Survey Metode Formal dalam Verifikasi dan Validasi

(Study of Formal Methods in Verification and Validation)

Munirah M.[1], Aslan Alwi[2], MHD. Reza M.I. Pulungan[3]

[1,2] Dosen Program Studi Teknik Informatika, Universitas Muhammadiyah Ponorogo
[3] Dosen Program Studi Ilmu Komputer, Universitas Gadjah Mada Yogyakarta
[1]munirah.mt@gmail.com

Abstrak– **Paper ini ditujukan untuk mensurvey metode formal yang digunakan orang baik itu dalam usaha mengembangkan software & harware, atau dalam usaha melakukan verifikasi dan validasi. Beberapa metode formal yang hendak ditinjau adalah CPN, PFSA, Actor Model, Rebeca, LTL, ACTL, CTL, UML, Hybrid Automata dan ASM. Pada setiap tinjauan, dijelaskan motivasi dan gagasan dasar metode formal bersangkutan secara intuitif dan implementasi-implementasinya yang merujuk pada berbagai paper-paper penelitian yang telah dilakukan orang menyangkut metode formal bersangkutan.**

Kata Kunci– Metode Formal, Verifikasi dan Validasi, Survey

*Abstract***– *This paper is intended to survey the formal methods that people use both in an effort to develop software and harware, or in an attempt to perform verification and validation. Some formal methods are about to be reviewed CPN, PFSA, Actor Model, Rebeca, LTL, ACTL, CTL, UML, Hybrid Automata and ASM. At each review, described the motivation and the basic idea of formal methods in question intuitively and implementation-implementation that refers to a variety of research papers that have been made concerning the formal methods in question.***

Keywords– Formal Methods, Verification and Validation, Study

I. PENDAHULUAN

Metode formal secara keseluruhan dalam pengertian teoritis adalah sebuah struktur bentuk matematika simbolik atau grafik, dan merupakan permodelan dari sistem-sistem nyata yang ada. Tujuan metode formal untuk melakukan otomatisasi koding yang mempercepat proses produksi dengan kualitas pengembangan yang terjaga, serta untuk melakukan verifikasi baik itu secara manual atau secara otomatis dengan mengembangkan perangkat lunak yang dapat melakukan verifikasi terhadap artifak-artifak pengembangan.

Jika dilihat dari letak posisi metode formal dalam pengembangan software atau hardware, dia terletak pada fase analisis dan fase design dan berakhir pada fase koding. Posisi ini dapat dilihat secara intuitif pada tipe pengembangan *waterfall* atau juga tipe pengembangan iteratif (*water-scrum-fall atau agile*).

Metode formal secara konkrit sebagai sebuah sistem sintaks pengembangan yang praktis baik secara grafik atau simbolik, adalah digunakan orang untuk membangun artifak-artifakdesign. Metode formal secara abstrak sebagai sebuah sistem semantik, dalam artian metode formal sebagai sebuah bahasa formal memiliki skema penilaian logika (*true-false*), adalah digunakan orang untuk melakukan verifikasi terhadap artifak-artifak design dan naskah kode yang dihasilkan.

Untuk melakukan verifikasi dengan metode formal, terlebih dahulu spesifikasi yang terdiri dari himpunan *requirement*, dimana himpunan *requirement* diperoleh dari wawancara terhadap user customer atau dari kebutuhan awal sistem sendiri, adalah terlebih dahulu dirumuskan atau ditulis secara *well-formed* (sesuai *grammar* metode formal) kedalam proposisi-proposisi atau formula-formula metode formal. Selanjutnya verifikasi dilakukan berdasarkan metode formal adalah sebuah penilaian terhadap artifak-artifak pengembangan dan naskah kode yang dihasilkan, dilakukan secara deduktif berdasarkan metode formalnya. Deduksi itu nantinya menghasilkan analisa-analisa yang menilai bahwa apakah

sistem yang sedang dikembangkan (*software* atau *hardware*) adalah memenuhi spesifikasi formal yang telah dirumuskan. Manakala sistem telah memenuhi seluruh spesifikasi formal maka sistem dinyatakan terverifikasi dengan baik.Metode-metode formal yang hendak ditinjau pada makalah ini adalah cpn, fsa, *hidden markov*, *actor model*, rebeca, LTL, ACTL, CTL, UML.

Sejumlah metode formal juga digunakan untuk mengkonstruksikan secara langsung design atau koding secara konkrit dengan cara melakukan deduksi terhadap spesifikasi formal yang ada, terus menerus sehingga diperoleh design konkrit atau kode konkrit dari sistem. Metode-metode formal yang melakukan deduksi seperti ini misalnya abstract state machine (ASM), B-method, Event-B, *Refinement calculus*, dan sebagainya.Masing-masing metode formal akan dibahas dalam paper ini.

Manfaat yang dari hasil kajian ini adalah pengguna nantinya dapat memilih metode formal yang bisa digunakan sesuai dengan kebutuhannya.

II. METODE

Metode yang digunakan dalam melakukan kajian ini adalah studi literatur. Metode ini dilakukan dengan studi kepustakaan melalui membaca buku-buku maupun artikel-artikel ilmiah terkait yang dapat mendukung penulisan paper ini.

III. HASIL DAN PEMBAHASAN

1. *Actor Model*

Actor model adalah sebuah metode formal yang digunakan membangun sebuah landasan teoritik bagi sistem komputasi yang terdistribusi.Sebuah metode formal yaitu sebagai sebuah perumusan formal padamana sistem komputasi terdistribusi dapat dibangun secara sintatik, dengan membangun bahasa formal yaitu berupa bahasa yang ditulis dalam bentuk BNF untuk digunakan menulis kode-kode pemrograman yang membangun komputasi terdistribusi secara konkrit.

