teknis maupun non teknis.
3. Kakak, mbak, adik-adik, serta keluargaku yang telah membuat hidup penulis menjadi lebih hidup dan mengasyikkan.
4. Bapak Dr. Imam Mukhlash, M.T., karena telah menjadi dosen pembimbing yang menyenangkan dan banyak sekali memberikan bantuan serta bimbingan kepada penulis selama mengerjakan tesis.
5. Guru-guruku yang selalu membimbing dan mendukungku setiap saat.
6. Semua dosen-dosenku di S2 Matematika ITS. Khususnya kepada Bapak Dr. Chairul Imron, MI.Komp. dan Bapak Dr. Subiono, M.S., karena telah menjadi dosen penguji yang menyenangkan.
7. Teman-temanku di S2 Matematika ITS angkatan 2014 yang telah menemani penulis dalam bermain dan belajar di kampus.

8. Semua pihak yang tidak dapat penulis sebutkan satu per satu; semoga Allah SWT menganugerahkan rasa ikhlas dan rendah hati kepada semua pihak yang telah membantu penulis, kepada semua pihak yang tidak membantu penulis, maupun kepada penulis sendiri.

Akhirnya, penulis menyadari bahwa tesis ini masih banyak terdapat kekurangan yang semuanya disebabkan oleh kelemahan dan keterbatasan penulis. Saran dan kritik demi perbaikan di masa datang akan sangat penulis hargai. Semoga tesis ini dapat bermanfaat bagi kita semua. Aamiin.

Surabaya, Januari 2017

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Sistem *Max-Plus-Linear* (MPL) adalah suatu kelas sistem *event* diskrit dengan ruang keadaan kontinu yang mengkarakterisasi kejadian diskrit sekuensial (Adzkiya, 2013). Model MPL digunakan untuk menggambarkan sinkronisasi waktu antara proses *interleaved*, dengan asumsi bahwa waktu kejadian sekarang bergantung secara *linear* (dalam aljabar *max-plus*) pada kejadian *event* sebelumnya. Model MPL telah banyak digunakan dalam analisis dan penjadwalan jaringan infrastruktur, seperti sistem komunikasi, kereta api, produksi dan manufaktur atau sistem biologi (Brackley, 2012). Sistem ini tidak bisa memodelkan *concurrency* dan terkait dengan *subclass* dari *Timed Petri Nets*, yaitu *Timed Event Graph*.

Analisis klasik dari model MPL didasarkan pada aljabar atau geometri. Hal ini memungkinkan untuk menyelidiki sifat Model seperti perilaku *transien*, rezim periodik, atau perilaku dinamik lainnya. Adzkiya (2013) telah mengeksplorasi pendekatan alternatif untuk analisis yang didasarkan pada abstraksi berhingga model MPL *autonomous* dan *nonautonomous*.

Prosedur abstraksi yang diajukan menghasilkan Sistem Transisi (TS) dengan jumlah *state* berhingga. *State* berhingga dari TS diperoleh dengan mempartisi ruang keadaan dari model MPL, sedangkan hubungan antara pasangan *state* di TS ditetapkan dengan memeriksa apakah ada lintasan dari model MPL yang diperbolehkan untuk melakukan transisi antara daerah partisi yang bersesuaian. TS dengan *state* berhingga mungkin berisi transisi yang layak untuk model MPL dan dengan demikian dapat mendekati dinamika model MPL. Secara komputasi, karakterisasi ini dilakukan dengan analisis *forward-reachability* untuk *Piece-Wise Affine* (PWA) yang merepresentasikan dinamika MPL. Model PWA ditandai dengan *vector fields* yang *affine* (*linear*, ditambah *offset*), dalam *polytopes* yang konveks atas ruang *state* dan input. Dalam rangka membangun

hubungan formal antara model dan abstraksinya, Adzkiya et al (2013) berpendapat bahwa secara umum abstraksi LTS mensimulasikan model MPL aslinya, dan selanjutnya mereka menyediakan syarat cukup untuk membangun hubungan *bisimulation* antara model abstrak dan konkret. Pendekatan tersebut memanfaatkan representasi daerah spasial dan dinamika berdasarkan *Difference Bound Matrices* (DBM). Representasi ini memungkinkan untuk komputasi yang cepat (Adzkiya, 2013).

Adzkiya (2013) telah mengimplementasikan prosedur ini dalam MATLAB dengan struktur data *list*/matrik/vektor. Kekurangan dari implementasi ini, operasi membuat transisinya membutuhkan waktu komputasi yang lama. Kemudian Adzkiya et al (2015) telah menyempurnakan implementasi sebelumnya dalam JAVA dengan struktur data tree. Implementasi ini berhasil mempercepat waktu komputasinya tetapi membutuhkan alokasi memori yang lebih besar karena fungsi-fungsinya bersifat rekursif. Tesis ini akan membahas implementasi dalam C++ dengan menggunakan struktur data tree tanpa fungsi rekursif dengan harapan dapat memperbaiki hasil yang diperoleh pada dua implementasi sebelumnya.

1.2 Rumusan Masalah

Permasalahan yang akan diangkat dalam penelitian ini adalah bagaimana mengimplementasikan permasalahan abstraksi berhingga dari sistem *Max-Plus-Linear* dalam C++ dengan menggunakan struktur data tree tanpa fungsi rekursif sedemikian hingga perangkat lunak yang dihasilkan lebih efisien.

1.3 Tujuan dan Manfaat Penelitian

Tujuan dan manfaat penelitian ini adalah membuat aplikasi untuk abstraksi berhingga dari sistem *Max-Plus-Linier* yang efisien. Abstraksi yang dihasilkan oleh aplikasi tersebut dapat digunakan untuk memverifikasi sistem *Max-Plus-Linear* secara otomatis dengan *model checker*.

BAB 2
KAJIAN PUSTAKA

2.1 Sistem *Max-Plus-Linear*

Berikut adalah penjelasan tentang Sistem *Max-Plus-Linear* sebagaimana yang diulas di (Adzkiya, 2015). Didefinisikan $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_\varepsilon$, dan ε berturut-turut: himpunan bilangan asli $\{1, 2, 3, \dots\}$, himpunan bilangan real, $\mathbb{R} \cup \{\varepsilon\}$, dan $-\infty$. Untuk $\alpha, \beta \in \mathbb{R}_\varepsilon$, diperkenalkan dua operasi

$$\alpha \oplus \beta = \max\{\alpha, \beta\} \quad \text{dan} \quad \alpha \otimes \beta = \alpha + \beta$$

Diberikan $\beta \in \mathbb{R}$, *max-algebraic power* dari $\alpha \in \mathbb{R}$ dinyatakan dengan $\alpha^{\otimes\beta}$ yang bersesuaian dengan $\alpha \times \beta$ pada aljabar konvensional. Aturan untuk urutan evaluasi operator *max-algebraic* sama seperti aljabar konvensional: *max-algebraic power* memiliki prioritas tertinggi, dan perkalian *max-algebraic* memiliki presedensi yang lebih tinggi daripada penjumlahan *max-algebraic*. Operasi-operasi dasar *max-algebraic* diperluas ke matriks seperti berikut. Misal matriks $A, B \in \mathbb{R}_\varepsilon^{m\times n}$, $C \in \mathbb{R}_\varepsilon^{m\times p}$, $D \in \mathbb{R}_\varepsilon^{p\times n}$, dan skalar $\alpha \in \mathbb{R}_\varepsilon$; kita peroleh

$$[\alpha \otimes A](i,j) = \alpha \otimes A(i,j) = \alpha + A(i,j),$$

$$[A \oplus B](i,j) = A(i,j) \oplus B(i,j) = \max\{A(i,j), B(i,j)\},$$

$$[C \otimes D](i,j) = \bigoplus_{k=1}^{p} C(i,k) \otimes D(k,j) = \max_{k\in\{1,\cdots,p\}} \{C(i,k) + D(k,j)\},$$

untuk semua $i \in \{1, \cdots, m\}$ dan $j \in \{1, \cdots, n\}$. Notasi $A(i,j)$ menyatakan nilai skalar dari elemen matriks A baris ke-i kolom ke-j. Perhatikan persamaan antara $\oplus$, $\otimes$ dan $+$, $\times$ untuk operasi matriks dan vektor dalam aljabar konvensional. Diberikan $m \in \mathbb{N}$, *max-algebraic power* ke-m dari matriks A dinotasikan dengan $A^{\otimes m}$ dan berkorespondensi dengan $A \otimes \cdots \otimes A$ (m kali). Perhatikan bahwa $A^{\otimes 0}$ adalah suatu matriks identitas *max-plus* dimensi *n*, yaitu elemen diagonalnya 0 dan elemen bukan diagonalnya ε. Notasi berikut diambil untuk alasan kemudahan: suatu vektor dengan semua komponen sama dengan 0 juga dinotasikan dengan 0.

Demikian juga, vektor dengan semua komponen sama dengan $-\infty$ dinotasikan dengan ε.

Suatu sistem *Max-Plus-Linear* (MPL) *Autonomous* didefinisikan sebagai

$$\mathbf{x}(k) = A \otimes \mathbf{x}(k-1) \tag{2.1}$$

dimana $A \in \mathbb{R}_{\varepsilon}^{n\times n}$, $x(k-1) = [x_1(k-1) \cdots x_n(k-1)]^T \in \mathbb{R}^n$ untuk $k \in \mathbb{N}$. Lebih lanjut, ruang keadaan dinotasikan $\mathbb{R}^n$ (daripada $\mathbb{R}_{\varepsilon}^n$), yang juga menyatakan secara tidak langsung bahwa matriks keadaan A adalah baris berhingga. Kita menggunakan 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 keadaan $x_i(k)$ menyatakan waktu kejadian ke-k dari even ke-i.

Definisi 2.1 (Matriks Regular atau Baris Berhingga (Heidergott, 2006)) Suatu matriks $A \in \mathbb{R}_{\varepsilon}^{n\times n}$ disebut regular (baris berhingga) jika A mengandung satu elemen bukan ε pada tiap barisnya.

Contoh 1 Perhatikan sistem MPL dimensi dua berikut yang memodelkan suatu jaringan kereta api sederhana antara dua kota. Di sini $x_i(k)$ menyatakan waktu keberangkatan ke-k pada stasiun i untuk $i \in \{1,2\}$, dan diperbaharui sebagai

$$\mathbf{x}(k) = \begin{bmatrix} 2 & 5 \\ 4 & 3 \end{bmatrix} \otimes \mathbf{x}(k-1), \text{ atau secara ekuivalen,}$$

$$\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \begin{bmatrix} \max\{2 + x_1(k-1), 5 + x_2(k-1)\} \\ \max\{4 + x_1(k-1), 3 + x_2(k-1)\} \end{bmatrix}. \tag{2.2}$$

Perhatikan bahwa dalam contoh ini A adalah suatu matriks baris berhingga.

