TRANSFORMASI *CHANNEL SYSTEM* KE *LABELLED TRANSITION SYSTEM*

Siti Mutmainah[1] dan Reza Pulungan[2]
Jurusan Ilmu Komputer dan Elektronika
Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada
FMIPA Gedung Selatan, Sekip Unit III, Kotak Pos BLS. 21
55281 Yogyakarta, Indonesia
Telp. (+62 274) 546194, Faks. (+62 274) 546194
E-mail: [1]Siti.mutmainah@mail.ugm.ac.id

Abstract

A parallel system must be developed with extra precision to achieve a high level of dependability to produce error-free software.
Therefore, we need an appropriate and formal model of the system. The model is necessary for verification software in model checking. A model that is often used in model checking is Labelled Transition System (LTS). An LTS can be generated from Program Graphs (PG), which are representations of programs in any programming language. However, in modeling a parallel system we need to describe the communications(model of interactivity) between the Program Graphs. For this purpose, we need and use Channel Systems (CS).
Therefore, we often need to transform CS to LTS. The aim of this research is to developa transformation algorithm to generate an LTS from a given CS. Through the concept of parallelism, the algorithm is constructed by exploiting the so-called Structured Operational Semantic (SOS) of the CS.
The transformation algorithm is implemented by developing a prototype tool designed with 3 essential components:input, process and output. Input is a text file that contains a model of a parallel system in the form of a CS. The model will be processed by using thetransformation algorithm that generates an LTS as output text file.
Functionally, the algorithm can handle models of parallel systems written ina CS that consists of several PGs containing both looping and branching. Moreover, it has ability to satisfy various parallelism models such as interleaving, synchronous and asynchronous.

***Keywords**: Channel System (CS), Labelled Transition System (LTS), parallel, Program Graph (PG), statesemantic.*

A. Pendahuluan

Perangkat lunak yang bebas dari kesalahan adalah perangkat lunak yang dengan tepat mengikuti spesifikasinya (Sommerville, 2001: p.22). Oleh karena itu diperlukan spesifikasi sistem yang tepat dan formal untuk mendefinisikan sistem yang akan diimplementasikan. Keberadaan semantik formal sangat berguna untuk menghasilkan deskripsi yang tidak ambigu dari sebuah model sistem reaktif. Hal ini berguna untuk mengetahui tingkat *correctness* dari sistem tersebut. Correctness dapat dicapai dengan

teknik verifikasi. Dalam verifikasi ada dua metode yang sering digunakan, yaitu *theorem proving* dan *model checking*.

Model checking merupakan teknik efektif untuk menemukan kesalahan desain yang potensial terjadi. Teknik ini menggunakan metode formal untuk memverifikasi semua state yang terhingga banyaknya dalam sebuah sistem dan memverifikasi kebenaran dari spesifikasi sistem yang diekspresikan dalam logika temporal. Model checking bekerja secara atomatis dan relatif cepat. Jika terdapat error dalam desain, model checking akan memproduksi sebuah *counterexample* yang digunakan sebagai penunjuk sumber error (Baier dan Katoen, 2008: p.12).

Hampir semua tool model checking menggunakan Labelled Transition System (LTS) dalam pemrosesannya. Hal ini dikarenakan LTS merupakan bentuk universal dan mempunyai semantik yang jelas. LTS digunakan sebagai dasar untuk mendeskripsikan behavior proses-proses yang terjadi dalam sistem. LTS dapat dihasilkan melalui program graph (PG). Program graph adalah suatu formalisme yang dapat merepresentasikan semua program yang ditulis dalam bahasa pemrograman apapun. Namun karena yang akan dimodelkan adalah sistem reaktif, maka harus ada model komunikasi untuk menggambarkan interaktifitas antar proses. Oleh karena itu dibutuhkan sebuah *channel system*(CS) sebagai media ketika terjadi proses pengiriman (*sending*) atau penerimaan (*receiving*) nilai yang melibatkan variabel-variabel yang dimiliki oleh beberapa program graph. Bentuk LTS yang dihasilkan akan sangat bermanfaat dalam pengecekan model, sehingga penulis memandang perlu adanya sebuah tool otomatis untuk menghasilkan bentuk LTS dari channel system.

Dalam penelitian Tretmans (2008: p.1-38) dideskripsikan model dan bahasa yang digunakan, seperti LTS dan beberapa variantnya untuk memodelkan spesifikasi, implementasi dan testing. Penelitian ini juga secara formal mendefinisikan relasi implementasi *ioco* yang mengekspresikan apakah implementasi sudah sesuai dengan spesifikasinya atau belum. Sedangkan De Nicola dan Loreti (2007: p.133-146) pernah melakukan penelitian di mana LTS dijadikan sebagai model fundamental untuk

membuktikan kesesuaian *property* dari sistem yang berjalan secara *concurrent*. Dalam penelitiannya mereka mengenalkan MLTS (*Multiple Labelled Transition System*) sebagai generalisasi dari LTS yang memiliki fitur yang sangat penting ketika mempertimbangkan bahasa dan model untuk pemrograman network. Dijelaskan pula mengenai bagaimana MLTS dapat digunakan untuk mendeskripsikan semantik operasional dari calculus untuk mobilitas yang dikenal dengan *asynchronous π-calculus* (Boudol, 1992: p.15). Selain itu Uchitel et al. (2003: p.19-27) pernah menulis bahwa *state machine* seperti LTS secara umum digunakan untuk mendeskripsikan behavior dari sebuah sistem. Formalisasi tersebut diasumsikan menjadi deskripsi yang lengkap dari behavior sebuah sistem pada level abstraksi (Keller, 1976: p.371–384). Penelitian yang mereka lakukan dengan menggunakan PLTS (*Partial Labelled TransitionSystem*) untuk mengambil apa saja yang belum terdefinisikan dari behavior sebuah sistem. PLTS merupakan perluasan dari LTS yang secara tidak langsung memodelkan *action* yang kemungkinan tidak pernah terjadi dalam masing-masing state. Mereka menggunakan pendekatan Whittle dan Schumann (Whittle dan Schumann, 2000: p.314-323).