Pada sisi yang komplemen, juga berarti bahwa sebuah perumusan formal padamana sistem komputasi terdistribusi dapat diverifikasi secara semantik dengan membangun sebuah semantik formal yaitu berupa bahasa formal yang diberi nilai true-false pada formula-formulanya. Landasan teoritik yang formal bagi komputasi terdistribusi dalam model actor pertama kali dikemukakan oleh [1], dimana dikemukakan secara formal bahwa unit komputasi yang paling primitif pada komputasi terdistribusi adalah sebuah *actor*. Kemudian dikemukakan proposisi-proposisi dasar guna mendeskripsikan *actor* secara lengkap sebagai sebuah unit komputasi.

Secara sederhana, *actor* adalah sebuah satuan komputasi terkecil dalam sistem komputasi terdistribusi, dia memiliki memory sendiri yang bukan memory global untuk digunakan melakukan komputasi secara sendirian terlepas dari *actor-actor* lain atau lingkungan, dan juga memiliki buffer untuk menyusun semua pesan yang masuk atau yangakan keluar dan memprosesnya. Dalam dirinya terdapat sejumlah metode atau perangkat komputasi mandiri yang digunakan untuk mengolah pesan yang masuk atau mengirim pesan.

Metode ini juga bekerja dengan sistem pewaktuan (*clock* lokal) sendiri, terlepas dari sistem waktu global (*clock* global), yang menjadwal, membuat batasan *delay* dan *threshold* terhadap komputasinya sendiri berdasarkan sistem waktu tersebut. Secara umum, komputasi terdistribusi antar *actor*, atau komputasi yang dibangun oleh sejumlah acktor adalah proses bertukar pesan satu sama lain dan proses komputasi secara mandiri di masing-masing actor.

Pertanyaan bahwa apakah di dalam actor itu sendiri, yang pada awalnya dinyatakan sebagai komputasi sekuensial secara lokal, adalah dapat juga diterapkan konkurensi lokal (komputasi paralel) dalam artian bahwa selain seluruh actor melakukan komputasi konkuren, terdistribusi secara global bersama-sama, akan tetapi juga melakukan komputasi konkuren (paralel) dalam dirinya sendiri.Sehingga diharapkan sejumlah pesan yang masuk pada actor, actor dapat mengeksekusinya secara paralel.Dengan demikian memungkinkan skalabilitas pengembangan bagi pengembangan berbasis actor.

Masalah ini dijawab oleh [2] dengan mengemukakan solusi yaitu perlu dibuat sebuah protokol baru yang dapat mewadahi komunikasi antar actor dimana actor-actor itu sendiri bekerja secara paralel dalam dirinya untuk mengolah pesan-pesan yang masuk.Diusulkan pulah oleh [2] PAM (Parallel Actors Monitors), yang nantinya dapat mengatur protokoler pertukaran pesan secara paralel antar actor.Artinya bahwa setiap actor dimungkinkan untuk mengirim pesan

tidak saja secara sekuensial pada satu target actor, akan tetapi secara paralel pada actor lain. PAM yang diusulkan oleh [2] menjaga sifat *entanglement* proses antar proses paralel antar actor.Pada sisi yang lain, [3] mengusulkan gagasan tentang bagaimana cara mengimplementasikan komputasi konkuren oleh actor-actor dengan cara menerapkan memory sharing atau secara umum component sharing, akan tetapi diterapkan secara efisien dan dapat mengatasi *deadlock* (dua actor atau lebih saling menunggu untuk menggunakan sebuah *resource*) ataupun *data races* (dua atau lebih actor berlomba menggunakan *resource* pada saat yang sama, dan salah satunya ada yang berusaha menulis).

Diusulkan oleh [3] gagasan domain dan *synchronization views*, pada proses berbagi komponen ini, dimana sebuah domain adalah sebuah koleksi objek yang siap digunakan bersama oleh sejumlah actor, dan untuk setiap actor yang hendak menggunakan dan diberi izin untuk mengakses domain, diberi view, dan untuk setiap view yang diberikan kepada masing-masing actor yang hendak mengakses domain tersebut, terdapat sinkronisasi view yang menjamin tidak terjadinya proses deadlock dan data races.

2. Rebeca

Rebeca (*Reactive Object Language*) adalah sebah bahasa formal sebagaimana bahasa pemrograman lainnya seperti C, Java, Pascal, dan lain sebagainya, akan tetapi bangunan sintaks yang ada pada Rebeca ditujukan pada pembuatan kode program yang berjalan pada sistem komputasi konkuren, ini dapat dilihat pada manual rebeca 1.0.2 [4]. Rebeca menggunakan paradima komputasi berbasis Actor model. Di dalam sintaksnya, rebeca mengimplementasikan sebuah actor sebagai sebuah kelas. Didalam kelas ini dapat dideklarasikan semua actor (kelas lain) yang dapat dihubungi atau diajak berkomunikasi melalui *send-receive message*, semua actor ini dideklarasikan dalam kelas sebagai *knownrebecs*.

Juga kelas diinisialisasi dengan state awal, sebagai state awal komputasi di dalam rebeca. Dan dideklarasikan beberapa fungsi atau metode yang siap digunakan untuk mengolah pesan yang masuk atau juga mengirim pesan ke *knownrebecs*. Kotak pesan pada kelas dinyatakan sebagai sebuah array dimana seluruh pesan masuk disimpan pada array ini, dan metode-metode dapat mengaksesnya.

Sintaks rebeca dapat dilihat pada gramatika di bawah ini.