2.2 Sistem *Piecewise Affine*

Berikut adalah penjelasan tentang Sistem *Piecewise Affine* sebagaimana yang diulas di (Leenaerts, 1998) dan (Adzkiya, 2015). Sistem dinamik *Piece-wise*

Affine (PWA) dikarakterisasi oleh suatu partisi dari ruang keadaan dengan *affine* (linier, plus konstan) dinamik yang aktif pada tiap himpunan cover. Sistem PWA adalah *well-posed* jika *state* berikutnya dari setiap *state* adalah tunggal. Sistem PWA cukup ekspresif untuk memodelkan sejumlah besar proses fisik, dan dapat mengaproksimasi dinamika *nonlinier* dengan akurasi yang berubah-ubah via *local linearizations* pada *operating point* yang berbeda. Sistem PWA telah dipelajari oleh beberapa peneliti.

Bagian ini membahas sistem PWA yang diperoleh dari sistem MPL. Sistem PWA yang diperoleh adalah *well-posed* karena sistem MPL juga *well-posed*. Konstruksi sistem PWA dari sistem MPL memiliki kompleksitas yang kombinanatorik: untuk meningkatkan *performance* digunakan pendekatan *backtracking*.

Suatu sistem MPL dikarakterisasi oleh matriks keadaan $A \in \mathbb{R}_{\varepsilon}^{n\times n}$ dapat diekspresikan sebagai suatu sistem PWA dalam domain kejadian. *Affine dynamics*, bersama dengan daerah yang bersesuaian pada ruang keadaan, dapat dikonstruksi dari koefisien $\mathbf{g} = (g_1, \dots, g_n) \in \{1, \dots, n\}^n$. Untuk setiap i, koefisien g_i mengkarakterisasi *maximal term* pada persamaan keadaan ke- i $x_i(k) = \max\{A(i,1) + x_1, \dots, A(i,n) + x_n\}$, yaitu $A(i,j) + x_j \le A(i,g_i) + x_{g_i}$, untuk semua $j \in \{1, \dots, n\}$. Mengikuti hal ini, himpunan keadaan yang bersesuaian dengan **g**, dinyatakan dengan $R_{\mathbf{g}}$, adalah

$$R_{\mathbf{g}} = \bigcap_{i=1}^{n} \bigcap_{j=1}^{n} \{\mathbf{x} \in \mathbb{R}^n : A(i,j) + x_j \le A(i,g_i) + x_{g_i}\}. \tag{2.3}$$

Suatu titik atau keadaan $\mathbf{x} \in \mathbb{R}^n$ berada pada $\mathbb{R}_{\mathbf{g}}$ jika $\max_{j\in\{1,\dots,n\}}\{A(i,j) + x_j\} = A(i,g_i) + x_{g_i}$, untuk semua $i \in \{1, \dots, n\}$.

Affine dynamics yang aktif pada $R_{\mathbf{g}}$ mengikuti secara langsung dari definisi **g** (lihat paragraf sebelumnya) sebagai

$$x_i(k) = x_{g_i}(k-1) + A(i,g_i), \qquad i \in \{1, \dots, n\}. \tag{2.4}$$

Diberikan matriks keadaan baris berhingga A, algoritma 2.1 mendeskripsikan prosedur umum untuk menkontruksi sistem PWA yang berhubungan dengan suatu sistem MPL.

Algoritma 2.1 (Pembentukan sistem PWA dari suatu sistem MPL)

Input: $A \in \mathbb{R}_{\varepsilon}^{n\times n}$, suatu matriks keadaan baris berhingga

Output: **R**, **A**, **B**, suatu sistem PWA pada $\mathbb{R}^n$,

dimana **R** adalah himpunan daerah-daerah dan **A**, **B** menyatakan suatu himpunan *affine dynamics*

inisialisasi **R**, **A**, **B** dengan himpunan kosong

`for all` $\mathbf{g} \in \{1, \dots, n\}^n$ `do`

buat daerah $R_{\mathbf{g}}$ berdasarkan (2.3)

`if` $R_{\mathbf{g}}$ tidak kosong `then`

buat matriks $A_{\mathbf{g}}$, $B_{\mathbf{g}}$ dimana $\mathbf{x}_k = A_{\mathbf{g}}x(k-1) + B_{\mathbf{g}}$, berdasarkan (2.4)

simpan hasil, $\mathbf{R} \coloneqq \mathbf{R} \cup \{R_{\mathbf{g}}\}$, $\mathbf{A} \coloneqq \mathbf{A} \cup \{A_g\}$, $\mathbf{B} \coloneqq \mathbf{B} \cup \{B_g\}$

`end if`

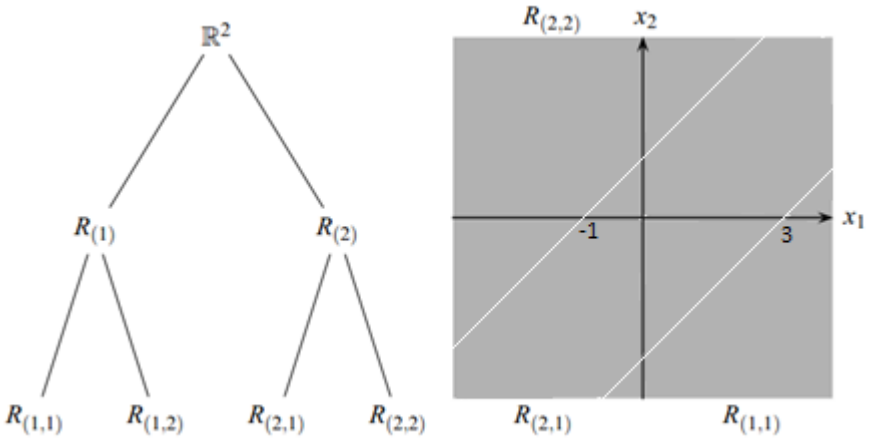
`end for`

Pengamatan penting yang membawa kepada perbaikan kompleksitas yaitu tidak perlu untuk melakukan iterasi atas semua kemungkinan koefisien pada **g**, seperti yang diusulkan pada algoritma 2.1. Kita dapat melakukan teknik *backtracking*: dengan pendekatan ini, kita memasukkan koefisien parsial $(g_1, \dots, g_n)$ untuk $k \in \{1, \dots, n\}$ dan daerah yang bersesuaian

$$R_{(g_1,\dots,g_k)} = \bigcap_{i=1}^{k} \bigcap_{j=1}^{n} \{\mathbf{x} \in \mathbb{R}^n : A(i, g_i) + x_{g_i} \geq A(i, j) + x_j\}. \tag{2.5}$$

Perhatikan bahwa jika daerah yang berhubungan dengan beberapa koefisien parsial $(g_1, \dots, g_k)$ kosong, maka daerah yang berkorespondensi dengan koefisien

$(g_1, \ldots, g_n)$ juga kosong, untuk semua $g_{k+1}, \ldots, g_n$. Himpunan semua koefisien dapat dinyatakan sebagai *potential search tree*.



Gambar 2.1. (kiri) *potential search tree* untuk sebuah sistem MPL dimensi 2. (kanan) daerah yang berhubungan dengan sistem PWA yang dibentuk oleh sistem MPL dalam (2.1).

Untuk sebuah sistem MPL dimensi 2, *potential search tree* diberikan pada Gambar 2.1 (kiri). Algoritma *backtracking* melintasi *tree* secara berulang, mulai dari *root*, dalam *depth-first order*. Pada tiap-tiap *node*, algoritma memeriksa apakah daerah yang berkorespondensi kosong. Jika daerah kosong, komputasi terhadap keseluruhan *sub-tree* dengan *root node* tersebut dilewati.

Contoh 2 Dengan referensi kepada sistem MPL dalam (2), diperoleh sistem PWA

$$\mathbf{x}(k) = \begin{cases} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \mathbf{x}(k-1) + \begin{bmatrix} 2 \\ 4 \end{bmatrix}, & \text{jika } \mathbf{x}(k-1) \in R_{(1,1)}, \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathbf{x}(k-1) + \begin{bmatrix} 5 \\ 4 \end{bmatrix}, & \text{jika } \mathbf{x}(k-1) \in R_{(2,1)}, \\ \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}(k-1) + \begin{bmatrix} 5 \\ 3 \end{bmatrix}, & \text{jika } \mathbf{x}(k-1) \in R_{(2,2)}, \end{cases}$$

dimana $R_{(1,1)} = \{x \in \mathbb{R}^2 : x_1 - x_2 \geq 3\}$, $R_{(2,1)} = \{x \in \mathbb{R}^2 : -1 \leq x_1 - x_2 \leq 3\}$, dan $R_{(2,2)} = \{x \in \mathbb{R}^2 : x_1 - x_2 \leq -1\}$, seperti yang ditunjukkan pada Gambar 2.1

(kanan). Daerah $R_{(1,2)}$ tidak muncul karena sama dengan himpunan kosong. Seperti yang dijelaskan di atas, *affine dynamics* yang berkorespondensi dengan suatu daerah dikarakterisasi oleh himpunan **g**. Sebagai contoh, *affine dynamics* dari $R_{(2,1)}$ diberikan oleh $x_1(k) = x_2(k-1) + 5$, $x_2(k) = x_1(k-1) + 4$.

2.3 *Difference Bound Matrices*

Bagian ini memperkenalkan definisi dari *Difference Bound Matrices*. DBM akan digunakan secara ekstensif dalam prosedur abstraksi sistem MPL.

Definisi 2.2 (*Difference Bound Matrices* (Adzkiya, 2015)) suatu DBM dalam $\mathbb{R}^n$ adalah interseksi dari berhingga himpunan yang didefinisikan sebagai $x_i - x_j \bowtie_{i,j} \alpha_{i,j}$, dimana $\bowtie_{i,j} \in \{<, \leq\}$ menyatakan tanda pertidaksamaan, bilangan $\alpha_{i,j} \in \mathbb{R} \cup \{+\infty\}$ menyatakan batas atas, untuk $i, j \in \{0, \dots, n\}$, dan nilai dari variabel khusus x_0 selalu sama dengan 0. Himpunan ini merupakan himpunan bagian dari $\mathbb{R}^n$ yang dikarakterisasi oleh nilai variabel $x_1, \dots, x_n$.

2.4 Sistem Transisi

Berikut adalah penjelasan tentang Sistem Transisi sebagaimana yang diulas di (Baier, 2008) dan (Adzkiya, 2015). Bagian ini memperkenalkan notasi dari sistem transisi, kelas standar dari model untuk menyatakan sistem *hardware* dan *software*.