Dalam penelitian ini penulis bermaksud untuk memproduksi LTS dengan cara yang berbeda yakni dengan mentransformasi CS. CS digunakan untuk menggambarkan protokol komunikasi di mana channel itu sendiri memiliki buffer *first-in, first-out* yang berisi nilai atau *message*. Proses yang berkomunikasi dengan proses lain di dalam sistem dapat melalui channel tersebut.

B. Landasan Teori

Labelled Transition *System* (LTS)

Labelled Transition System (LTS) adalah model dasar untuk merepresentasikan reaktifitas, kongkurensi dan komunikasi sistem dan sering digunakan dalam ilmu komputer sebagai model untuk menggambarkan behavior dari sistem. Definisi menurut Baier dan Katoen (2008) bahwa sebuah Labelled Transition System adalah sebuah tuple (S, Act, →, I, AP, L), di mana:

- S adalah himpunan dari *state*.
- *Act* adalah himpunan dari *action*.
- S × Act × S yang merupakan sebuah relasi transisi.
- I adalah himpunan bagian S (I S) dan merupakan himpunan dari *state* inisial.
- AP adalah himpunan dari *atomic proposition*.
- L : S 2^{AP} adalah sebuah fungsi *labeling*.

Program *Graph*

Program Graph (PG) adalah bentuk representasi dari program yang menjalankan sebuah proses. Proses tersebut digambarkan dalam bentuk graph yang terdiri atas lokasi awal, lokasi akhir dan transisi yang menunjukkan adanya perpindahan dari lokasi awal ke lokasi akhir. Definisi menurut Baier dan Katoen (2008) bahwa secara formal definisi sebuah Program Graph adalah sebuah tuple (Loc, Act, Effect, →, Loc0, g0), di mana:

- Loc adalah set dari lokasi yang terbatas jumlahnya.
- Act adalah set dari action.
- Effect: Act × Eval(var) → Eval(var); misalnya jika α adalah x := x + y maka Effect(α, [x = 1, y = 7]) = [x = 8, y = 7]. Jadi, Effect di sini menggambarkan sebuah akibat yang terjadi terhadap perubahan nilai variabel jika action α dijalankan.
- → adalah himpunan bagian dari Loc × Cond(Var) × Act × Loc, yaitu relasi transisi kondisional.
- Loc0 adalah himpunan bagian dari Loc yang merupakan set dari inisial lokasi.
- g0 adalah elemen dari Cond(Var) yang merupakan inisial dari kondisi.

Channel System

Dalam sistem, sebuah proses akan berkomunikasi dengan proses lain dengan menggunakan channel. Menurut Baier dan Katoen (2008) channel system adalah sebuah cara untuk menggambarkan protokol komunikasi yang terjadi dalam sebuah sistem yang dapat ditulis dengan:

$$CS = [P_1 \,||\, P_2 \,||\, \boxtimes\boxtimes\boxtimes \,||\, P_n]$$

P_i adalah program *graph* pada sepasang (Var,Chan), di mana:

- Var adalah set dari variable.
- Chan adalah set dari channel yang memiliki kapasitas Cap(.) dan domain Dom(.). Jadi dapat ditulis dengan:

$$Pi = (Loci, Acti, Effecti, \; i, Loc0, g0)$$

Interleaving

Konsep interleaving merupakan bentuk konstruksi dari beberapa komponen di mana action dari beberapa komponen yang bersifat independen tersebut digabungkan (interleaved), sehingga dimungkinkan penggunaan satu atau lebih prosessor. Representasi interleaving dari sistem paralel dijadikan ide untuk membuat penjadwalan dari sebuah eksekusi yang berjalan secara kongkuren (Baier dan Katoen, 2008: p.36). Misalkan ada action independen α dan β yang merupakan pilihan non-deterministic yang dieksekusi dalam urutan sembarang, maka dapat ditulis dengan:

$$\text{Effect}(\alpha \,|||\, \beta, \eta) = \text{Effect}\,((\alpha; \beta) + (\beta; \alpha), \eta)$$

Operator titik koma menunjukkan eksekusi sekuensial, operator '+' menunjukkan pilihan non-deterministic dan operator ||| untuk eksekusi secara kongkuren dari aktifitas-aktifitas independen. Jadi,

$TS_1 \,|||\, TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ell_1 \times \ell_2, AP_1 \cup AP_2, L)$ di mana relasi transisi $\rightarrow$ didefinisikan dengan aturan seperti:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)} \; dan \; \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')}$$

dan fungsi label didefinisikan dengan L(s1, s2) = L(s1) L(s2). Maka dengan demikian, program graph PG1 (pada Var1) dan PG2 (pada Var2) tidak memiliki shared

variable (Var1 ∩ Var2 =). Jadi, jika menggunakan operator interleaving dapat ditulis TS(PG1)|||TS(PG2) yang menunjukkan behavior dari eksekusi simultan PG1 dan PG2.