<rebeca statement> ::= <assignment> | <send message> | <conditional>
<assignment> ::= <identifier> "=" <expression> ";"
<send message> ::= <rebec name> "." (<argumen list>*)
<argumen list> ::= <expression> | <argumen list> "." <expression>
<conditional> ::= <if then statement> | <if then else statement>
<if then statement> ::= if "(" <boolean expression> ")" then <rebeca compound>
<if then else statement> ::= if "(" <boolean expression> ")" then <rebeca compound> else <rebeca compound>

Gagasan tentang bagaimana jika mengimplementasikan rebeca berdasarkan pewaktuan, yaitu dengan mengenalkan sintaks baru pada sintaks rebeca yang sudah ada sebelumnya diusulkan oleh [5]. Sintaks ini adalah sintaks waktu, sehingga menghasilkan bahasa formal timed rebeca. Permasalahan yang ditemukan oleh [5] adalah jika mengimplementasikan rebeca sebagai server layanan diatas infrastruktur jaringan yang menjadikan *delay* jaringan sebagai suatu faktor yang penting. Ini berakibat, komputasi yang terjadi dalam sebuah rebec (actor) tentulah harus memiliki *clock* lokal tersendiri yang senantiasa harus tersinkronisasi dengan *clock* jaringan (*clock* global). dimana sebuah rebec dapat menjadwal dan menjangka sebuah komputasi dalam dirinya, menjangka seberapa lama waktu pengiriman dianggap gagal, dan sebagainya.

Dengan mengusulkan *solusi timed rebeca*, akan membuka pemecahan struktur sintaks yang dapat digunakan untuk menjalankan sebuah rebec di atas sebuah infrastruktur jaringan dimana delay waktu adalah sebuah faktor kritis [5].

Rebeca digunakan untuk memodelkan serangan terhadap celah didalam jaringan komputasi terdistribusi yang asinkron [6]. Beberapa orang sebelumnya menggunakan metode formal lain untuk merumuskan serangan dalam bentuk formal sebagai sebuah proposisi atau formula, kemudian dengan memetakan seluruh celah dalam jaringan, sehingga membentuk sebuah jejak atau path

vulnerabilities dalam jaringan, mereka membangun kombinasi jejak yang mungkin mencapai target serangan.Kemudian serangkaian (proposisi atau formula) dideduksikan didalam seluruh kombinasi jejak yang mungkin yang mencapai target untuk mengetahui apakah serangan itu terverifikasi dengan baik di dalam jaringan.

Rebeca juga digunakan oleh [6] untuk merumuskan serangan tersebut, menyatakan jejak-jejak *vulnerabilitis*, kemudian mengkonstruksikan seluruh kombinasi serangan yang mungkin, dan mengujinya dengan menggunakan permodelan actor based yang memakai bahasa rebeca.

3. LTL (*Linier temporal logic*)

LTL (*Linier temporal logic*) adalah metode formal yang digunakan orang untuk melakukan verifikasi terhadap sebuah sistem berdasarkan spesifikasi formal yang ada. Cara kerja metode formal ini, untuk LTL, yaitu bahwa sistem terlebih dahulu dinyatakan dalam struktur kripke, yaitu sebuah FSA (*finite state automata*) tetapi dengan tambahan pemetaan, yang memetakan tiap-tiap state dengan proposisi-proposisi atomik. Proposisi-proposisi atomik adalah formula-formula dasar yang merepresentasikan secara formal apa yang dilakukan sistem disuatu state. Kemudian, seluruh jejak (*path*) atau sebuah *run* yangmungkin dibariskan satu persatu.Path merupakan barisan state yang dijalani oleh sebuah proses *run*, sebuah barisan linier dalam waktu yang tak bercabang.Kemudian setiap *requirement* dalam spesifikasi formal dinyatakan dalam logika temporal, menjadi sebuah proposisi atau formula logika temporal. Demikian juga sebelumnya semua proposisi atomik pada struktur kripke juga dinyatakan dalam bentuk logika temporal.

Selanjutnya satu persatu formula spesifikasi dideduksikan dalam logika temporal diatas setiap *path* atau *run*.Jika formula itu berlaku di dalam *path*, artinya bahwa terdapat state dalam *path* tersebut dimana deduksi formula adalah benar pada state tersebut. Jika formula spesifikasi itu berlaku, maka formula tersebut dikatakan terverifikasi. Demikian seterusnya dengan menguji seluruh formula spesifikasi diatas semua *path.* Jika seluruh formula terverifikasi, maka sistem terverifikasi dengan baik (*well formed*).

Akan tetapi terdapat permasalahan dalam menerapkan *model checking* pada LTL, yaitu dimana pada *model checking* ini, setiap jejak (*path*) state ke state yang terjadi dalam waktu sebagai sebuah hasil komputasi boleh jadi menimbullkan "*explotion state*" dalam memori, dikarenakan *path* yang dihasilkan adalah begitu panjang, atau rangkaian state ke state dalam waktu yang dihasilkan menjadi sangat panjang sehingga terjadi ledakan kebutuhan memori untuk menyimpan seluruh jejak state dalam sebuah larik di memori. [7] mengusulkan solusi untuk masalah ini dengan menerapkan metode hash pada larik yang terjadi dari komputasi. Dimana larik state ke state dalam waktu yang dihasilkan sebagai sebuah linier temporal, setiap statenya dipetakan ke dalam larik hash, sehingga diharapkan dapat mengatasi ledakan kebutuhan memory. Demikian karena sebuah fungsi hash memetakan state yang sama atau state yang ekivalen dalam satu titik saja atau dalam satu pemetaan ke sebuah string yang menyatakan kode atau hash dari state yang ekivalen tersebut.