Definisi 2.3 (Sistem Transisi (Baier, 2008)) Suatu sistem transisi TS dikarakterisasi oleh *quintuple* $(S, \longrightarrow, I, AP, L)$ dimana

- S adalah himpunan keadaan
- $\longrightarrow \subseteq S \times S$ adalah relasi transisi
- $I \subseteq S$ adalah himpunan keadaan awal
- AP adalah himpunan *atomic proposition*, dan
- $L : S \to 2^{AP}$ adalah fungsi pelabelan

TS disebut berhingga jika kardinalitas dari S dan AP berhingga.

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

Definisi 2.4 (*Direct Predecessors* dan *Direct Successors* (Baier, 2008)) Misal $TS = (S, \longrightarrow, I, AP, L)$ adalah suatu sistem transisi. Untuk $s \in S$, himpunan *direct succesors* dan *direct predecessors* dari s didefinisikan sebagai

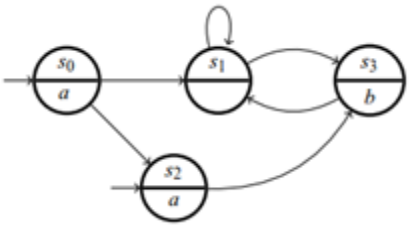
$$Post(s) = \{s' \in S : s \longrightarrow s'\} \quad \text{dan} \quad Pre(s) = \{s' \in S : s' \longrightarrow s\}.$$

Notasi untuk himpunan *direct successors* dan *direct predecessors* diperluas ke himpunan bagian dari S dalam cara yang jelas: untuk $C \subseteq S$,

$$Post(C) = \bigcup_{s \in C} Post(s) \quad \text{dan} \quad Pre(C) = \bigcup_{s \in C} Pre(s).$$

Suatu sistem transisi $TS = (S, \longrightarrow, I, AP, L)$ disebut *deterministic* jika $|I| \leq 1$ dan $|Post(s)| \leq 1$ untuk semua *state* s. *Trace* dari suatu *path* didefinisikan sebagai *finite* atau *infinite word* atas alfabet 2^{AP} yang diperoleh dengan menerapkan fungsi pelabelan pada semua *state* dalam *path*. Himpunan *traces* dari suatu sistem transisi TS didefinisikan sebagai himpunan *traces* yang terkait dengan semua *path* dalam TS.

Contoh 3 Perhatikan contoh sistem transisi dalam Gambar 2.2. Himpunan *state* $S = \{s_0, s_1, s_2, s_3\}$. Himpunan *initial state* $I = \{s_0, s_2\}$. Himpunan *atomic proposition* $AP = \{a, b\}$, dengan fungsi pelabelan: $L(s_0) = a$, $L(s_1) = \emptyset$, $L(s_2) = \{a\}$, $L(s_3) = \{b\}$.



Gambar 2.2 Representasi grafis dari sistem transisi pada contoh 3

2.5 Spesifikasi

Berikut adalah penjelasan tentang spesifikasi sebagaimana yang diulas di (Baier, 2008) dan (Adzkiya, 2015). Bagian ini memperkenalkan *Linear Temporal Logic* (LTL) dan *Computational Tree Logic* (CTL), merupakan *logical formalism* yang cocok untuk menentukan properti. Yang akan dibahas adalah sintaks dan semantik dari LTL dan CTL.

2.5.1 *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:

Definisi 2.5 (Sintaks dari *Linear Temporal Logic* (Baier, 2008)) Formula LTL atas himpunan *atomic proposition* AP dibentuk menurut grammar berikut:

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

dimana $a \in AP$.

Operator *Boolean* diantaranya ¬ (negasi), ∧ (konjungsi), dan ∨ (disjungsi), sedangkan operator *temporal* O (*next*), ⋃ (*until*), □ (*always*), dan ◊ (*eventually*). Operator *until* memungkinkan untuk memperoleh *temporal modalities* ◊ dan □. Modalitas O adalah *unary prefix operator* dan memerlukan satu formula LTL sebagai argumennya. Formula $\mathrm{O}\varphi$ 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 dimana φ_2 terpenuhi dan φ_1 terpenuhi pada semua momen sampai momen masa depan tersebut. Modalitas $\Diamond$ dan modalitas $\Box$ adalah *operator unary prefix* dan memerlukan sebuah formula LTL sebagai argumen: formula $\Diamond \varphi$ terpenuhi jika φ akan bernilai benar secepatnya di masa depan, sedankan formula $\Box \varphi$ 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. 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.

2.5.2 *Computation Tree Logic*

Definisi 2.6 (Sintaks *Computation Tree Logic* (Baier, 2008)) *State formulae* CTL atas himpunan *atomic proposition AP* dibentuk berdasar grammar berikut:

$$\Phi ::= true|a|\Phi_1 \wedge \Phi_2|\neg\Phi|\exists\varphi|\forall\varphi$$

dimana $a \in AP$ dan φ adalah suatu *path formulae*. *Path formulae* CTL dibentuk berdasar *grammar* berikut:

$$\varphi ::= \bigcirc\Phi|\Phi_1 \cup \Phi_2$$

dimana Φ, Φ_1 dan Φ_2 adalah *state formulae*.

CTL membedakan antara *state formulae* dan *path formulae*. *State formulae* menyatakan properti dari *state*, sedangkan *path formulae* menyatakan properti dari *path*, yakni barisan dari *state*. *Temporal operator* $\bigcirc$ dan $\cup$ mempunyai arti yang sama seperti dalam operator LTL dan *path*. *Path formulae* dapat berubah menjadi *state formulae* dengan *member prefix path quantifier* $\exists$ (dibaca "untuk beberapa *path*") atau *path quantifier* $\forall$ (dibaca "untuk semua *path*").

Definisi 2.7 (*Universal Fragment* dari CTL (Baier, 2008)) *Universal fragment* dari CTL, dinyatakan dengan ∀CTL, mengandung *state formulae* Φ dan *path formulae* φ yang diberikan, untuk $a \in AP$, dengan

$$\Phi ::= true|false|a|\neg a|\Phi_1 \wedge \Phi_2|\Phi_1 \vee \Phi_2|\forall\varphi$$

$$\varphi ::= O\Phi|\Phi_1 U \Phi_2|\Phi_1 R \Phi_2$$

Definisi 2.8 (*Existential Fragment* dari CTL (Baier, 2008)) *Existential fragment* dari CTL, dinyatakan dengan ∃CTL, mengandung *state formulae* Φ dan *path formulae* φ yang diberikan, untuk $a \in AP$, dengan

$$\Phi ::= true|false|a|\neg a|\Phi_1 \wedge \Phi_2|\Phi_1 \vee \Phi_2|\exists\varphi$$

$$\varphi ::= O\Phi|\Phi_1 U \Phi_2|\Phi_1 R \Phi_2$$

2.6 Ekuivalensi dan Abstraksi

Berikut adalah penjelasan tentang Ekuivalensi dan Abstraksi sebagaimana yang diulas di (Baier, 2008) dan (Adzkiya, 2015). Abstraksi adalah suatu konsep dasar yang memungkinkan analisis sistem transisi yang besar bahkan tak berhingga. Suatu abstraksi diidentifikasi oleh himpunan *state* abstrak $\hat{S}$; fungsi abstraksi f, yang menghubungkan setiap state konkrit s dari sistem transisi TS ke state abstrak yang merepresentasikannya; dan himpunan *atomic proposition AP* yang melabeli *state* abstrak dan konkrit.

Biasanya sistem transisi abstrak mensimulasikan sistem transisi konkrit yang bersesuaian. Hubungan simulasi digunakan sebagai dasar untuk teknik abstraksi, dimana idenya adalah untuk menggantikan model yang kongkrit untuk dilakukan verifikasi oleh model abstrak yang lebih kecil dan untuk memverifikasi model abstrak tersebut terhadap yang asli. Hubungan simulasi *preorder* di ruang *state* mengharuskan setiap kali *s*' mensimulasikan *s*, maka *state s*' dapat meniru semua perilaku bertahap dari *s*, namun sebaliknya tidak dijamin. Definisi formal dari tatanan simulasi diberikan di bawah ini.

Definisi 2.9 (Simulasi Order (Baier, 2008)) Misal $TS_i = (S_i, Act_i, \longrightarrow_i, I_i, AP, L_i)$, $i \in \{1, 2\}$, menjadi sistem transisi terhadap *AP*. Sebuah simulasi untuk (TS_1, TS_2) adalah relasi biner $R \subseteq S_1 \times S_2$ sehingga

1. untuk setiap $s_1 \in I_1$ terdapat $s_2 \in I_2$ sedemikian hingga $(s_1, s_2) \in R$
2. untuk semua $(s_1, s_2) \in R$ berlaku:
   (a) $L_1(s_1) = L_2(s_2)$
   (b) jika $s'_1 \in Post(s_1)$ maka terdapat $s'_2 \in Post(s_2)$ dengan $(s'_1, s'_2) \in R$.

Sebuah sistem transisi TS_1 disimulasikan oleh TS_2 (atau ekuivalen, TS_2 mensimulasikan TS_1) jika terdapat simulasi R untuk (TS_1, TS_2).

Kami secara singkat menguraikan ide-ide penting dari abstraksi yang diperoleh dengan memetakan himpunan *state* yang konkrit ke satu *state* yang 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 (Baier, 2008) 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 (Baier, 2008) Misalkan TS_2 mensimulasikan TS_1, diasumsikan TS_1 tidak memiliki *terminal state*, dan φ adalah properti *linear-time*. Jika TS_2 terpenuhi, maka TS_1 juga terpenuhi.

State s dalam sistem transisi TS disebut terminal jika dan hanya jika $Post(s) = \emptyset$. Hasil tersebut juga dipergunakan dalam formula LTL, karena formula LTL adalah suatu *linear-time property*. Secara umum kebalikan dari proposisi yang dijelaskan sebelumnya tidak benar. Lebih tepatnya, jika TS_2 tidak memenuhi φ, maka tidak dapat ditarik kesimpulan bahwa TS_1 tidak memenuhi φ karena *trace* dari *path* yang mengganggu φ mungkin menjadi perilaku sehingga TS_1 tidak dapat melakukan secara keseluruhan. Hasil yang serupa berlaku untuk suatu fragmen dari CTL, seperti yang ditunjukkan pada proposisi berikut:

Proposisi 2.3 (Baier, 2008) Misalkan TS_2 mensimulasikan TS_1, asumsikan TS_1 dan TS_2 tidak memiliki *terminal state*, misalkan φ merupakan suatu *universal fragment* CTL atau suatu *existential fragment* CTL. Jika TS_2 memnuhi φ, maka TS_1 juga memenuhi φ.

Tujuan ekuivalensi bisimulasi adalah untuk mengidentifikasi sistem transisi dengan struktur cabang yang sama, sehingga dapat mensimulasi satu dengan yang lainnya dengan cara bertahap.