Shared Variable

Shared variable adalah variabel yang dapat digunakan dan dirubah oleh semua proses. Jika terdapat action α dan β maka untuk mengatasi program paralel dengan shared variable, maka operator interleaving dapat didefinisikan pada level program graph. Interleaving program graph PG1 dan PG2 di lambangkan dengan PG1|||PG2, sehingga hasil interleaving dari dua program graph tersebut adalah TS(PG1|||PG2). Definisi interleaving Baier dan Katoen (2008: p.46) pada program graph secara formal adalah PGi = (Loci,Acti, Effecti, →0i, Loc0i, g0i). Program Graph PG1|||PG2 untuk variabel Var1 Var2 didefinisikan dengan:

$$PG_1|||PG_2 = (Loc_1 \times Loc_2, Act_1 \bigcup Act_2, Effect, \hookrightarrow, Loc0_1 \times Loc0_2, g0_1 \wedge g0_2)$$

di mana → didefiniskan dengan aturan seperti pada:

$$\frac{l_1 \overset{g:\alpha}{\rightarrow}_1 l_1'}{(l_1, l_2) \overset{g:\alpha}{\rightarrow} (l_1', l_2)} dan \frac{l_2 \overset{g:\alpha}{\rightarrow}_2 l_2'}{(l_1, l_2) \overset{g:\alpha}{\rightarrow} (l_1, l_2')}$$

Sedangkan fungsi Effect terhadap *action* dan evaluasi variabel dapat ditulis:

Effect(α, η) = Effecti(α, η) , jika α Acti

Program *graph* PG1 dan PG2 memiliki variabel Var1 ∩ Var2 yang di dalamnya terdapat *share variables* atau sering disebut ”global” variabel. Sedangkan variabel Var1 Var2 pada PG1 atau Var2 Var1 pada PG2 dinamakan variabel lokal.

Handshaking

Handshaking diartikan bahwa ada interaksi yang synchronous antara proses-proses yang berjalan secara kongkuren di mana proses bisa berinteraksi hanya jika kedua komponen sudah memiliki shake hands dalam waktu yang bersamaan di mana

pengirim pesan akan mengirimkan pesan jika si penerima pesan sudah siap untuk menerima pesan tersebut. Adapun rule untuk handshaking adalah:

• *Interleaving* untuk $\alpha \notin H$ adalah:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)}$$

$$\frac{s_2 \xrightarrow{\alpha}_2 s_2'}{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')}$$

• *Handshaking* untuk α H, yaitu:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \wedge s_2 \xrightarrow{\alpha}_2 s_2'}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2')}$$

Definisi *handshaking* (Baier dan Katoen, 2008: p.48-49) secara formal adalah $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$. Jika i = 1, 2 adalah transition system $TS_1 \|_H TS_2$ maka dapat didefinisikan oleh:

$$TS_1 \|_H TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

di mana L(S_1, S_2) = $L_1(S_1)$ $L_2(S_2)$ dan relasi transisi → didefiniskan seperti rule handshaking.

C. Desain dan Implementasi

Sistem ini dirancang dengan arsitektur 3 komponen utama, yaitu input, proses dan output. Komponen input berupa sebuah file teks yang memodelkan sistem dalam bentuk CS. Arsitektur ini dapat digambarkan seperti pada Gambar 1.

Gambar 1. Arsitektur sistem

File input akan diproses dengan menggunakan algoritma transformasi yang di dalamnya terdapat beberapa sub proses untuk menjalankan proses paralel dan komunikasi antar PG yaitu interleaving, synchronous dan asynchronous.

Format input

Sebelum menjalankan algoritma transformasi perlu adanya pengaturan format file input agar penulisannya dapat dikenali secara konsisten. Format file input tersebut adalah sebagai berikut:

```
Channel System <Nama CS>
CS = (PG1 || PG2 || .. || PG_n)
Var <Jumlah variabel global>
<Nama varibel><Tipe variabel><[Domain]><Nilai awal>
.
.
.
PG#<Nama Program Graph>
{
Var <Jumlah variabel lokal>
<Nama variabel><Tipe variabel><Domain><Nilai awal>
.
.
.
loc <Jumlah lokasi>
trans <Jumlah transisi>
<Lokasi awal><Lokasi akhir><[Guard]><[Action]>
.
.
.
}
```

Format output

Output dari sistem ini adalah berupa file LTS. Format penulisan output ini adalah sebagai berikut:

```
Elapsed time in minute : second : milisecond is
Biggest memory used :
Nama CS:
STATE AWAL:
JUMLAH STATE :
JUMLAH TRANSISI :
<state1><state2><guard><action>
.
.
```

.

Adapun Format penulisan untuk <state> adalah <lokasi_PG1...lokasi_PGn><nilai_variabel><isi_channel>). Nilai variabel dan isi channel didapatkan dari action yang dijalankan pada state awal dan berpindah ke state berikutnya yang melibatkan fungsi Eval untuk mengevaluasi nilai-nilai variabel dan isi channel yang ada pada state tersebut.