Diusulkan sebuah template bagi konstruksi formula-formula spesifikasi dalam konteks logika temporal oleh [8]. Template tersebut terdiri dari daftar formula yang ditulis dalam proposisi atau formula logika temporal, dimana dengan template itu berharap bahwa seseorang manakala hendak menguji sebuah sistem dengan model checking, dia cukup menggunakan template tersebut untuk merumuskan formula-formula dalam spesifikas formal sistem tersebut dalam logika temporal yang ada pada template, selanjutya mengujinya satu persatu dalam paradigma LTL. Dalam papernya, [8] membuktikan kesahihan dari template itu dalam beberapa penurunan teorema.

Cara untuk melakukan reduksi terhadap ruang state bagi sebuah sistem diusulkan oleh [9], dimana padanya direpresentasikan sebagai sebuah state/event LTL.

State/Event LTL adalah sesuatu yang berbeda dengan LTL biasanya (states LTL), yaitu dimana jejak state (*path* atau *run* dari komputasi) pada State/Event LTL juga mengikutkan transisi antar state sebagai bagian dari *path* atau *run.* Jadi sebuah run dalam State/Event LTL adalah barisan dalam waktu state ---> transisi (atau aksi, atau event) ---> state berikut, silih berganti membentuk *run*.

Akan tetapi didalam state/event LTL, karena mengikutkan transisi juga sebagai *run*, memungkinkan terjadinya rantai yang *partial order*, bukan *totally order* secara keseluruhan, artinya mungkin disana terdapat pilihan aksi (pilihan transisi) untuk menuju state berikut yang

sama. [9] mengemukakan gagasan tentang bagaimana cara melakukan reduksi tehadap state/event LTL tersebut dengan mereduksi bagian-bagian yang partial order tersebut, jika memungkinkan (dalam hal ini terdapat ekivalensi) sehingga salah satu dari lintasan *run* dapat direduksi.

4. CTL (*Computation tree logic*)

Adapun CTL, hanya sedikit berbeda dengan LTL.CTL menggunakan FSA yang non-deterministik. Demikian juga perumusan kripke strukturnya. Sehingga kumpulan path yang diperoleh adalah berupa tree.Jadi kita memperoleh sekumpulan tree, padamana formula-formula spesifikasi hendak diuji keberlakuannya.

Dikemukakan oleh [10] tentang cara untuk menerjemahkan LTL, CTL, dan CTL* ke dalam μ-*calculus*, untuk lebih mempermudah kalkulasi dikarenakan penggunaan LTL, CTL yang berarti juga CTL* dapat memberikan ledakan jumlah state dalam *run* nya. [24] menawarkan sebuah algoritma yang dapat memperbaharui model kripke agar dapat memenuhi formula dalam CTL.

5. ACTL (*Action Computation Tree Logic*)

ACTL adalah sebuah metode yang serupa dengan CTL, yaitu sama merupakan logika pencabangan waktu (*branching time logic*). Dimana barisan yang diperoleh akibat suatu *run* yang dijalankan pada *finite state automata* menghasilkan sebuah *tree* yang digunakan untuk menalar keabsahan proposisi-proposisi di atasnya.

Akan tetapi berbeda dengan CTL, yang melandaskan konstruksi *tree* di atas sebuah model kripke, ACTL melandaskan konstruksi *tree* nya di atas sebuah LTS (*labelled transition system*). Dimana pada CTL, setiap node dari *tree* adalah state, maka setiap *node* dari *tree* pada ACTL transisi yang telah diberi label sebuah *action* [11].

Juga pada paper berikutnya [12], menunjukkan cara bagaimana menggunakan ACTL untuk melakukan *model checking* pada sistem transport perkerataapian. Pada paper tersebut dimodelkan lintasan kereta api dengan menggunakan *finite state automata*, dan merumuskan transisi-transisinya untuk kendaraan berupa kereta dan mobil yang melintasi lintasan kereta. Permodelan ini merupakan permodelan konkuren dengan melihat bahwa kereta dan mobil dapat memiliki keputusan sendiri sehingga berarti komputasi sendiri yang harus sinkron pada lintasan kereta agar tak terjadi tabrakan kendaraan pada lintasan kereta api tersebut.

Pada permodelan tersebut, [12] menyatakan tiga macam formula yang nantinya hendak diverifikasi dalam *tree* dari pada komputasi yang berjalan, yaitu formula untuk state, formula untuk transisi (*action*) dan formula untuk jejak (*path*), keseluruhan formula dirumuskan dalam operator-operator logika temporal.

Selanjutnya proses *model checking* dilakukan dengan membangun *binary tree* sebagai pohon keputusan untuk memeriksa setiap proposisi dalam *tree* dari pada komputasi konkuren yang berjalan (*run*).

6. *Hybrid Automata*

Hybrid Automata adalah automata yang menggabungkan permodelan sistem kontinu dan sistem diskrit.State dari hybrid automata adalah sebuah sistem kontinu, atau sistem dengan waktu kontinu, dinyatakan oleh sebuah atau sejumlah persamaan *differensial* bersama dengan himpunan batasannya (*boundary*).

Ditulis dalam [13] bahwa sistem kontinu itu dinyatakan dalam bentuk F(x,dx,w)=0, atau dalam bentuk bergantung waktu yaitu sebagai F(x(t),dx(t),w(t))=0. Adapun transisi dari *hybrid automata* adalah himpunan batasan-batasan tertentu atau kondisi-kondisi yang bila terpenuhi menyebabkan sistem kontinu berubah menjadi sistem kontinu yang lain.