Definisi 2.10 (Ekuivalensi Bisimulasi (Baier, 2008)) Untuk $i \in \{1,2\}$ misal TS_i adalah sistem transisi atas *AP*, yaitu $TS_i = (S_i, \longrightarrow_i, I_i, AP, L_i)$. Bisimulasi untuk (TS_1, TS_2) adalah relasi biner $\mathcal{R} \subseteq S_1 \times S_2$ sedemikian sehingga

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

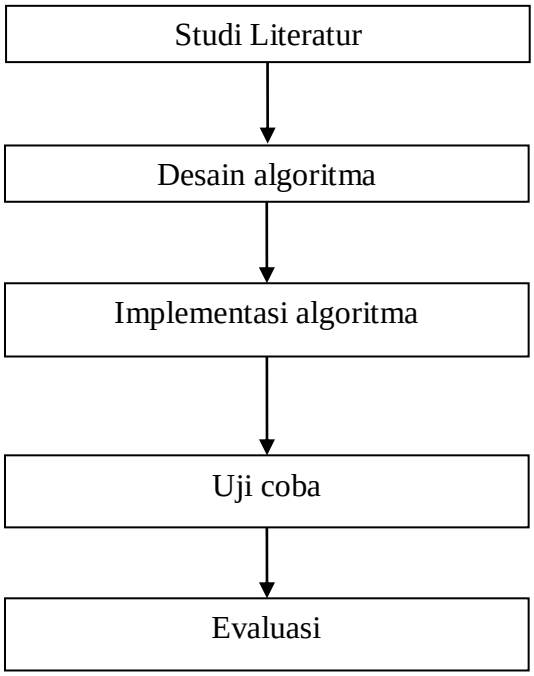
Sistem Transisi TS_1 dan TS_2 ekuivalen-bisimulasi jika terdapat bisimulasi $\mathcal{R}$ untuk (TS_1, TS_2).

Ekuivalensi bisimulasi menunjukkan kemungkinan *mutual* dan *stepwise simulation*. Ekuivalensi bisimulasi mempertahankan semua formula yang dapat dinyatakan dalam LTL dan CTL. Hasil ini memungkinkan model melakukan

pengecekan pada sistem transisi *bisimulation quotient* sambil menjaga kedua hasil *affirmative* dan negatif dari model checking.

BAB 3
METODE PENELITIAN

Pada penelitian ini, langkah-langkah yang dilakukan adalah studi literatur, desain algoritma, implementasi algoritma, uji coba, dan evaluasi. Alur metodologi dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur Metodologi

Studi Literatur

Pada tahap ini direview literatur-literatur mengenai abstraksi, sistem max plus, verifikasi, bahasa pemrograman C++.

Desain algoritma

Setelah itu didesain algoritma untuk abstraksi dengan struktur data tree tanpa fungsi rekursif.

Implementasi algoritma

Kemudian pada tahap ini dilakukan implementasi algoritma dan struktur data sistem abstraksi berhingga dengan menggunakan bahasa C++.

Uji coba dan Evaluasi

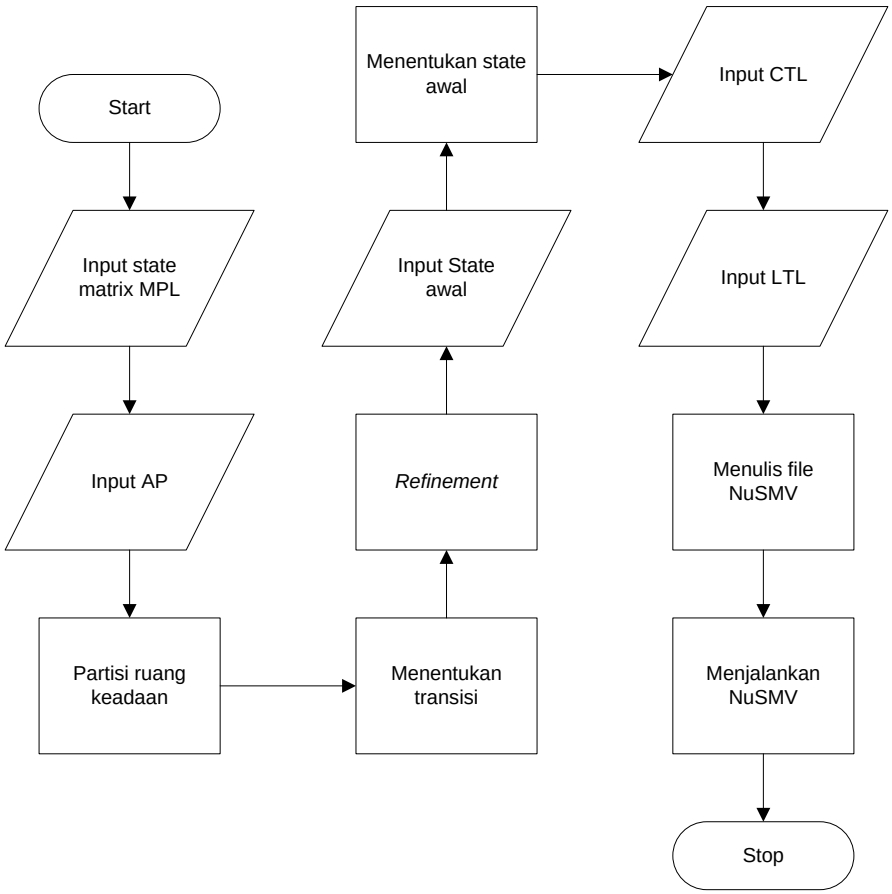
Pada tahap ini dilakukan pengujian terhadap perangkat lunak yang dibangun. Pengujian dilakukan dengan cara membandingkan *output* dari perangkat lunak terhadap implementasi sebelumnya yaitu VeriSiMPL 2.0. Apabila *output* yang dihasilkan sama, maka dianggap bahwa perangkat lunak telah diimplementasikan dengan benar. Sedangkan jika *output* yang dihasilkan berbeda, maka dilakukan perbaikan hingga *output* yang dihasilkan sama. Setelah itu dilakukan uji perbandingan waktu komputasi perangkat lunak yang telah dihasilkan dengan VeriSiMPL 2.0.

BAB 4
PEMBAHASAN

Pada bagian ini akan dijelaskan diagram alur proses alur proses abstraksi berhingga secara garis besar, desain kelas untuk menyelesaikan permasalahan abstraksi berhingga, dilanjutkan dengan implementasi kelas, dan yang terakhir adalah melakukan uji coba perbandingan dengan perangkat lunak abstraksi berhingga yang telah diimplementasikan sebelumnya yaitu VeriSiMPL 2.0.

4.1 Diagram Alir Proses

Diagram alir proses abstraksi berhingga sistem MPL autonomous secara garis besar bisa dilihat pada Gambar 4.1.



Gambar 4.1 Diagram alir proses abstraksi

Pertama *user* memasukkan *state matrix* model MPL *autonomous*. Matriks ini harus merupakan matriks *regular* (baris berhingga). Kemudian user diminta untuk menginputkan banyaknya himpunan *Atomic Proposition* (AP), dimana 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*, dimana pertama dilakukan partisi AP, 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 mempunya 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 dimana *success return* dari kegiatan ini adalah nilai *true* atau *false* terhadap verifikasi.

4.2 Desain Kelas

Untuk menyelesaikan permasalahan abstraksi berhingga ini, akan dibentuk kelas-kelas untuk menyatakan obejek-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*.

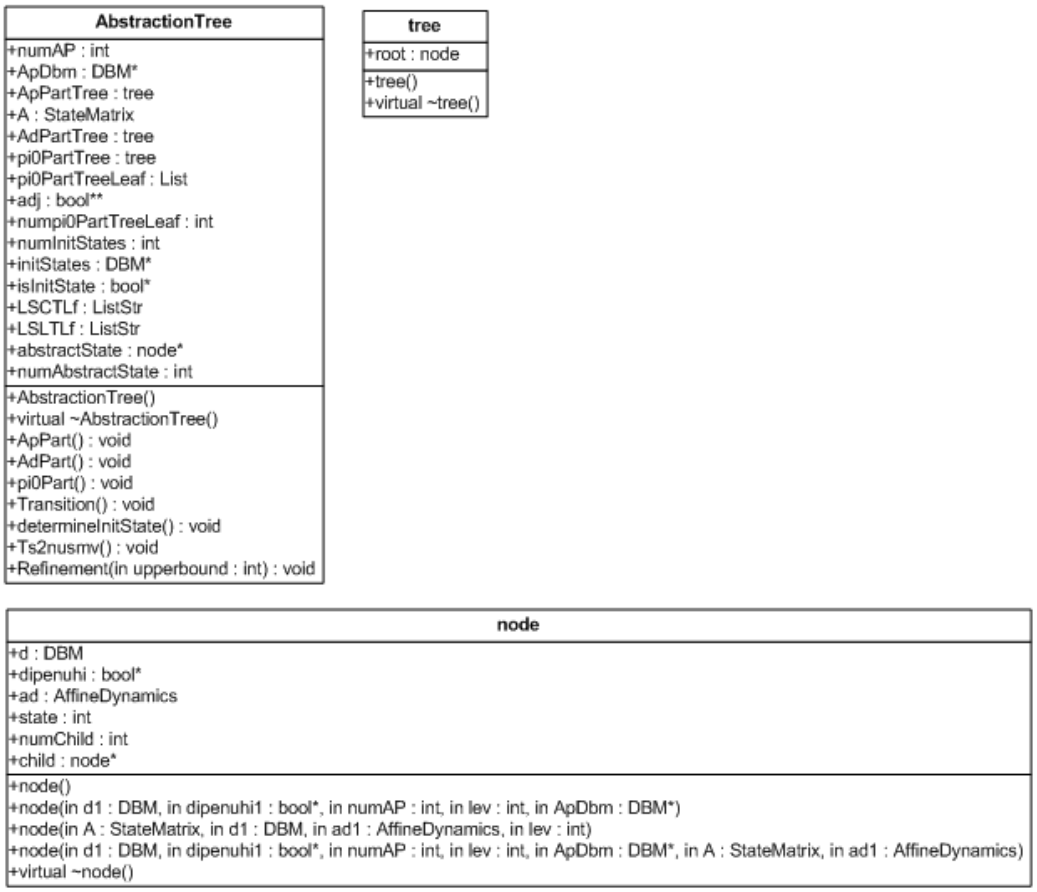
2. Kelas *Difference-Bound Matrices*

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

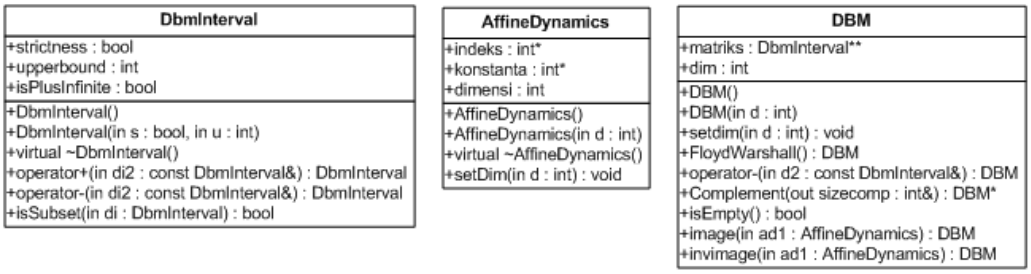
3. Kelas Tambahan

   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.*

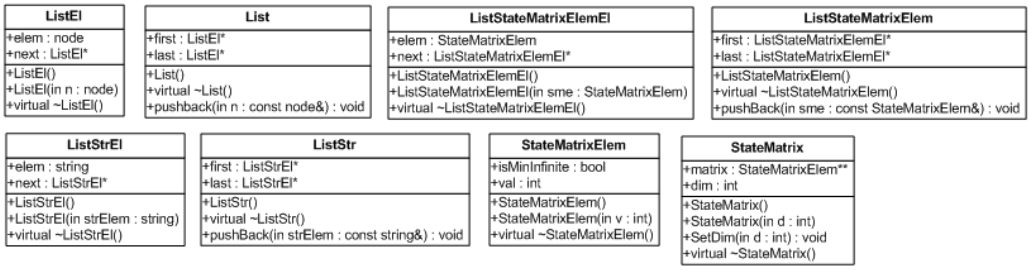
Desain UML kelas-kelas di atas dapat dilihat pada Gambar 4.2, Gambar 4.3, dan Gambar 4.4.