Algoritma transformasi

Transformasi dari CS ke LTS merupakan sebuah proses untuk mengetahui semantik dari CS yang terdiri atas beberapa program graph (PG) yang berjalan secara parallel. Pembentukan LTS dimulai dengan membaca file input, kemudian diproses dengan menggunakan algoritma transformasi sehingga menghasilkan file output berupa LTS. Algoritma transformasi yang diimplementasikan dapat dilihat pada Gambar 2.

```
1. baca_File_Input;
2. FOR (stateAwal : listStateAwal)
3. begin
4.     IF stateAwal.punya_Transisi
5.     begin
6.       FOR (calonTransisi : listCalonTransisi)
7.       begin
8.          IF guard.terpenuhi
9.          begin
10.             IF listChannel > 0
11.             begin
12.                IF kapasitas(channel) > 0
13.                   action_Asynchronous;
14.                ELSE
15.                action_Synchronous;
16.             end;
17.             action_Interleaving;
18.          end;
20.             buat_State_Akhir;
21.             simpan_Transisi;
22.             simpan_State;
23.          end;
24.             end;
25.       end;
26.  IF !list_transisi.sama
27.  begin
28.    insert(transisi)
29.  end;
30.  IF !list_state.sama
31.  begin
32.    insert(state)
33.  end;
34.  cetak_LTS;
```

Gambar 2. Algoritma transformasi CS ke LTS

Implementasi

Algoritma transformasi akan diimplementasikan menggunakan bahasa pemrograman Java di lingkungan Netbean IDE 7.0 yang dijalankan pada sistem operasi Windows 7, Processor i3-330M, RAM 4 GB.

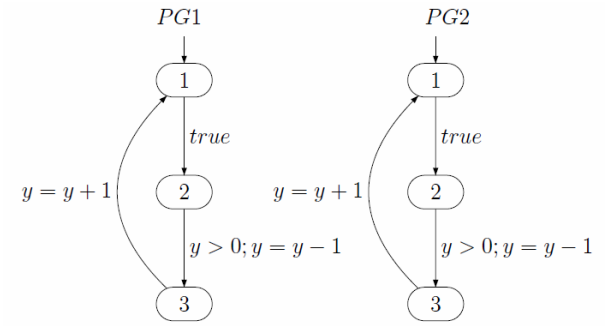
D. Pembahasan

Pengujian untuk proses paralel

Pengujian proses *interleaving*. Input teks yang digunakan untuk lakukan pengujian *interleaving* menggunakan algoritma *Mutual Exclusion* yaitu:

```
Channel System Mutex
Mutex = [ PG1 || PG2 ]
var 1
y INT 0..1 1
PG#PG1
{
loc 3
trans 3
1 2 [true]
2 3 [y>0] [y:=y-1]
3 1 [true] [y:=y+1]
}
PG#PG2
{
loc 3
trans 3
1 2 [true]
2 3 [y>0] [y:=y-1]
3 1 [true] [y:=y+1]
}
```

Dari model input tersebut untuk menggambarkan 2 proses yang berjalan secara paralel yaitu PG_1 dan PG_2 sebenarnya dapat dibuat dalam bentuk *graph* seperti pada Gambar 3.

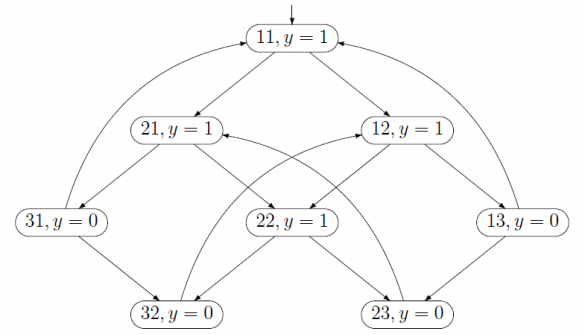


Gambar 3. PG_1 dan PG_2

Adapun LTS hasil dari proses *interleaving* di atas adalah sebagai berikut:

```
Elapsed time in minute:second:miliseconds is 00:00:500
Biggest Memory Used : 2123480 B
Nama CS : Mutex
INIT STATE : PG1_1 PG2_1
JUMLAH STATE : 8
JUMLAH TRANSISI : 14
(PG1_1 PG2_1, , y=1 )(PG1_2 PG2_1, , y=1 ) [true]
(PG2_1 PG1_1, , y=1 )(PG2_2 PG1_1, , y=1 ) [true]
(PG1_2 PG2_1, , y=1 )(PG1_3 PG2_1, , y=0 ) [y>0] [y:=y-1]
(PG2_1 PG1_2, , y=1 )(PG2_2 PG1_2, , y=1 ) [true]
(PG2_2 PG1_1, , y=1 )(PG2_3 PG1_1, , y=0 ) [y>0] [y:=y-1]
(PG1_1 PG2_2, , y=1 )(PG1_2 PG2_2, , y=1 ) [true]
(PG1_3 PG2_1, , y=0 )(PG1_1 PG2_1, , y=1 ) [true] [y:=y+1]
(PG2_1 PG1_3, , y=0 )(PG2_2 PG1_3, , y=0 ) [true]
(PG2_2 PG1_2, , y=1 )(PG2_3 PG1_2, , y=0 ) [y>0] [y:=y-1]
(PG1_2 PG2_2, , y=1 )(PG1_3 PG2_2, , y=0 ) [y>0] [y:=y-1]
(PG2_3 PG1_1, , y=0 )(PG2_1 PG1_1, , y=1 ) [true] [y:=y+1]
(PG1_1 PG2_3, , y=0 )(PG1_2 PG2_3, , y=0 ) [true]
(PG1_3 PG2_2, , y=0 )(PG1_1 PG2_2, , y=1 ) [true] [y:=y+1]
(PG2_3 PG1_2, , y=0 )(PG2_1 PG1_2, , y=1 ) [true] [y:=y+1]
```