Hybrid automata digunakan untuk memodelkan sistem hybrid, yaitu sistem dimanadibangun dari komposisi sistem kontinu dan sistem diskrit.Tidak saja keduanya diletakkan bersamaan dalam satu permodelan, akan tetapi kedua sistem ini saling berinteraksi membangun sebuah sistem hibrid [13]. [14] menggunakan strong negasi ACTL untuk melakukan verifikasi terhadap sistem hibrid. Strong negasi adalah sebuah versi ACTL yang formula-formulanya dibangun dari negasi formula-formula dasar versi ACTL aslinya.

7. UML

UML (*Unified Modelling Language*) adalah kumpulan metode formal yang digunakan untuk melakukan permodelan pengembangan perangkat lunak dalam artifak-artifak diagram atau grafik formal yang konsisten.UML adalah sebuah bahasa visual untuk pembuatan diagram

dan untuk kebutuhan komunikasi di dalam pengembangan perangkat lunak [15].

Salah satu contoh metode formal yang terdapat dalam UML adalah diagram state dari UML. Diagram state ini merupakan *finite state automata* yang mana *node-node* nya atau state nya adalah sebuah kelas yang memiliki *property* rupa data dan *method*. Keunikan dari *finite state* versi UML ini adalah bahwa dia dapat membangun rantai state yang tersarang atau *nested*, dimana sebuah rantai state yang baru dapat dibangun di dalam sebuah state, dan seterusnya dapat dibangun secara tersarang terus menerus.

Penggunaan *abstract state machine* (ASMs) diusulkan oleh [16] untuk menyatakan UML state machine. Dalam hal ini dia membangun sebuah framework yang dapat menyatakan UML *state machine* dalam kalimat-kalimat yang persis tidak sekedar sebuah gambar diagram sebagaimana bawaan UML.

Dengan demikian [16] mengusulkan sebuah cara untuk bernalar atau berdeduksi dalam sintaks teks bagi sebuah UML *state machine*. [17] mengusulkan penggunaan UML untuk memodelkan sistem-sistem biologi. Sistem biologi telah berkembang semakin kompleks dalam permodelannya, [17] melihat bahwa permodelan sistem biologi dengan menggunakan persamaan-persamaan differensial mungkin dapat dirubah dalam cara pandang yang berbeda, dimana [17] melihat bahwa sistem biologi dapat dilihat dalam cara pandang *Object Oriented* (OO), dengan demikian permodelan UML dapat dilakukan untuk memodelkan sistem biologi yang kompleks dalam diagram-diagram UML.

Terlihat sebuah masa depan bagi sistem biologi bahwa mereka dapat melakukan rekayasa-rekayasa sistem biologi sebagaimana melakukan pengembangan perangkat lunak dengan terlebih dahulu membangun artifak-artifak UML yang lengkap [17].

8. CPN (*Colour Petri Net*)

CPN adalah sebuah petri net dengan tambahan bahwa dia boleh memiliki token yang berwarna atau petri yang berwarna.Token yang berwarna menggambarkan struktur data yang boleh campuran dari berbagai tipe data, dan warn petri yang berbeda-beda boleh menggambarkan struktur data yang berbeda atau kondisi yang berbeda-beda.

Sebuah CP net adalah sebuah tupel terurut CPN.CPN = (Σ,P,T,A,N,C,E,Mo), dimana:

Σ= himpunan berhingga tak kosong dari jenis (tipe) disebut juga himpunan warna.
P = himpunan berhingga tempat atau petry atau place ditulis p.
T= himpunan berhingga transisi ditulis t.
A = himpunan berhingga busur atau arc.
N = fungsi node yang memetakan sebuah arc di A ke dua node di (PXT)U(TXP).
C = fungsi warna, fungsi yang memetakan setiap warna pada Σ ke place di P.
E = fungsi ekpresi arc yang memetakan setiap arc ke multi set, multi set atas warna yang berasosiasi dengan place.
Mo = adalah inisialisasi penandaan (marking).

Dikembangkan oleh [18] sebuah cara pandang baru bahwa metode formal adalah sebuah masa depan yang menjanjikan bagi sistem biologi, dia mengatakan bahwa matematika (metode formal) adalah sebuah mikroskop masa depan bagi penelitian-penelitian di bidang biologi.

Dalam papernya [18] menunjukkan bahwa metode formal yang muncul dari inspirasi biologi, yaitu apa yang disebut sebagai *enhanced mobile membranes*, dapat dipadukan dengan *coloured petri net* untuk meninjau lebih luas sifat-sifat dalam *enhanced mobile membrane*.

Enhanced mobile membrane adalah sebuah metode formal yang dikembangkan berdasarkan inspirasi biologi. Dalam metode formal ini, sejumlah sel yang dinyatakan dalam membran, segala aktivitasnya dirumuskan dalam rule-rule dasar, berupa aturan-aturan produksi, kemudian sebuah permodelan proses biologi dalam level sel dapat dimodelkan berdasarkan metode formal ini.

[19] menggunakan *Coloured timed petri net* bersama dengan UML guna memodelkan dan mengembangkan *The Integrating the Healthcare Enterprise* (IHE), yaitu sebuah sistem enterprise yang membangun jejala alarm untuk memberi peringatan dan masukan terhadap perkembangan kesehatan pasien dalam sebuah sistem kesehatan atau rumah sakit.

9. Probabilistic FSA (*Finite State Automata*)

Probabilistic FSA adalah sebuah FSA tetapi non deterministik.Non deterministik artinyauntuk sebuah input a sebarang, state berpindah kepada lebih dari satu state yangmungkin.Misal state A, B, dan C. Diperoleh contoh:A--a-->B dan A--a-->C.

