**<CONTOH JUDUL: PENGEMBANGAN SISTEM INFORMASI DI PUSKESMAS DAERAH PEDESAAN>**

**Laporan Tugas Akhir I**

**Disusun sebagai syarat kelulusan mata kuliah**

**IF4091/Tugas Akhir I dan Seminar**

**Oleh**

**Hafizh Afkar Makmur**

**NIM : 13514062**



**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO & INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**Januari 2021**

**<CONTOH JUDUL: PENGEMBANGAN SISTEM INFORMASI DI PUSKESMAS DAERAH PEDESAAN>**

**Laporan Tugas Akhir I**

**Oleh**

**Hafizh Afkar Makmur**

**NIM : 13514062**

**Program Studi Teknik Informatika**

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bandung, <tanggal>

Mengetahui,

Pembimbing,

<Nama Pembimbing>

NIP. <NIP Pembimbing>

**<CONTOH JUDUL: PENGEMBANGAN SISTEM INFORMASI DI PUSKESMAS DAERAH PEDESAAN>**

**Laporan Tugas Akhir I**

**Oleh**

**Hafizh Afkar Makmur**

**NIM : 13514062**

**Program Studi Teknik Informatika**

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bandung, <tanggal>

Mengetahui,

|  |  |
| --- | --- |
| Pembimbing I,  <Nama Pembimbing-II>  NIP. <NIP Pemb II> | Pembimbing II,  <Nama Pembimbing-II>  NIP. <NIP Pemb II> |

DAFTAR ISI

[DAFTAR ISI 1](#_Toc60643217)

[DAFTAR LAMPIRAN 2](#_Toc60643218)

[DAFTAR GAMBAR 3](#_Toc60643219)

[DAFTAR TABEL 4](#_Toc60643220)

[DAFTAR SINGKATAN 5](#_Toc60643221)

[DAFTAR ISTILAH 6](#_Toc60643222)

[BAB I PENDAHULUAN 7](#_Toc60643223)

[I.1 Latar Belakang 7](#_Toc60643224)

[I.2 Rumusan Masalah 9](#_Toc60643225)

[I.3 Tujuan 9](#_Toc60643226)

[I.4 Batasan Masalah 10](#_Toc60643227)

[I.5 Metodologi 10](#_Toc60643228)

[I.6 Jadwal Pelaksanaan Tugas Akhir 11](#_Toc60643229)

[BAB II STUDI LITERATUR 12](#_Toc60643230)

[II.1 Metode Formal 12](#_Toc60643231)

[II.1.1 Pengenalan Metode Formal 12](#_Toc60643232)

[II.1.2 Spesifikasi dan Verifikasi Program 17](#_Toc60643233)

[II.2 Pemrograman Fungsional 19](#_Toc60643234)

[II.3 Haskell 19](#_Toc60643235)

[BAB III <RENCANA PENYELESAIAN MASALAH> 20](#_Toc60643236)

[III.1 20](#_Toc60643237)

[III.2 20](#_Toc60643238)

[III.3 20](#_Toc60643239)

[DAFTAR PUSTAKA 22](#_Toc60643240)

DAFTAR LAMPIRAN

[Lampiran A. Contoh Judul Lampiran 7](#_Toc406869964)

[A.1 Contoh Judul Anak Lampiran 7](#_Toc406869965)

DAFTAR GAMBAR

<jika diperlukan>

[Gambar II.1. Tahapan konstruksi koleksi retorik kalimat 3](#_Toc406855725)

DAFTAR TABEL

<jika diperlukan>

[Tabel II.1. Pengelompokan *Tag* MARC-21 4](#_Toc406855720)

DAFTAR SINGKATAN

<jika diperlukan>

DAFTAR ISTILAH

<jika diperlukan>

# PENDAHULUAN

## Latar Belakang

Sudah menjadi rahasia umum bahwa seorang pemrogram menginginkan programnya bebas dari kesalahan. Banyak waktu yang sudah terhabiskan oleh berbagai perusahaan teknologi untuk memastikan bahwa produk yang mereka rilis tidak memiliki kecacatan berarti yang mungkin saja bisa menelan korban jiwa. Kesalahan atau bug yang bisa ditangkap diawal proses pengembangan sebuah program bisa diatasi dengan lebih mudah daripada bug yang hanya disadari jauh dalam sebuah proses pengembangan. Salah satu metode yang dikembangkan oleh para ilmuwan komputasi untuk bisa menangkap sebanyak-banyaknya atau bahkan seluruh kesalahan yang mungkin terjadi dalam sebuah program pada tahap yang seawal mungkin adalah Metode Formal.

Metode Formal merupakan sebuah teknik untuk memanfaatkan sistem pembuktian matematika untuk membuktikan bahwa sebuah program akan berjalan sebagimana spesifikasi yang sudah ditetapkan. Teknik ini biasanya dilakukan dengan merepresentasikan bagian-bagian dari spesifikasi kebutuhan yang diinginkan dan program yang dikembangkan ke dalam simbol-simbol matematika yang kemudian akan dibuktikan kecocokannya dengan menggunakan berbagai teknik matematika yang sudah paten. Teknik ini bisa dilakukan baik secara manual maupun secara otomatis dengan menggunakan beberapa program yang sudah dirilis sebelumnya seperti Z3, CVC4, atau MathSat. Program otomatis ini bisa membantu mempercepat proses pencarian bug yang sebelumnya berpotensi menjadi sebuah proses whack-a-mole tiada akhir menjadi sebuah proses yang bisa dengan mudah mengurangi jumlah kesalahan yang mungkin terjadi dalam program jauh sebelum kesalahan itu biasanya akan disadari dengan metode lain. Faktanya, Metode Formal sudah digunakan dan direkomendasikan oleh baik IEC, ESA, FAA, maupun NASA sebagai sebuah subjek yang harus dikuasai oleh setiap ilmuwan maupun teknisi komputasi berikut dengan kompetensi matematika yang dibutuhkan untuk memahaminya (Baier, 2008).