LTS yang dihasilkan sebenarnya dapat diilustrasikan dalam bentuk *graph* seperti pada Gambar 4. PG_1 dan PG_2 akan berjalan secara bergantian dari 1 lokasi ke lokasi berikutnya dan melakukan *action* yang ada pada saat itu. Dilihat dari hasilnya ketika 2 proses PG_1 dan PG_2 berjalan secara paralel dengan model seperti pada input, maka jumlah state yang dihasilkan adalah 8 dan transisi adalah 14. Jumlah tersebut tidak akan berubah meskipun dalam pengujian dilakukan perluasan domain variabel.



Gambar 4. Hasil interleaving PG_1 dan PG_2

Pengujian **proses *synchronous*.** Adapun input teks yang digunakan dalam pengujian ini adalah sebagai berikut.

```
Channel System myCS
myCS = [ PG1 || PG2 ]
var 2
c chan [0] null
x INT 0..2 1
PG#PG1
{
var 0
loc 2
trans 2
1 2 [true] [c!x]
2 1 [true] [x:=x+1]
}
PG#PG2
{
var 1
v INT 0..2 1
loc 2
trans 2
1 2 [true]
2 1 [true] [c?v]
}
```

Adapun hasil LTS yang terbentuk dari file input tersebut adalah sebagai berikut.

```
Elapsed time in minute:second:miliseconds is 00:00:125

Biggest Memory used : 2475592 B
Nama CS : myCS
INIT STATE : PG1_1 PG2_1
JUMLAH STATE : 9
JUMLAH TRANSISI : 12
(PG1_1PG2_1,PG2_v=1,x=1,c=n)(PG1_2PG2_1,PG2_v=1,x=1,c=n)[true][c!x]
```

```
(PG2_1PG1_1,PG2_v=1,x=1,c=n)(PG2_2 PG1_1,PG2_v=1,x=1,c=n)[true]
(PG1_2PG2_1,PG2_v=1,x=1,c=n)(PG1_1PG2_1,PG2_v=1,x=2,c=n)[true] [x:=x+1]
(PG2_1PG1_2,PG2_v=1,x=1,c=n)(PG2_2PG1_2,PG2_v=1,x=1,c=n)[true]
(PG2_2PG1_1,PG2_v=1,x=1,c=n)(PG2_1PG1_2,PG2_v=1,x=1,c=n)[][SYNC]
(PG1_1PG2_1,PG2_v=1,x=2,c=n)(PG1_2PG2_1,PG2_v=1,x=2,c=n)[true][c!x]
(PG2_1PG1_1,PG2_v=1,x=2,c=n)(PG2_2 PG1_1,PG2_v=1,x=2,c=n)[true]
(PG2_2 PG1_2,PG2_v=1,x=1,c=n)(PG2_1PG1_2,PG2_v=1,x=1,c=n)[true][c?v]
(PG1_2PG2_2,PG2_v=1,x=1,c=n)(PG1_1PG2_2,PG2_v=1,x=2,c=n)[true] [x:=x+1]
(PG2_1PG1_2,PG2_v=1,x=2,c=n)(PG2_2PG1_2,PG2_v=1,x=2,c=n)[true]
(PG2_2 PG1_1,PG2_v=1,x=2,c=n)(PG2_1 PG1_2,PG2_v=2,x=2,c=n)[][SYNC]
(PG2_2 PG1_2,PG2_v=1,x=2,c=n)(PG2_1PG1_2,PG2_v=1,x=2,c=n)[true][c?v]
```

Dari hasil pengujian dengan menggunakan model *synchronous* di atas, maka dapat diketahui bahwa terjadi proses *synchronous* sebanyak 2 kali yaitu pada transisi `(PG2_2PG1_1,PG2_v=1,x=1,c=n)` ke `(PG2_1PG1_2,PG2_v=1,x=1,c=n)` dan `(PG2_2 PG1_1,PG2_v=1,x=2,c=n)` ke `(PG2_1 PG1_2,PG2_v=2,x=2,c=n)`. Karena jumlah kapasitas *channel* yang digunakan adalah nol, maka nilai x yang dikirim oleh PG_1 diterima langsung oleh PG_2 dan diletakkan di variabel v. LTS yang dihasilkan akan berubah ketika dilakukan perluasan domain variabel.