Untuk setiap input a dan cabang-cabang panahnya beri *probability* dimana berlaku aturan: Jumlah seluruh *probability* sebuah input pada sebuah state adalah 1.

Contoh:
A--a(0,6)-->B dan A--a(0,4)-->C.

Dimana 0,6 + 0,4 = 1

Jika hanya ada satu panah transisi bagi satu input, maka probability input pada satu panah transisi tersebut adalah 1.Jika tidak ada panah transisi bagi satu input pada sebuah state, maka probability input tersebut adalah 0.

Cara menjalankan sebuah string input pada sebuah *probabilistic* FSA adalah dengan melewatkan string pada seluruh kemungkinan.Ambil semua variasi string yang berakhir pada final state.Hitung total probability masing2 string pada jalurnya, dengan cara memperkalikan seluruh probability pada jalurnya.Lalu pilih yang nilai peluangnya maksimum sebagai jalur state yang terpilih dilewati oleh string.
Contoh :
String abca, melewati 3 macam jalur yang berakhir di final state.
Untuk jalur pertama: a(0,3).b(0,2).c(1).a(1) diperoleh (0,3).(0,2).(1).(1)=0,06
Untuk jalur kedua : a(0,4).b(0,1).c(1).a(1) diperoleh (0,4).(0,1).(1).(1)=0,04
Untuk jalur ketiga : a(0,3).b(0,3).c(1).a(1) diperoleh (0,3).(0,3).(1).(1)=0,09
Diperoleh jalur ketiga yang terpilih.

Dalam paper [20] mengusulkan sebuah permodelan Probabilistic FSA (PFSA) dalam sebuah ruang vektor yang ternormalisasi diatas medan bilangan nyata dengan terlebih dahulu mengemukakan landasan ide dalam ukuran (*measure*). [20] mengkonstruksikan sebuah ruang ukuran probabilistik (*probabilistic measure space*) diatas himpunan alphabet dan himpunan string kombinasinya, sebagaimana gagasan asli dari ruang ukuran probabilistik yang didefinisikan diatas himpunan bilangan rill, terbatas pada sebuah interval tertentu.

Nantinya oleh [20] pada himpuan string itu diberlakukan sifat-sifat σ-aljabar, dengan mendefinisikan himpunan string baru yaitu himpunan string dengan prefixnya adalah elemen-elemen dari himpunan string hingga dari seluruh kombinasi alphabet yang mungkin. Setelah σ-aljabar berhasil dikonstruksikan, [20] kemudian mengkonstruksikan sebuan ruang ukuran probabilistik (*probabilistik measure space*) yang nantinya digunakannya untuk membangun ruang vektor ternormalisasi.

[21] menggunakan *probabilistic finite state automata* untuk memodelkan *time series* dari signal, dimana signal dienkoding ke dalam blok-blok yang ditandai sebagai alphabet. Kemudian perambatan signal selanjutnya dilihat sebagai proses sekuensial dari alphabet tersebut membangun sebuah string secara probabilistik.Dengan demikian sebuah pola signal dapat dinyatakan oleh sebuah untai string yang probabilistik, yang berarti sebuah pola signal dapat dinyatakan oleh sebuah *probabilistic finite state automata* (PFSA) yang menerima string probabilistik tersebut.

10. ASM (*Abstract State Machine*)

ASM terdiri atas state awal (*initial state*) dan himpunan rule (JIKA MAKA) yang hendak digunakan untuk memproses atau mentransisikan state awal.

State awal adalah sebuah tupel yang terdiri dari sejumlah himpunan data, fungsi-fungsi yang menyatakan variabel, relasi-relasi yang merelasikan antar himpunan data dalam bentuk *tupel* juga, atau dalam bentuk relasi lain misal relasi sub himpunan, relasi elemen himpunan, relasi sama dengan, relasi lebih kecil, relasi lebih besar, dsb.Secara umum, state adalah struktur data dalam konstruksi teori himpunan dan logika.

Contoh:
State awal S_0.
S_0 = (H1,H2,H3,x,y,z,w, x elemen H1,y elemen H2, z elemen H3, x<y, w elemen H1xH2)

State awal dan seluruh *rule* dinyatakan dalam *pseudocode*.

Pseudocode untuk state awal.

State awal dinyatakan dalam *pseudocode* juga sebagai berikut:
H1 : *integer*
H2 : *real*
H3 : *string*
x : H1
y : H2
z : H3
w : H1xH2
x < y

Atau disingkat dalam bentuk sbb:
integer x:= 0
real y := 3.6
string z := ""
w : *record*
integer x:= 5
real y:= 4.5
end.
JIKA x < y THEN *true else false*
Menyatakan sebuah struktur data.

Pseudocode untuk *rule-rule* adalah sebagai berikut:
Contoh :
rule1: JIKA x < 10 THEN x := 6
rule2: JIKA x=6 THEN x+3

Gambaran ASM secara komputasi sequensial adalah sebagai berikut:
S_0 ---*rule1*---> S_1 ---*rule2*---> S_2

Dimana S_1 dan S_2 adalah state-state hasil *update* dari S_0.
Diperoleh sebuah gambar graf linier, *sequensial*, yang *totally ordered set*.

Gambaran ASM secara komputasi konkuren adalah sebagai berikut:
Sebuah gambaran *partially ordered set*.