Gambar 4.2 Diagram kelas abstraksi



Gambar 4.3 Digram kelas *Difference-Bound Matrices*



Gambar 4.4 Diagram kelas-kelas Tambahan

4.3 Kelas Kelas Tambahan

4.3.1 ListEl

Kelas ListEl terdiri dari data *elem* bertipe *node* dan data *next* bertipe pointer ke *ListEl*. Data *elem* berisi informasi yang dibawa setiap elemen *list* yaitu berupa *node* dari suatu *tree*. Sedangkan data *next* berisi alamat dari elemen berikutnya dalam *list*.

```
class ListEl
{
   node elem;
   ListEl *next;

   ListEl();
   ListEl(node n);
   virtual ~ListEl();
};
```

4.3.2 List

Kelas *List* terdiri atas data *first* dan *last* bertipe pointer ke *ListEl*, beserta metode *pushBack* bertipe *void*. Data *first* berisi alamat elemen pertama dari list,

sedangkan data *last* berisi alamat elemen terakhir dari list. Metode *pushBack* melakukan penambahan elemen baru di akhir list.

```
class List
{
   ListEl *first, *last;

   List();
   virtual ~List();
   void pushBack(const node& n);
};
```

4.3.3 StateMatrixElem

Kelas *StateMatrixElem* berisi data *isMinInfinite* bertipe *bool* dan data *val* bertipe *integer*. Data *val* adalah nilai dari elemen matriks. Data *isMinInfinite* bernilai *true* jika *val* bernilai negative tak hingga, dan bernilai *false* jika *val* berhingga.

```
class StateMatrixElem
{
   bool isMinInfinite;
   int val;

   StateMatrixElem();
   StateMatrixElem(int v);
   virtual ~StateMatrixElem();
};
```

4.3.4 StateMatrix

Kelas *StateMatrix* terdiri atas data *matriks* berukuran *dim* x *dim*, yang setiap elemennya bertipe *StateMatrixElem*. Ukuran dari matriks persegi akan ditentukan secara dinamis oleh fungsi *SetDim*.

```
class StateMatrix
{
   StateMatrixElem **matrix;
   int dim;

   StateMatrix();
   StateMatrix(int d);
   void SetDim(int d);
   virtual ~StateMatrix();
};
```

4.4 DBM

4.4.1 DBMInterval

```
class DbmInterval
{
   bool strictness;
   int upperbound;
   bool isPlusInfinite;

   DbmInterval();
   DbmInterval(bool s,int u);
   virtual ~DbmInterval();
   DbmInterval operator+(const DbmInterval& di2) const;
   DbmInterval operator-(const DbmInterval& di2) const;
   bool isSubset(DbmInterval di);
};
```

Kelas *DbmInterval* merupakan komponen dari kelas *DBM*, yaitu $x_i - x_j \bowtie_{i,j} \alpha_{i,j}$, dimana $\bowtie_{i,j} \in \{<, \leq\}$. Variabel *strictness* menentukan tanda pertidaksamaan yang digunakan, bernilai *true* apabila bertanda $\leq$ dan bernilai *false* apabila bertanda $<$. Nilai dari $\alpha_{i,j}$ disimpan dalam variabel *upperbound*, dan variabel *isPlusInfinite* menentukan apakah $\alpha_{i,j}$ positif tak berhingga atau tidak. Jika variabel *isPlusInfinite* bernilai true maka $\alpha_{i,j}$ positif tak berhingga. Jika variabel *isPlusInfinite* bernilai *false* maka $\alpha_{i,j}$ berhingga.

A. Operator +

Operator + digunakan untuk mendefinisikan operasi gabungan dua DbmInterval sebagai berikut:

a. $\{x_i - x_j \leq \alpha\} + \{x_j - x_k \leq \beta\} = \{x_i - x_k \leq \alpha + \beta\}$
b. $\{x_i - x_j < \alpha\} + \{x_j - x_k \leq \beta\} = \{x_i - x_k < \alpha + \beta\}$
c. $\{x_i - x_j \leq \alpha\} + \{x_j - x_k < \beta\} = \{x_i - x_k < \alpha + \beta\}$
d. $\{x_i - x_j < \alpha\} + \{x_j - x_k < \beta\} = \{x_i - x_k < \alpha + \beta\}$

Operator ini akan digunakan pada proses *Floyd Warshall* yang merupakan metode pada kelas DBM.

B. Operator –

Operator – mendefinisikan operasi irisan dari dua DbmInterval. Sifat operasi ini sama seperti pada operasi irisan dua interval bilangan riil. Irisan dari dua DbmInterval sebagai berikut:

a. $\{x_i - x_j \leq \alpha\} - \{x_i - x_j \leq \beta\} = \{x_i - x_j \leq \min(\alpha, \beta)\}$

b. $\{x_i - x_j < \alpha\} - \{x_i - x_j \leq \beta\} = \begin{cases} \{x_i - x_j < \alpha\}, & \text{jika } \alpha < \beta \\ \{x_i - x_j < \alpha\}, & \text{jika } \alpha = \beta \\ \{x_i - x_j \leq \beta\}, & \text{jika } \alpha > \beta \end{cases}$

c. $\{x_i - x_j < \alpha\} - \{x_i - x_j < \beta\} = \{x_i - x_k < \min(\alpha, \beta)\}$

Misalnya, interval $(-\infty, 10] \cap (-\infty, 5) = (-\infty, 5)$.

C. isSubset

Fungsi *isSubset* menentukan apakah suatu DbmInterval merupakan subset dari DbmInterval yang lain. Sifat operasi ini sama halnya seperti pada interval bilangan riil. Misal ingin ditentukan apakah interval 1 $\{x_i - x_j \bowtie_1 \alpha\}$ merupakan subset dari interval 2 $\{x_i - x_j \bowtie_2 \beta\}$

a. Jika $\beta > \alpha$ maka interval 1 subset dari interval 2
b. Jika $\beta < \alpha$ maka interval 1 bukan subset dari interval 2
c. Jika $\beta = \alpha$,
    i. Jika $\bowtie_2 = \leq$, maka interval 1 subset dari interval 2
    ii. Jika $\bowtie_2 = <$, maka:
        - Jika $\bowtie_1 = <$, maka interval 1 subset dari interval 2
        - Jika $\bowtie_1 = \leq$, maka interval 1 bukan subset dari interval 2

4.4.2 DBM

Kelas *DBM* terdiri atas data *matriks* berukuran (*dim*+1) x (*dim*+1), yang setiap elemennya bertipe *DbmInterval*. Ukuran dari matriks persegi akan ditentukan secara dinamis oleh fungsi *setdim*.

```
class DBM
{
   DbmInterval **matriks;
   int dim;
```

```
    DBM();
    DBM(int d);
    ~DBM();
    void setdim(int d);
    DBM FloydWarshall();
    DBM operator-(const DBM& d2) const;
    DBM* Complement(int& sizecomp);
    bool isEmpty();
    DBM image(AffineDynamics ad1);
    DBM invimage(AffineDynamics ad1);
};
```

A. FloydWarshall

Fungsi *FloydWarshall* menentukan representasi bentuk kanonik dari suatu DBM. Bentuk kanonik akan digunakan untuk: 1) menghitung image dari sebuah DBM terhadap sebuah affine dynamics; 2) menghitung inverse image dari sebuah DBM terhadap sebuah affine dynamics; 3) Menentukan apakah sebuah DBM kosong atau tidak. Potongan kode program dari fungsi FloydWarshall adalah sebagai berikut:

```
DBM DBM::FloydWarshall()
{
    DBM temp(dim);
    temp.matriks=matriks;
    DbmInterval val;

    for(int i=0;i<=dim;i++)
       for(int j=0;j<=dim;j++)
          for(int k=0;k<=dim;k++)
          {
             val=temp.matriks[i][k]+temp.matriks[k][j];
             if(val.isSubset(temp.matriks[i][j]))
                temp.matriks[i][j]=val;
          }
    return temp;
}
```

Contoh Perhatikan DBM berikut: $\{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_4 \leq -3, x_2 - x_1 \leq -3, x_2 - x_4 \leq -3, x_3 - x_1 \leq 2\}$. Dalam deskripsi himpunan ini, kita telah menghilangkan pertidaksamaan yang tidak terhingga, misalnya $x_3 - x_4 < +\infty$. Dapat ditunjukkan bahwa representasi bentuk kanoniknya diberikan oleh $\{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_4 \leq -3, x_2 - x_1 \leq -3, x_2 - x_4 \leq -6, x_3 - x_1 \leq 2, x_3 - x_4 \leq -1\}$. Perhatikan bahwa batas atas dari $x_2 - x_4$ dan $x_3 - x_4$ lebih kecil dibandingkan dengan DBM semula.

B. Operator –

Operator – digunakan untuk mendefinisikan operasi irisan pada dua DBM. Irisan dari dua DBM diperoleh dari irisan semua elemen dari matriksnya pada posisi yang bersesuaian. Kode program dari fungsi ini adalah sebagai berikut:

```
DBM DBM::operator-(const DBM& d2) const
{
   DBM temp(dim);
   for(int i=0;i<=dim;i++)
      for(int j=0;j<=dim;j++)
         temp.matriks[i][j]=matriks[i][j]-d2.matriks[i][j];
   return temp;
}
```

Contoh Misalkan $D = \{\mathbf{x} \in \mathbb{R}^3 : x_1 - x_2 \leq 5, x_2 - x_3 < 3\}$ dan $E = \{\mathbf{x} \in \mathbb{R}^3 : x_1 - x_3 < 7, x_2 - x_3 \leq 3\}$. Irisan dari D dan E diberikan oleh $F = \{\mathbf{x} \in \mathbb{R}^3 : x_1 - x_2 \leq 5, x_1 - x_3 < 7, x_2 - x_3 < 3\}$.