Pengujian untuk proses asynchronous. Input teks yang digunakan dalam pengujian ini menggunakan 2 variabel global yaitu c bertipe channel dengan kapasitas 2 dan nilai awal null yang artinya kosong dan y bertipe integer dengan domain 0 sampai 1 dan nilai awal 1. Model tersebut adalah sebagai berikut.

```
Channel System myCS
myCS = [ PG1 || PG2 ]
var 2
c chan [2] null
y INT 0..1 1
PG#PG1
{
loc 3
trans 3
1 2 [y>0] [y:=y-1]
2 3 [true] c!y
3 1 [true] [y:=y+1]
}
PG#PG2
{
var 1
x INT 0..2 0
loc 3
trans 3
1 2 [y>0] [y:=y-1]
2 3 [true] c?x
3 1 [true] [y:=y+1]
}
```

Adapun hasil LTS dari file input tersebut adalah sebagai berikut.

```
Elapsed time in minute:second:miliseconds is 00:00:486
Biggest Memory used : 2731568 B
Nama CS : myCS
INIT STATE : PG1_1 PG2_1
JUMLAH STATE : 13
```

```
JUMLAH TRANSISI : 15
(PG1_1PG2_1,PG2_x=0,y=1,c=nn)(PG1_2PG2_1,PG2_x=0,y=0,c=nn)[y>0][y:=y-1]
(PG2_1PG1_1,PG2_x=0,y=1,c=nn)(PG2_2PG1_1,PG2_x=0,y=0,c=nn)[y>0][y:=y-1]
(PG1_2PG2_1,PG2_x=0,y=0,c=nn)(PG1_3PG2_1,PG2_x=0,y=0,c=n0)[true] [c!y]
(PG2_2PG1_1,PG2_x=0,y=0,c=nn)(PG2_3PG1_1,PG2_x=0,y=0,c=nn)[true] [c?x]
(PG1_3PG2_1,PG2_x=0,y=0,c=n0)(PG1_1PG2_1,PG2_x=0,y=1,c=n0)[true] [y:=y+1]
(PG2_3PG1_1,PG2_x=0,y=0,c=nn)(PG2_1PG1_1,PG2_x=0,y=1,c=nn)[true] [y:=y+1]
(PG1_1PG2_1,PG2_x=0,y=1,c=n0)(PG1_2PG2_1,PG2_x=0,y=0,c=n0)[y>0][y:=y-1]
(PG2_1PG1_1,PG2_x=0,y=1,c=n0)(PG2_2PG1_1,PG2_x=0,y=0,c=n0)[y>0][y:=y-1]
(PG1_2PG2_1,PG2_x=0,y=0,c=n0)(PG1_3PG2_1,PG2_x=0,y=0,c=00)[true] [c!y]
(PG2_2PG1_1,PG2_x=0,y=0,c=n0)(PG2_3PG1_1,PG2_x=0,y=0,c=nn)[true] [c?x]
(PG1_3PG2_1,PG2_x=0,y=0,c=00)(PG1_1PG2_1,PG2_x=0,y=1,c=00)[true] [y:=y+1]
(PG1_1PG2_1,PG2_x=0,y=1,c=00)(PG1_2PG2_1,PG2_x=0,y=0,c=00)[y>0][y:=y-1]
(PG2_1PG1_1,PG2_x=0,y=1,c=00)(PG2_2PG1_1,PG2_x=0,y=0,c=00)[y>0][y:=y-1]
(PG2_2PG1_1,PG2_x=0,y=0,c=00)(PG2_3PG1_1,PG2_x=0,y=0,c=n0)[true] [c?x]
(PG2_3PG1_1,PG2_x=0,y=0,c=n0)(PG2_1PG1_1,PG2_x=0,y=1,c=n0)[true] [y:=y+1]
```

Dari LTS yang dihasilkan diketahui bahwa proses pengiriman terjadi sebanyak 2 kali yaitu pada transisi `(PG1_2PG2_1,PG2_x=0,y=0,c=nn)(PG1_3PG2_1,PG2_x=0,y=0,c=n0)[true] [c!y]` dan `(PG1_2PG2_1,PG2_x=0,y=0,c=n0)(PG1_3PG2_1,PG2_x=0,y=0,c=00)[true] [c!y]`. Pada saat transisi `(PG1_2PG2_1,PG2_x=0,y=0,c=nn)` ke `(PG1_3PG2_1,PG2_x=0,y=0,c=n0)` terlihat bahwa pengiriman nilai `y=0` dilakukan dengan menggunakan *channel* `c` sehingga `c` yang semula berisi `nn` (kosong) menjadi `n0` (buffer pertama berisi nol). Begitu pula pada transisi `(PG1_2PG2_1,PG2_x=0,y=0,c=n0)` ke `(PG1_3PG2_1,PG2_x=0,y=0,c=00)`di mana nilai c yang semula berisi `n0` menjadi `00`, karena ada pengiriman nilai `y=0`. Sedangkan proses penerimaan nilai terjadi sebanyak 3 kali yaitu pada transisi `(PG2_2PG1_1,PG2_x=0,y=0,c=nn)`