Gambaran ASM secara komputasi terdistribusi adalah sebagai berikut: Sebuah *partially ordered set* dari sejumlah *totally ordered subset* atau *partially ordered subset*.Dimana *rule-rule* antar *subset* adalah *rulesend-receive message*. Sedang *rule-rule* di dalam *subset* adalah *rule-rule* sebagaimana biasanya.

[22] yang juga merupakan penggagas ASM sendiri, menunjukkan dalam papernya bahwa ASM dapat digunakan untuk meninjau teori bahasa dan menggunakannya dalam konteks yang lebih luas sebagai penerapan ASM dalam teori bahasa.

[23] menggunakan ASM untuk merumuskan spesifikasi dalam enginering, dan menggunakan SAT untuk memeriksa sifat kelengkapan (*completness*) dan konsistensi dari pada spesifikasi tersebut.Kelengkapan artinya bahwa spesifikasi memenuhi respon yang diperlukan untuk sebarang input dari bermacam-macam kelas input, dan konsistensi adalah berarti tidak terdapat kontradiksi dalam proposisi-proposisi spesifikasi.

Secara singkat, hasil kajian yang telah dibahas diatas dapat dilihat dalam Tabel 1 berikut ini.

TABEL I
HASIL KAJIAN PENELITIAN

No.	Model Metode Formal	Pengguna Metode	Hasil Penelitian
1	Actor Model	Joeri De Koster	Mengusulkan gagasan tentang bagaimana cara mengimplementasikan komputasi konkuren oleh actor-actor dengan cara menerapkan memory sharing atau secara umum component sharing, akan tetapi diterapkan secara efisien dan dapat mengatasi *deadlock* (dua actor atau lebih saling menunggu untuk menggunakan sebuah *resource*) ataupun *data races.*
2	Rebeca (Reactive Object Language)	Arni Hermann Reynisson	Mengusulkan gagasan tentang bagaimana jika mengimplementasikan rebeca berdasarkan pewaktuan, yaitu dengan mengenalkan sintaks baru pada sintaks rebeca yang sudah ada sebelumnya.
3	LTL *(Linier temporal logic)*	N. Benes	Mengusulkan cara untuk melakukan reduksi terhadap ruang state bagi sebuah sistem, dimana padanya direpresentasikan sebagai sebuah state/event LTL.
4	CTL *(Computation tree logic)*	Sjoerd Cranen	Mengemukakan cara untuk menerjemahkan LTL, CTL, dan CTL* ke dalam μ-*calculus*,

			untuk lebih mempermudah kalkulasi dikarenakan penggunaan LTL, CTL yang berarti juga CTL* dapat memberikan ledakan jumlah state dalam *run* nya.
5	ACTL *(Action Computation Tree Logic)*	Robert Meolic	Memodelkan lintasan kereta api dengan menggunakan *finite state automata*, dan merumuskan transisi-transisinya untuk kendaraan berupa kereta dan mobil yang melintasi lintasan kereta.
6	Hybrid Automata	Zhi Han	Menggunakan strong negasi ACTL untuk melakukan verifikasi terhadap sistem hibrid. Strong negasi adalah sebuah versi ACTL yang formula-formulanya dibangun dari negasi formula-formula dasar versi ACTL aslinya.
7	UML (*Unified Modelling Language*)	Ken Webb	Mengusulkan penggunaan UML untuk memodelkan sistem-sistem biologi. Sistem biologi telah berkembang semakin kompleks dalam permodelannya, Ken Webb[11] melihat bahwa permodelan sistem biologi dengan menggunakan persamaan-persamaan differensial mungkin dapat dirubah dalam cara pandang yang berbeda, dimana Ken Webb[11] melihat bahwa sistem biologi dapat dilihat dalam cara pandang *Object Oriented* (OO), dengan demikian permodelan UML dapat dilakukan untuk memodelkan sistem biologi yang kompleks dalam diagram-diagram UML.
8	CPN (Colour Petri Net)	Maria Pia Fanti	Menggunakan *Coloured timed petri net* bersama dengan UML guna memodelkan dan mengembangkan *The Integrating the Healthcare Enterprise* (IHE), yaitu sebuah sistem enterprise yang membangun jejala alarm untuk memberi peringatan dan masukan terhadap perkembangan kesehatan pasien dalam sebuah sistem kesehatan atau rumah sakit.
9	Probabilistic FSA (*Finite State Automata*)	Yicheng Wen	Mengusulkan sebuah permodelan Probabilistic FSA (PFSA) dalam sebuah ruang vektor yang ternormalisasi diatas medan bilangan nyata dengan terlebih dahulu mengemukakan landasan ide dalam ukuran (*measure*).
10	ASM (*Abstract State Machine*)	Yuri Gurevich	ASM dapat digunakan untuk meninjau teori bahasa dan menggunakannya dalam konteks yang lebih luas sebagai penerapan ASM dalam teori bahasa.

IV. PENUTUP

Demikian survei tentang berbagai macam metode formal yang telah dikonstruksikan orang dan diterapkan diberbagai bidang. Secara keseluruhan konstruksi metode formal dilakukan diatas gagasan teori himpunan dalam matematika atau pengembangannya semisal tipe teori, dan juga dikembangkan diatas logika proposisi (*first order logic*).

Metode formal telah menjadi tool yang menjanjikan diberbagai bidang diluar ilmu komputer sendiri, terutama di bidang sistem-sistem biologi, dan bidang lain yang mungkin dalam lingkup rekayasa mesin, elektronika, robotika, aeronotika, dan berbagai bidang lain yang mungkin diterapkan di masa depan.

Pengembangan kajian dapat dilanjutkan dengan banyak metode formal lainnya agar dapat menjadi acuan ataupun referensi bagi yang akan meneliti tentang pemodelan verifikasi dan validasi dengan menggunakan metode formal.