C. Complement

Fungsi *Complement* menghitung komplemen dari suatu DBM. Secara umum, komplemen dari sebuah DBM merupakan gabungan dari beberapa DBM. Kode program dari fungsi ini adalah sebagai berikut:

```
DBM* DBM::Complement(int &sizecomp)
{
   int m=0;
   for(int i=0;i<=dim;i++)
      for(int j=0;j<=dim;j++)
         if(i!=j && !matriks[i][j].isPlusInfinite)
            m++;
   sizecomp=m;
   if(m==0)
      return 0;
   int *rowPos,*colPos,k=-1;
   rowPos=new int[m];
   colPos=new int[m];
   for(int i=0;i<=dim;i++)
      for(int j=0;j<=dim;j++)
         if(i!=j && !matriks[i][j].isPlusInfinite)
         {
            k++;
            rowPos[k]=i;
            colPos[k]=j;
         }
   DBM *comp;
```

```
    comp=new DBM[m];
    for(int i=0;i<m;i++)
    {
       comp[i].setdim(dim);
       for(int j=0;j<i;j++)
       {
          DBM set(dim);
          set.matriks[rowPos[j]][colPos[j]]=matriks[rowPos[j]
          ][colPos[j]];
          comp[i]=comp[i]-set;
       }
       DBM setc(dim);
       setc.matriks[colPos[i]][rowPos[i]].strictness=!matrik
       s[rowPos[i]][colPos[i]].strictness;
       setc.matriks[colPos[i]][rowPos[i]].upperbound=-
       1*matriks[rowPos[i]][colPos[i]].upperbound;
       setc.matriks[colPos[i]][rowPos[i]].isPlusInfinite=fal
       se;
       comp[i]=comp[i]-setc;
    }
    return comp;
}
```

Contoh Tentukan komplemen dari $D = \{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_2 \leq 5, x_2 - x_3 < 3, x_3 - x_4 < 1\}$. DBM D adalah suatu suatu irisan dari 3 himpunan, yaitu $set_1 = \{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_2 \leq 5\}$, $set_2 = \{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_2 < 3\}$, $set_3 = \{\mathbf{x} \in \mathbb{R}^4 : x_3 - x_4 < 1\}$. Langkah pertama adalah menghitung komplemen dari setiap himpunan, yaitu $set_1^c = \{\mathbf{x} \in \mathbb{R}^4 : x_2 - x_1 < -5\}$, $set_2^c = \{\mathbf{x} \in \mathbb{R}^4 : x_3 - x_2 \leq -3\}$, $set_3^c = \{\mathbf{x} \in \mathbb{R}^4 : x_4 - x_3 \leq -1\}$. Komplemen dari D diperoleh sebagai $set_1^c \cup (set_1 \cap set_2^c) \cup (set_1 \cap set_2 \cap set_3^c)$, yaitu $\{\mathbf{x} \in \mathbb{R}^4 : x_2 - x_1 < -5\} \cup \{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_2 \leq 5, x_3 - x_2 \leq -3\} \cup \{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_2 \leq 5, x_2 - x_3 < 3, x_4 - x_3 \leq -1\}$.

Prosedur umum untuk menentukan komplemen dari sebuah DBM D dalam $\mathbb{R}^n$ adalah sebagai berikut. Misalkan DBM D adalah irisan dari m himpunan, dinyatakan dengan $set_1, \ldots, set_m$. Komplemen dari D diberikan oleh $\bigcup_{i=1}^{m}((\bigcap_{j=1}^{i-1} set_j) \cap set_i^c)$, dimana $\bigcap_{j=1}^{0} set_j$ adalah himpunan kosong pada $\mathbb{R}^n$. Jumlah maksimal operasinya berorder $O(n^3)$.

D. isEmpty

Fungsi *isEmpty* menentukan apakah suatu DBM kosong atau tidak. Kita gambarkan suatu prosedur untuk mengecek apakah suatu DBM kosong. Dengan

menggunakan representasi graf potensial, himpunan constraint yang unfeasible adalah yang membentuk lintasan dengan bobot negatif dalam graf. Dengan kata lain, lintasan ini berkorespondensi dengan suatu constraint $x_i - x_i \bowtie_{i,i} \alpha_{i,i}$ dengan $\bowtie_{i,i} \in \{<, \leq\}$ dan $\alpha_{i,i} < 0$, yang unfeasible. Sebagai konsekuensinya, untuk menguji apakah suatu DBM kosong atau tidak, kita cukup memeriksa keberadaan sirkuit tersebut: hal ini dapat diperoleh dengan menggunakan algoritma Bellman-Ford. Ketika suatu DBM dalam bentuk kanonik, pengujian untuk negative cycles dapat direduksi untuk mengecek apakah terdapat i sedemikian sehingga $\bowtie_{i,i}$ adalah $<$ atau $\alpha_{i,i} < 0$. Dalam hal ini, kompleksitas pengecekan kekosongan sebuah DBM adalah linier.

Kode program dari fungsi ini adalah sebagai berikut

```
bool DBM::isEmpty()
{
   DBM thisD(dim),FW(dim);
   thisD.matriks=matriks;
   FW=thisD.FloydWarshall();
   for(int i=0;i<=dim;i++)
      if(FW.matriks[i][i].upperbound<0)
         return true;
      else if(FW.matriks[i][i].upperbound==0 &&
      FW.matriks[i][i].strictness==false)
         return true;
   return false;
}
```

Contoh Perhatikan DBM berikut $\{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_4 \leq -3, x_2 - x_1 \leq -3, x_2 - x_4 \leq -3, x_3 - x_1 \leq 2, x_0 - x_0 = 0, x_1 - x_1 = 0, x_2 - x_2 = 0, x_3 - x_3 = 0, x_4 - x_4 = 0\}$. Representasi bentuk kanonik dari DBM tersebut adalah $\{\mathbf{x} \in \mathbb{R}^4 : x_1 - x_4 \leq -3, x_2 - x_1 \leq -3, x_2 - x_4 \leq -6, x_3 - x_1 \leq 2, x_3 - x_4 \leq -1, x_0 - x_0 = 0, x_1 - x_1 = 0, x_2 - x_2 = 0, x_3 - x_3 = 0, x_4 - x_4 = 0\}$. Karena batas atas dari $x_i - x_i$ untuk $i \in \{0, 1, 2, 3, 4\}$ adalah 0 dan bertanda pertidaksamaan $\leq$, maka DBM tidak kosong.

E. image

Setiap *region* dan *affine dynamics* dari sistem PWA yang dibentuk oleh matriks max-plus baris berhingga adalah suatu DBM dalam $\mathbb{R}^n$. Tiap-tiap *affine dynamics* (4) dapat membentuk suatu DBM dalam $\mathbb{R}^{2n}$ yang terdiri dari titik-titik

$(\mathbf{x}(k-1), \mathbf{x}(k)) \in \mathbb{R}^n \times \mathbb{R}^n$ sedemikian sehingga $\mathbf{x}(k)$ adalah image dari $\mathbf{x}(k-1)$, yaitu $\mathbf{x}(k) = A \otimes \mathbf{x}(k-1)$. Lebih tepatnya, DBM yang diperoleh dengan menuliskan ulang ekspresi dari *affine dynamics* sebagai $\bigcap_{i=1}^{n}\{\mathbf{x}(k-1), \mathbf{x}(k)\}: x_i(k) - x_{g_i}(k-1) \leq A(i, g_i)\} \cap \bigcap_{i=1}^{n}\{(\mathbf{x}(k-1), \mathbf{x}(k)): x_i(k) - x_{g_i}(k-1) \leq A(i, g_i)\}$.

Prosedur umum untuk menghitung image dari suatu DBM dalam $\mathbb{R}^n$: 1) hitung *cross product* dari DBM dan $\mathbb{R}^n$; 2) iriskan *cross product* dengan DBM yang dibentuk oleh ekspresi dari *affine dynamics*; 3) hitung bentuk kanonik dari irisan yang diperoleh; dan akhirnya 4) proyeksikan representasi bentuk kanonik atas $\{x_1(k), \ldots, x_n(k)\}$. Jumlah maksimal operasi dari prosedur secara kritis tergantung pada langkah ke-3, yaitu $O(n^3)$.

Fungsi *image* menghitung image dari suatu DBM terhadap sebuah *affine dynamics*. Kode program fungsi ini adalah sebagai berikut

```
DBM DBM::image(AffineDynamics ad1)
{
   DBM temp(2*dim);
   for(int i=0;i<=dim;i++)
      for(int j=0;j<=dim;j++)
         temp.matriks[i][j]=matriks[i][j];
   for(int i=0;i<dim;i++)
   {
      DbmInterval di1(true,ad1.konstanta[i]),di2(true,-
      1*ad1.konstanta[i]);
      temp.matriks[ad1.indeks[i]][i+dim+1]=di1;
      temp.matriks[i+dim+1][ad1.indeks[i]]=di2;
   }
   temp=temp.FloydWarshall();
   DBM img(dim);
   for(int i=1;i<=dim;i++)
   {
      img.matriks[0][i]=temp.matriks[0][i+dim];
      img.matriks[i][0]=temp.matriks[i+dim][0];
      for(int j=1;j<=dim;j++)
         img.matriks[i][j]=temp.matriks[i+dim][j+dim];
   }
   return img;
}
```

Contoh Hitung *image* dari $\{\mathbf{x} \in \mathbb{R}^2: 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, x_1 - x_2 \leq 0\}$ terhadap *affine dynamics* $x_1' = x_2 + 5, x_2' = x_2 + 3$. *Cross product* dari DBM dan $\mathbb{R}^2$ adalah $\{(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^4: 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, x_1 - x_2 \leq 0\}$. Irisan dari *cross*

product dan DBM yang dibentuk oleh *affine dynamics* adalah $\{(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^4 : 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, x_1 - x_2 \leq 0, x_1' - x_2 = 5, x_2' - x_2 = 3, x_2' - x_1' = -2\}$. Proyeksi terhadap $\{x_1', x_2'\}$ dihitung dengan menghilangkan semua pertidaksamaan yang mengandung x_1 atau x_2, yang menghasilkan $\{\mathbf{x}' \in \mathbb{R}^2 : 5 \leq x_1' \leq 6, 3 \leq x_2' \leq 4, x_2' - x_1' = -2\}$.