Ke `(PG2_3PG1_1,PG2_x=0,y=0,c=nn,` `(PG2_2PG1_1,PG2_x=0,y=0,c=n0)` ke `(PG2_3PG1_1,PG2_x=0,y=0,c=nn)` dan `(PG2_2PG1_1,PG2_x=0,y=0,c=00)` ke `(PG2_3PG1_1,PG2_x=0,y=0,c=n0).` Pada transisi `(PG2_2PG1_1,PG2_x=0,y=0,c=nn)`ke `(PG2_3PG1_1,PG2_x=0,y=0,c=nn)` terlihat bahwa nilai c pada state awal dan state akhir tidak berubah, karena channel c kosong, jadi tidak ada nilai yang dapat diterima oleh state akhir. Pada transisi `(PG2_2PG1_1,PG2_x=0,y=0,c=n0)` ke `(PG2_3PG1_1,PG2_x=0,y=0,c=nn)` nilai `c` berubah dari `n0` menjadi `nn` karena ada penerimaan `0` dan kemudian diletakkan di variabel `x` sehingga semua buffer di `c` menjadi kosong. Sedangkan pada transisi `(PG2_2PG1_1,PG2_x=0,y=0,c=00)`ke `(PG2_3PG1_1,PG2_x=0,y=0,c=n0)`nilai `c` berubah dari `00` menajadi `n0` karena ada penerimaan nilai `0` sehingga buffer yang ke-2 pada channel `c` menjadi kosong.

Pengujian dengan beberapa bentuk PG

Bentuk **PG berulang.** Hasil pengujian ini sudah terlihat pada pengujian sebelumnya yaitu pada proses *interleaving, synchronous* dan *asynchronous*, karena contoh file input yang diujikan adalah bentuk PG yang berulang.

Bentuk PG bercabang. Bentuk input dengan PG bercabang yang diujicobakan adalah sebagai berikut.

```
Channel System CS1
CS1 = [PG1 || PG2]
var 2
x INT 0..2 0
v INT 0..2 0
PG#PG1
{
loc 2
trans 2
1 2 [true]
1 3 [true] [x:=x+1]
}
PG#PG2
{
loc 2
trans 2
1 2 [true] [x:=x+v]
1 3 [true]
}
```

Model tersebut terdiri atas 2 buah PG yang memiliki bentuk bercabang di mana masing-masing PG terdapat 2 transisi yaitu dari lokasi 1 ke lokasi 2 dan dari lokasi 1 ke lokasi 3. Model tersebut menggunakan 2 buah variabel global yaitu x dan v yang bertipe integer dengan domain 0 sampai 2 dan nilai awal 0. Ketika file input teks dengan model tersebut dijalankan maka akan menghasilkan file LTS sebagai berikut:

```
Elapsed time in minute:second:miliseconds is 00:00:78
Biggest Memory Used : 1547496 B
Nama CS : CS1
INIT STATE : PG1_1 PG2_1
JUMLAH STATE : 9
JUMLAH TRANSISI : 12
(PG1_1 PG2_1, , x=0 v=0 )(PG1_2 PG2_1, , x=0 v=0 ) [true]
(PG1_1 PG2_1, , x=0 v=0 )(PG1_3 PG2_1, , x=1 v=0 ) [true] [x:=x+1]
(PG2_1 PG1_1, , x=0 v=0 )(PG2_2 PG1_1, , x=0 v=0 ) [true] [x:=x+v]
(PG2_1 PG1_1, , x=0 v=0 )(PG2_3 PG1_1, , x=0 v=0 ) [true]
(PG2_1 PG1_2, , x=0 v=0 )(PG2_2 PG1_2, , x=0 v=0 ) [true] [x:=x+v]
(PG2_1 PG1_2, , x=0 v=0 )(PG2_3 PG1_2, , x=0 v=0 ) [true]
(PG2_1 PG1_3, , x=1 v=0 )(PG2_2 PG1_3, , x=1 v=0 ) [true] [x:=x+v]
(PG2_1 PG1_3, , x=1 v=0 )(PG2_3 PG1_3, , x=1 v=0 ) [true]
(PG1_1 PG2_2, , x=0 v=0 )(PG1_2 PG2_2, , x=0 v=0 ) [true]
(PG1_1 PG2_2, , x=0 v=0 )(PG1_3 PG2_2, , x=1 v=0 ) [true] [x:=x+1]
(PG1_1 PG2_3, , x=0 v=0 )(PG1_2 PG2_3, , x=0 v=0 ) [true]
(PG1_1 PG2_3, , x=0 v=0 )(PG1_3 PG2_3, , x=1 v=0 ) [true] [x:=x+1]
```

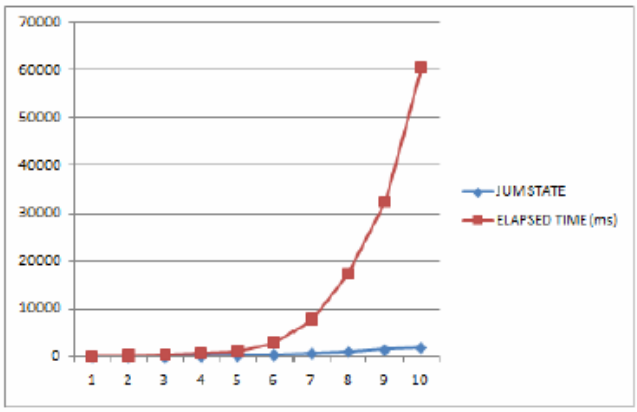
Dari LTS yang dihasilkan didapat jumlah state sebanyak 9 dan transisi sebanyak 12. Jumlah state dan transisi akan cenderung naik jika domain variabel diperluas.