REFERENSI

[1] Carl Hewitt, Hendry Baker Jr, "Actors and Continous Functionals", Massachusetts Institut of Technology, Laboratory for Computer Science, 1977, Cambridge, Massachusetts.

[2] Christophe Scholliersa, Éric Tanterb, Wolfgang De Meutera, "Parallel actor monitors: Disentangling task-level parallelism from data partitioning in the actor model", Science of Computer Programming, Volume 80, Part A, 1 February 2014, Pages 52-64, (ScienceDirect.com).

[3] Joeri De Koster, Stefan Marr, Theo D'Hondt, Tom Van Cutsem, "Domains: Safe sharing among actors Original Research Article" Science of Computer Programming, In Press, Corrected Proof, Available online 18 February 2014, (ScienceDirect.com).

[4] Hossein Hojjat, "Rebeca2 Reference Manual version 1.0.2" Formal Lab, University of Teheran, Desember 18, 2006.

[5] Arni Hermann Reynisson, Marjan Sirjani, Luca Aceto, Matteo Cimini, Ali Jafari, Anna Ingolfsdottir, Steinar Hugi Sigurdarson, "Modelling and simulation of asynchronous real-time systems using Timed Rebeca" Original Research Article Science of Computer Programming, Volume 89, Part A, 1 September 2014, Pages 41-68, (ScienceDirect.com).

[6] Hamid Reza Shahriari, Mohammad Sadegh Makarem, Marjan Sirjani, Rasool Jalili, Ali Movaghar, "Vulnerability analysis of networks to detect multiphase attacks using the actor-based language Rebeca", Computers & Electrical Engineering, Volume 36, Issue 5, September 2010, Pages 874-885, (ScienceDirect.com).

[7] J. Barnat, J. Havlicek, P. Rockai, "Distributed LTL Model Checking with Hash Compation" Electrical notes in theoritical computer science 296 (2013) 79 93 (ScienceDirect.com).

[8] Salamah Salamah, Ann Gates, Vladick Kreinovick, "Validated Templates for Specifications of Complex LTL Formula", The Journal of Systems and Software 85 (2012) 1915-1926, (ScienceDirect.com).

[9] N. Benes, L. Brim, B. Buhnova, I. Cerna, J. Sochor, P.Varekova, "Partial Order Reduction for State/Event LTL with Application to Component-Interaction Automata" Science of Computer Programming 76 (2011) 877-890, (ScienceDirect.com).

[10] Sjoerd Cranen, Jan Friso Groote*, Michel Reniers "A Linier Translation from CTL* to the First Order Modal μ-calculus", Theoritical Computer Science 412 (2011) 3129 3139, (ScienceDirect.com).

[11] Robert Meolic, Tatjana Kapus, Zmago Brezocˇnik, "CTL and ACTL patterns", EUROCON 2001, July 5-7, Bratislava, Slovak Republic.

[12] Robert Meolic, Tatjana Kapus, Zmago Brezocˇnik, "Model Checking: A Formal Method for Safety Assurance of Logistic Systems", 2nd Congress Transport - Trafffic - Logistics, October 2-3, 2000, Portorozˇ, Slovenia.

[13] Arjan Van der Schaft, Hans Schumacher, "An Introduction to Hybrid Dynamical Systems", Department of Systems, Signals Faculty of Mathematical Sciences and Control University of Twente and CWI, Centre for Mathematics and Computer Science, Amsterdam and Department of Econometrics and Center for Economic Research, Tilburg University.

[14] Zhi Han, Alongkrit Chutinan, Bruce H. Krogh, "ACTL Strong Negation and its Application to Hybrid System

Verification", Control Engineering Practice 14 (2006) 1259-1267, (ScienceDirect.com).

[15] Sinan Si Alhir, "Learning UML", Publisher O'Really, July 2003, ISBN 0-596-00344-7.

[16] Egon Bo¨rger, Alessandra Cavarra, Elvinia Riccobene, "On formalizing UML state machines using ASMs", Information and Software Technology 46 (2004) 287–292, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[17] Ken Webb, Tony White, "UML as a cell and biochemistry modeling language" , BioSystems 80 (2005) 283–302, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[18] Bogdan Aman, Gabriel Ciobanu. "Properties of enhanced mobile membranes via coloured Petri nets", Information Processing Letters 112 (2012) 243–248, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[19] Maria Pia Fanti, Stefano Mininel, Walter Ukovich, Federica Vatta, "Modelling alarm management workflow in healthcare according to IHE framework by coloured Petri Nets", Engineering Applications of Artificial Intelligence 25 (2012) 728–733, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[20] Yicheng Wen, Asok Ray, "Vector space formulation of probabilistic finite state automata", Journal of Computer and System Sciences 78 (2012) 1127–1141, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[21] Kushal Mukherjee, Asok Ray, "State splitting and merging in probabilistic finite state automata for signal representation and analysis", Signal Processing 104 (2014) 105–119, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[22] Yuri Gurevich, Margus Veanes, Charles Wallace, "Can Abstract State Machine be useful in language theory?", Theoritical Computer Science 376 (2007) 17-29, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[23] Martin Ouimet, Kristina Lundqvist, "Automated Verification of Completness and Consistency of Abstract State Machine Specification using a SAT Solver", Electronic Notes Theoritical Computer Science 190 (2007) 85-97, http://www.sciencedirect.com.ezproxy.ugm.ac.id.

[24] Miguel Carillo, David A. Rosenblueth, CTL Update of Kripke Models Througs Protections", Artificial Intelligence 211 (2014) 51-74, (ScienceDirect.com).