F. invimage

Prosedur untuk menghitung *inverse image* dari suatu DBM dalam $\mathbb{R}^n$ hampir sama seperti prosedur untuk menghitung *image*. Perbedaannya terletak pada langkah terakhir: representasi bentuk kanonik diproyeksikan terhadap $\{x_1(k-1), \ldots, x_n(k-1)\}$. Jumlah maksimal operasi untuk menghitung *inverse image* sama seperti menghitung *image*.

Fungsi *invimage* menghitung invers image dari suatu DBM. Kode program dari fungsi ini adalah sebagai berikut

```
DBM DBM::invimage(AffineDynamics ad1)
{
   DBM temp(2*dim);
   for(int i=1;i<=dim;i++)
   {
      temp.matriks[0][i+dim]=matriks[0][i];
      temp.matriks[i+dim][0]=matriks[i][0];
      for(int j=1;j<=dim;j++)
         temp.matriks[i+dim][j+dim]=matriks[i][j];
   }
   for(int i=0;i<dim;i++)
   {
      DbmInterval di1(true,ad1.konstanta[i]),di2(true,-
      1*ad1.konstanta[i]);
      temp.matriks[ad1.indeks[i]][i+dim+1]=di1;
      temp.matriks[i+dim+1][ad1.indeks[i]]=di2;
   }
   temp=temp.FloydWarshall();
   DBM invimg(dim);
   for(int i=0;i<=dim;i++)
      for(int j=0;j<=dim;j++)
         invimg.matriks[i][j]=temp.matriks[i][j];
   return invimg;
}
```

4.5 AffineDynamics

Bentuk umum dari *affine dynamics* adalah:

$$x_i(k) = x_{g_i}(k-1) + A(i, g_i), \qquad i \in \{1, \dots, n\}.$$

Untuk setiap i, $g_i \in \{1, \dots, n\}$. Kelas *AffineDynamics* terdiri atas vektor *indeks* dan vektor *konstanta*. Vektor *indeks* bernilai 1, 2, 3, …, n, dimana n adalah dimensi dari *AffineDynamics*. Nilai dari g_i disimpan dalam vektor *indeks*. Vektor *konstanta* bernilai bilangan bulat. Nilai dari $A(i, g_i)$ disimpan di dalam vektor *konstanta*. Nilai n disimpan dalam variabel *dimensi*.

```
class AffineDynamics
{
   int *indeks;
   int *konstanta;
   int dimensi;

   AffineDynamics();
   AffineDynamics(int d);
   virtual ~AffineDynamics();
   void setDim(int d);
};
```

4.6 Mempartisi Ruang Keadaan (Pi0)

Proses pembentukan partisi ruang keadaan diimplementasikan dalam bentuk konstruktor kelas *node*. Konstruktor tersebut dijalankan ketika dibentuk node *root* pada kelas *tree*. Pembentukan *node* akan terjadi secara rekursif sebanyak level dari tree tersebut. Jumlah level dari Pi0 *Tree Partition* sebanyak *nAP* + *dimA* + 1, dimana *nAP* adalah banyak anggota himpunan AP (*Atomic Proposition*, dan *dimA* merupakan dimensi dari state matriks MPL. Tipe data abstrak dari kelas *tree* dan *node* adalah sebagai berikut

```
class tree
{
   node root;

   tree();
   virtual ~tree();
};
class node
{
   DBM d;
   bool *dipenuhi;
   AffineDynamics ad;
   int state;
```

```
    int numChild;
    node *child;

    node();
    node(DBM d1, bool *dipenuhi1, int numAP, int lev, DBM
    *ApDbm, StateMatrix A, AffineDynamics ad1);
    virtual ~node();
};
```

Berikutnya akan dijelaskan data-data yang tersimpan dalam kelas *node*. Variabel *d* menyatakan DBM yang direpresentasikan oleh *node* tersebut. Variabel *dipenuhi* merupakan suatu array bertipe *Boolean*, dengan anggota sebanyak nAP. Variabel ini menyatakan himpunan proposisi atomik yang dipenuhi oleh DBM. Jika DBM memenuhi proposisi atomik yang ke-i, maka elemen ke-i dari variabel dipenuhi bernilai *true*. Jika DBM tidak memenuhi proposisi atomik yang ke-i, maka elemen ke-i dari variabel dipenuhi bernilai *false*. Variabel *ad* berisi *affine dynamics* yang aktif di DBM. Variabel *state* digunakan hanya ketika *node* berada di *leaf*. Pada kasus ini, variabel berisi penomoran yang unik dari *leaf* yang akan digunakan untuk identifikasi *state* abstrak. Variabel *numChild* berisi jumlah anak dari *node* ini. Jika variabel ini bernilai lebih dari nol, maka *node* ini bukan *leaf*. Jika variabel ini bernilai nol, maka node ini adalah *leaf*. Variabel *child* merupakan array dari *node*. Variabel ini berisi anak-anak dari *node* ini.

Proses pembentukan Pi0 *tree partition* ini diawali dengan *AP partition* dilanjutkan dengan *AD partition.* Pemanggilan konstruktor ini dilakukan pada fungsi *pi0Part* di kelas *AbstractionTree*. Tipe data abstrak dari kelas *AbstractionTree* adalah sebagai berikut

```
class AbstractionTree
{
    int numAP;
    DBM *ApDbm;
    StateMatrix A;
    tree pi0PartTree;
    List pi0PartTreeLeaf;
    bool **adj;
    int numpi0PartTreeLeaf;

    AbstractionTree();
    virtual ~AbstractionTree();
    void pi0Part();
    void Transition();
};
```

Berikutnya, akan dijelaskan arti dari setiap data. Variabel *numAP* berisi jumlah proposisi atomik. Variabel *ApDBM* merupakan array dari *DBM*, yang berjumlah sama dengan proposisi atomik. Proposisi atomik ke-i dipenuhi oleh *state-state* yang berada di dalam DBM ke-i. Variabel *A* berisi matriks keadaan dari sistem max plus. Variabel *pi0PartTree* berisi *space-partitioning tree* yang akan dibangun. Variabel *pi0PartTreeLeaf* berisi *node-node* yang menjadi *leaf* pada *space-partitioning tree*. Variabel *adj* merupakan *adjacency matrix*, yang setiap elemennya bertipe *Boolean*. Jika elemen (i,j) bernilai *true*, maka terdapat transisi dari *state* abstrak ke-j menuju ke *state* abstrak ke-i. Jika elemen (i,j) bernilai *false*, maka tidak terdapat transisi dari *state* abstrak ke-j menuju ke *state* abstrak ke-i. Variabel *numPi0PartTreeLeaf* berisi jumlah *node* yang menjadi *leaf*, yang sama dengan jumlah state abstrak.

Fungsi *pi0part* bertujuan untuk membangun *space-partitioning tree*. Kode program dari fungsi pi0Part adalah sebagai berikut

```
void AbstractionTree::pi0Part()
{
   DBM Rn(A.dim);
   bool *dipenuhi;
   dipenuhi=new bool[numAP];
   for(int i=0;i<numAP;i++)
      dipenuhi[i]=false;
   AffineDynamics ad(A.dim);
   node temp(Rn,dipenuhi,numAP,0,ApDbm,A,ad);
   pi0PartTree.root=temp;
}
```

Proses pembuatan *space-partitioning tree* terdiri dari 2 tahap. Tahap pertama adalah membangun *AP tree*. Tahap kedua adalah membangun *AD tree*. Gabungan dari *AP* dan *AD tree* merupakan *space-partitioning tree*.

Proses pembuatan *AP tree* adalah sebagai berikut. Pertama buat *node root* yang merepresentasikan $\mathbb{R}^n$. Kemudian, untuk setiap proposisi atomik, dibentuk partisi dari $\mathbb{R}^n$, dengan menggabungkan DBM dan komplemennya. Setelah itu, setiap partisi di iriskan dengan *leaf*: jika irisan tidak kosong, maka dibuat anak dari *leaf* tersebut; jika irisan kosong, maka tidak dibuat anak dari *leaf* tersebut. Hal ini dilakukan untuk setiap proposisi atomik. Pada akhirnya, proses ini akan menghasilkan *tree* dengan level sebanyak jumlah proposisi atomik ditambah 1.

Proses pembuatan AD *tree* yang berada di bawah (merupakan anak) dari *AP tree* adalah sebagai berikut. Untuk setiap baris, pada matriks keadaan, dibuat sebuah partisi dari $\mathbb{R}^n$, dengan cara sebagai berikut. Misalnya baris pertama diberikan oleh $[2 \quad 5]$. Ketika menghitung DBM yang diwakili oleh elemen pertama, diperoleh $2 + x_1 > 5 + x_2$. Tanda pertidaksamaan adalah *strict* karena 1 (indeks elemen) kurang dari 2. Untuk elemen yang kedua, diperoleh $5 + x_2 \geq 2 + x_1$. Tanda pertidaksamaan adalah *nonstrict* (tidak strict) karena 2 (index elemen) lebih dari 1. Disini diasumsikan bahwa himpunan irisan ikut dengan indeks elemen yang lebih besar.