Pengujian Performace

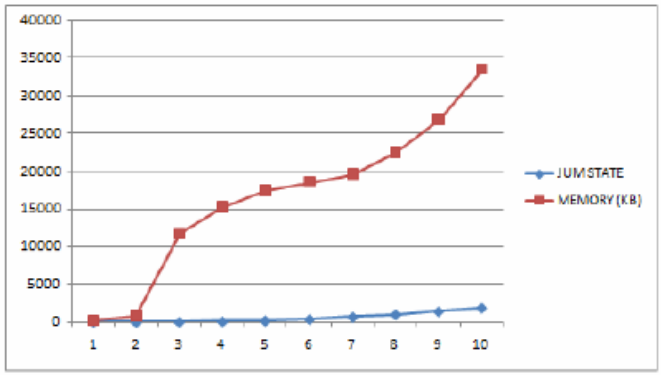
Input yang digunakan adalah sebuah model CS sebagai berikut.

```
Channel System CS1
CS1 = [PG1 || PG2]
var 4
c chan [2] null
x INT 0..1 1
z INT 0..1 1
y INT 0..1 1

PG#PG1
{
loc 3
trans 3
1 2 [true] [x:=x+z]
2 3 [x>y] [y:=y+1]
3 1 [true] c!y
}

PG#PG2
{
var 1
v INT 0..1 0
loc 3
trans 3
1 2 [true] [x:=x+z]
2 3 [x>y] [y:=y+1]
3 1 [true] c?v
}
```

Pengujian ini dilakukan dengan mengukur elapsed time dan memori yang terpakai pada saat menghasilkan state dalam jumlah tertentu. Adapun hasil pengamatan elapsed time terhadap jumlah state dapat dilihat pada Gambar 5. Sedangkan hasil pengamatan memory terhadap jumlah state dapat dilihat pada Gambar 6.



Gambar 5. Grafik hasil pengamatan *elapsed time* terhadap jumlah *state*



Gambar 6. Grafik hasil pengamatan memory terhadap jumlah state

Pada pengujian ini dilakukan perluasan domain variabel secara bertahap untuk memperoleh kenaikan elpased time dan memory. Domain variabel dimulai dari 0..1 sampai 0..10 di mana semua domain variabel dinaikan +1 untuk setiap tahap. Dari hasil pengujian terlihat bahwa elapsed time dan memory cenderung mengalami kenaikan seiring dengan naiknya jumlah state yang dihasilkan. Elapsed time naik secara kuadratik sedangkan kenaikan memory berjalan secara linier. Namun kenaikan tersebut sangat tergantung pada model sistem sebagai file inputnya.

D. Kesimpulan dan Saran

Kesimpulan

1. Telah dibuat sebuah algoritma transformasi yang memiliki kemampuan dalam memodelkan sistem paralel yang ditulis dalam bentuk *Channel System* (CS) untuk menghasilkan *Labelled Transition System* (LTS).
2. Pertumbuhan jumlah *state* dan transisi dapat dilakukan dengan memperluas domain variabel yang biasanya berdampak pada kenaikan *elapsed time* dan memory. Akan tetapi, memperluas domain variabel tidak selalu berdampak pada kenaikan jumlah *state* dan transisi, karena sangat bergantung pada bentuk PG yang ada di dalam CS. Hal ini terjadi pada model algoritma *Mutual Exclusion* (MUTEX). Berapapun luasnya domain variabel yang digunakan, jumlah *state* yang dihasilkan adalah 8 dan transisi yang terbentuk adalah 14. Sedangkan jumlah memory yang terpakai

cukup stabil, yaitu berkisar antara 2.475.896 Byte sampai 2.475.808 Byte dan elapsed time berkisar antara 123 milidetik sampai dengan 135 milidetik.

Saran

1. Penelitian selanjutnya dapat menggunakan file LTS hasil transformasi dari *channel system* ini sebagai input untuk proses model *checking* dari sebuah sistem.
2. Penelitian selanjutnya mungkin dapat dilakukan proses transformasi dengan *action* untuk memanipulasi string.
3. Penelitian selanjutnya dapat dilakukan untuk proses pengujian terbalik, misalnya transformasi dari LTS ke CS.

Daftar Pustaka

Baier, C. dan Katoen, J.-P. 2008. *Principles of Model-Checking*. MIT Press, Cambridge.

Boudol, G. 1992. Asynchrony and the π-calculus. *Rapport de Recherche 1702*. INRIA Sophia-Antipolis. URL http://www.inria.fr/RRRT/RR-1702.html.

De Nicola, R. dan Loreti,M. 2007 Multi Labelled Transition Systems: A Semantic Framework for Nominal Calculi. *Electr. Notes Theor. Comput. Sci.*. Vol. 169, Hal 133-146.

Keller, R. 1976. Formal verification of parallel programs. *Communications of the ACM*. No. 7, Vol 19, Hal. 371-384.

Sommerville, I. 2001. Software Engineering. Edisi 6. Yuhilza Hanum. Erlangga, Yogyakarta.

Tretmans, J. 2008. *Model based testing with labelled transition systems. Formal methods and testing*. Hierons, R. M. dan Bowen, J. P. dan Harman,M. Springer Verlag, Berlin.

Whittle, J. dan J. Schumann. 2000. Generating Statechart Designs from Scenarios. Pada *22nd International Conference on Software Engineering (ICSE'00)*. Limerick, Ireland.

Uchitel, S. dan Kramer, J. danMagee, J. 2003. Behaviour model elaboration using partial labelled transition systems. Pada *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering Series = ESEC/FSE-11*. Helsinki, Finland.