Fungsi *pi0Part* membuat variabel *temp* yang bertipe node. Kemudian konstruktor dari kelas *node* akan dijalankan. Di dalam konstruktor node, terdapat implementasi algoritma untuk membentuk pi0 *partition* (lihat penjelasan di paragraf sebelumnya). Proses pembuatan *AP* dan *AD partition* dibedakan berdasarkan level dari *node*. Jika level kurang dari *numAP*, maka proses tersebut termasuk *AP partition*, sedangkan jika level lebih dari atau sama dengan *numAP*, maka proses partisi tersebut termasuk AD *partition*. Kode program dari konstruktor dari *node* untuk membentuk *tree* ini adalah sebagai berikut

```
node::node(DBM d1, bool *dipenuhi1, int numAP, int lev, DBM
*ApDbm, StateMatrix A, AffineDynamics ad1)
{
   d=d1;
   dipenuhi=new bool[numAP];
   for(int i=0;i<numAP;i++)
      dipenuhi[i]=dipenuhi1[i];
   ad=ad1;
   numChild=0;
   if(lev<numAP)
   {
      int numApDbmComp;
      DBM *ApDbmComp=ApDbm[lev].Complement(numApDbmComp);
      DBM temp=d1-ApDbm[lev];
      DBM *DbmChild=new DBM[numApDbmComp+1];
      bool intersectwithApDbmIsEmpty=true;
      if(!temp.isEmpty())
      {
         intersectwithApDbmIsEmpty=false;
         DbmChild[0]=temp;
         numChild++;
      }
      for(int i=0;i<numApDbmComp;i++)
      {
```

```
      temp=d1-ApDbmComp[i];
      if(!temp.isEmpty())
      {
         DbmChild[numChild]=temp;
         numChild++;
      }
   }
   child=new node[numChild];
   AffineDynamics ad2(A.dim);
   for(int i=0;i<numChild;i++)
   {
      if(!intersectwithApDbmIsEmpty && i==0)
         dipenuhi[lev]=true;
      node tempchild1(DbmChild[i], dipenuhi, numAP,
      lev+1, ApDbm, A, ad2);
      child[i]=tempchild1;
      dipenuhi[lev]=false;
   }
}
else if(lev<numAP+A.dim)
{
   DBM *DbmChild=new DBM[A.dim];
   AffineDynamics *AffDynChild=new AffineDynamics[A.dim];
   for(int i=0;i<A.dim;i++)
   {
      DBM Ri(A.dim);
      if(!A.matrix[lev-numAP][i].isMinInfinite)
      {
         for(int j=0;j<A.dim;j++)
         {
            if(i!=j && !A.matrix[lev-
            numAP][j].isMinInfinite)
            {
               if(i<j)
                  Ri.matriks[i+1][j+1].strictness=true;
               else
                  Ri.matriks[i+1][j+1].strictness=false;
               Ri.matriks[i+1][j+1].isPlusInfinite=false;
               Ri.matriks[i+1][j+1].upperbound=A.matrix[
               lev-numAP][i].val-A.matrix[lev-
               numAP][j].val;
            }
         }
         DBM temp(A.dim);
         temp=d1-Ri;
         if(!temp.isEmpty())
         {
            DbmChild[numChild]=temp;
            AffDynChild[numChild].setDim(A.dim);
            for(int k=0;k<A.dim;k++)
            {
```

```
                    AffDynChild[numChild].indeks[k]=ad1.indek
                    s[k];
                    AffDynChild[numChild].konstanta[k]=ad1.ko
                    nstanta[k];
                }
                AffDynChild[numChild].indeks[lev-numAP]=i+1;
                AffDynChild[numChild].konstanta[lev-
                numAP]=A.matrix[lev-numAP][i].val;
                numChild++;
            }
         }
      }
      child=new node[numChild];
      for(int i=0;i<numChild;i++)
      {
         node
         nodetemp(DbmChild[i],dipenuhi1,numAP,lev+1,ApDbm,A,
         AffDynChild[i]);
         child[i]=nodetemp;
      }
   }
}
```

4.7 Menentukan Transisi

Di dalam fungsi *Transition*, hal pertama yang dilakukan adalah penyimpanan *node leaf* dari pi0 *partition tree*, kemudian dilanjutkan dengan penentuan transisi. Kedua proses ini dilakukan tanpa fungsi rekursif, yaitu dengan memanfaatkan variabel array *alamat*, array *anakKe*, dan *level*. Dimensi dari variabel *alamat* dan *anakKe* adalah sebanyak level dari *pi0 partition tree* yaitu *nAP* + *dimA* + 1 seperti yang telah dijelaskan sebelumnya. Kode program dari fungsi ini adalah sebagai berikut

```
void AbstractionTree::Transition()
{
   node **alamat=new node*[numAP+A.dim+1];
   alamat[0]=&pi0PartTree.root;
   alamat[1]=&pi0PartTree.root.child[0];
   int *anakKe=new int[numAP+A.dim+1],level=1;
   numpi0PartTreeLeaf=0;
   anakKe[0]=anakKe[1]=1;

   while(level>0)
   {
      if(alamat[level]->numChild==0)
      {
         alamat[level]->state=++numpi0PartTreeLeaf;
         pi0PartTreeLeaf.pushBack(*alamat[level]);
```

```
      while(anakKe[level]==alamat[level-1]->numChild)
      {
         level--;
         if(level==0)
            break;
      }
      if(level!=0)
      {
         alamat[level]=&alamat[level-1]-
        >child[anakKe[level]];
         anakKe[level]++;
      }
   }
   else
   {
      level++;
      anakKe[level]=1;
      alamat[level]=&alamat[level-1]->child[0];
   }
}

adj=new bool*[numpi0PartTreeLeaf];
for(int i=0;i<numpi0PartTreeLeaf;i++)
   adj[i]=new bool[numpi0PartTreeLeaf];
for(int i=0;i<numpi0PartTreeLeaf;i++)
   for(int j=0;j<numpi0PartTreeLeaf;j++)
      adj[i][j]=false;
ListEl *pEl=pi0PartTreeLeaf.first;
for(int i=0;i<numpi0PartTreeLeaf;i++)
{
   DBM img=pEl->elem.d.image(pEl->elem.ad);
   alamat[1]=&pi0PartTree.root.child[0];
   level=anakKe[1]=1;
   while(level>0)
   {
      DBM intersect=img-alamat[level]->d;
      if(!intersect.isEmpty())
      {
         if(alamat[level]->numChild==0)
         {
            adj[alamat[level]->state-1][i]=true;
            while(anakKe[level]==alamat[level-1]-
            >numChild)
            {
               level--;
               if(level==0)
                  break;
            }
            if(level!=0)
            {
               alamat[level]=&alamat[level-1]-
                >child[anakKe[level]];
```

```
                anakKe[level]++;
            }
        }
        else
        {
            level++;
            anakKe[level]=1;
            alamat[level]=&alamat[level-1]->child[0];
        }
    }
    else
    {
        while(anakKe[level]==alamat[level-1]->numChild)
        {
            level--;
            if(level==0)
                break;
        }
        if(level!=0)
        {
            alamat[level]=&alamat[level-1]-
            >child[anakKe[level]];
            anakKe[level]++;
        }
    }
  }
  pEl=pEl->next;
 }
}
```

Proses menentukan transisi adalah sebagai berikut. Pertama, untuk setiap *node leaf*, dihitung image dari DBM terhadap *affine dynamics* nya. Kemudian dicari *node leaf* mana saja yang beririsan dengan image tersebut dengan menggunakan *backtracking*, tanpa menggunakan fungsi rekursif.

Algoritma *backtracking* yang digunakan adalah sebagai berikut. Proses dimulai dari *root*, sebagai *node* yang ditunjuk. Hitung irisan dari DBM yang direpresentasikan oleh *node* yang ditunjuk dengan *image*. Kemudian, dicek apakah irisannya kosong:

1. Jika irisannya kosong, maka bisa dipastikan semua DBM yang direpresentasikan oleh anak dari *node* yang ditunjuk juga kosong. Sehingga *node* yang ditunjuk berganti ke anak yang berikutnya (bergeser ke kanan). Jika *node* yang ditunjuk adalah anak yang terakhir, maka *node* yang ditunjuk bergerak ke atas, dan coba untuk berganti ke anak yang berikutnya (bergeser ke kanan). Hal ini dilakukan terus menerus, hingga

diperoleh *node* baru yang belum pernah ditunjuk. Jika proses ini menunjuk node *root*, maka proses berakhir. Jika *node* ini menunjuk *node* baru, maka proses dilanjutkan ke langkah berikutnya.

2. Jika irisannya tidak kosong, maka *node* yang ditunjuk berikutnya adalah anak pertama (paling kiri) dari *node* sebelumnya. Jika *node* yang ditunjuk adalah *leaf*, maka terdapat transisi ke *node* yang ditunjuk. Pada kasus ini, langkah berikutnya adalah mencari anak setelahnya (proses ini bisa berulang, lihat paragraf sebelumnya).

4.8 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* pada percobaaan:
   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 $0 \leq x_1 - x_2 < 3$.

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

D. Formula CTL setiap percobaan sama yaitu:

   AX(AX(a))

   EG(AF(!a))

E. Formula LTL setiap percobaan sama yaitu:

   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 4.1.

Tabel 4.1 Beberapa attribut sistem transisi abstrak dan nilai verifikasi hasil uji coba

Uji coba ke-	**Ukuran Sistem Transisi**	**State awal**		**Nilai verifikasi formula**	
		VeriSiMPL 2.0	**Tesis**	**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-state*nya saja. Untuk *running time* uji coba dan rasio (*running time* VeriSiMPL 2.0 : *running time* program Tesis) bisa dilihat pada Tabel 4.2.

Tabel 4.2 *Running time* uji coba

Percobaan ke-	***Running Time* (detik)**		**Rasio**
	VeriSiMPL 2.0	**Tesis**	
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

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Proses penentuan transisi dapat diimplementasikan dengan menggunakan perulangan dan variable *Alamat, anakKe* dan *level*. Variabel *Alamat* bertipe array dari *pointer* ke *node*, dengan jumlah anggota sebanyak jumlah level dari *tree*. Variabel *anakKe* bertipe array dari integer, dengan jumlah anggota sebanyak jumlah level dari *tree*. Variabel *level* bertipe *integer*.

Dari beberapa hasil uji coba yang telah dilakukan, perangkat lunak yang dihasilkan pada Tesis 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.

5.2 Saran

Pada penelitian ini, pembentukan *partition tree* menggunakan fungsi rekursif. Pada penelitian selanjutnya, dapat dikonstruksi pembentukan *partition tree* tanpa menggunakan fungsi rekursif agar memori yang dibutuhkan lebih efisien.

DAFTAR PUSTAKA

D. Adzkiya, B. De Schutter, A. Abate, "Finite Abstractions of Max-Plus-Linear Systems", IEEE Transactions on Automatic Control, vol. 58, no. 12, pp. 3039-3053, December., 2013.

Brackley, C.A., Broomhead, D.S., Romano, M.C., Thiel, M.: A max-plus model of ribosome dynamics during mRNA translation. Journal of Theoretical Biology 303(0), 128–140 (2012)

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)

D. Adzkiya, Y. Zhang, A. Abate, "VeriSiMPL 2: An Open-Source Software for the Verification of Max-Plus-Linear Systems", Discrete Event Dynamic Systems, 2015, manuscript.

D. Adzkiya, B. De Schutter, and A. Abate. Abstraction and verification of autonomous max-plus-linear systems.In Proc. 31st Amer. Control Conf. (ACC'12), pages 721–726, Montreal, CA, June 2012.

C. Baier and J.-P. Katoen, Principles of Model Checking. Cambridge, MA: The MIT Press, 2008.

Leenaerts, D., van Bokhoven, W.: Piecewise Linear Modeling and Analysis. Kluwer Academic Publishers, Boston (1998)

BIOGRAFI PENULIS

Nama	:	Muhammadun
Tempat Lahir	:	Bangkalan
Tanggal Lahir	:	27 Oktober 1983
Jenis Kelamin	:	Laki-laki
Alamat	:	Jl. Trunojoyo VII/62E, Bangkalan
		Jawa Timur, Indonesia
HP	:	+62 857-4920-9633
E- Mail	:	muhammadun2002@gmail.com

Riwayat Pendidikan

1990-1996	SDN Pejagan IX Bangkalan, Indonesia
1996-1999	SLTPN 1, Bangkalan, Indonesia
1999-2002	SMUN 1, Bangkalan, Indonesia
2002-2009	Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
	Program Sarjana (S1), Matematika
2014-2017	Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
	Program Pascasarjana (S2), Matematika