Namun dibalik kelebihan dari Metode Formal juga bersembunyi kekurangannya. Metode Formal bukanlah merupakan sebuah metode yang mudah untuk dipelajari oleh seorang pemrogram yang tidak terlalu tertarik melakukan studi matematika apalagi praktisi pemrograman amatir yang semakin banyak bermunculan di masa ini. Dibutuhkan usaha tambahan yang tidak sepele untuk menganalisis, mengidentifikasi, dan mengubah karakteristik utama spesifikasi kebutuhan dan program menjadi representasi matematika yang bisa digunakan untuk analisis Metode Formal. Usaha yang dibutuhkan bisa menjadi sangat besar sehingga keuntungan yang bisa diraih dengan menggunakan metode ini dianggap tidaklah lebih baik daripada biaya yang harus dibayar sehingga metode ini hanya digunakan untuk sistem kritikal yang memiliki potensi kerugian yang sangat besar untuk setiap kesalahan yang terjadi sehingga beban untuk melakukan Metode Formal bisa dijustifikasi (Pena, 2016). Terlihatlah bahwa jika seseorang menginginkan metode ini untuk lebih banyak digunakan oleh kalangan pemrogram, maka dibutuhkan cara untuk mengurangi usaha dan beban yang dibutuhkan untuk menggunakan metode ini.

Salah satu metode untuk memudahkan pengembangan metode formal untuk perangkat lunak adalah dengan memungkinkan logika-logika untuk program dituliskan langsung berdampingan dengan program itu sendiri. Salah satu perangkat lunak yang dikembangkan untuk tujuan tersebut adalah Liquid Haskell untuk bahasa pemrograman Haskell. Pada dasarnya bahasa pemrograman berparadigma fungsional sudah mengalami keunggulan untuk pengaplikasian Metode Formal karena sifat bahasa tersebut yang transparan dan tidak menghasilkan “efek samping” saat mengeksekusi suatu fungsi sehingga seorang analis bisa dengan mudah melihat efek dari interaksi antara berbagai fungsi tanpa harus mempertimbangkan bahwa hasil dari interaksi tersebut mungkin berbeda dengan masukan yang sama; sama seperti sebuah fungsi matematika yang konvensional. Karena itu Haskell sebagai sebuah bahasa fungsional murni juga memungkinkan kemudahan pengaplikasian Metode Formal pada program yang dituliskan pada bahasa tersebut. Liquid Haskell memanfaatkan kemudahan ini dengan mengintegrasikan SMT Solver seperti Z3 atau CVC4 dan memanfaatkan perangkat tersebut untuk memastikan keabsahan program yang dibuat pada Haskell secara otomatis. Hal ini memungkinkan penulisan spesifikasi langsung pada program (atau dalam file yang berdampingan) dan kemudian melakukan pengecekan otomatis program terhadap spesifikasi yang sudah tertulis sehingga proses Metode Formal menjadi jauh lebih mudah.

Sebagai seorang pelajar dalam ilmu komputasi, mahasiswa S1 Teknik Informatika ITB (IF ITB) juga merupakah sebuah pihak yang akan sangat diuntungkan dalam mempelajari Metode Formal. Namun seperti pemrogram lain yang sudah disebutkan sebelumnya, hal ini merupakan suatu hal yang tidak mudah dilakukan dan waktu yang akan dihabiskan mungkin saja lebih baik digunakan untuk mengajarkan materi studi lain. Menggunakan kemudahan yang ditawarkan oleh Liquid Haskell yang sudah disebutkan sebelumnya, diharapkan mahasiswa IF ITB dapat memahami materi Metode Formal ini dalam waktu yang lebih mudah untuk dialokasikan serta dapat memberikan pemahaman yang lebih mumpuni.

## Rumusan Masalah

Berdasarkan latar belakang dan fokus masalah tersebut, terumuskan beberapa masalah yang sebaiknya diselesaikan:

1. Bagaimanakah sistem terbaik untuk menjelaskan Metode Formal kepada Mahasiswa S1 Teknik Informatika ITB dengan menggunakan Liquid Haskell?
2. Bagaimanakah penyusunan sistem materi yang baik untuk diajarkan dalam sistem ini?

## Tujuan

Penelitian ini bertujuan untuk:

1. Melakukan analisis materi Metode Formal yang bisa disarikan sebagai materi dalam learning tools.
2. Melakukan implementasi learning tools Metode Haskell dengan berbasis Haskell dan Liquid Haskell..

## Batasan Masalah

Batasan permasalahan dalam penelitian ini adalah sebagai berikut:

1. Target subjek pengajaran ini adalah Mahasiswa S1 Teknik Informatika ITB ataupun seorang pemrogram yang memiliki kompetensi dan pengalaman dalam membuat program dan mengetahui bahasa pemrograman Haskell.
2. Materi tidak ditujukan untuk mengajarkan teknik pembuktian matematika Metode Formal secara detail namun lebih merupakan sebuah panduan sederhana dalam menggunakan Metode Formal untuk meningkatkan akurasi program yang ditujukan untuk pemrogram yang tidak membutuhkan pengetahuan matematika mendalam untuk memahami materi tersebut.

## Metodologi

Metodologi pengerjaan penelitian ini adalah sebagai berikut:

1. Studi Literatur

Pada tahap ini dilakukan penelitian mendalam terhadap Metode Formal dan Liquid Haskell dengan menggunakan berbagai referensi baik daring maupun luring serta melakukan konsultasi kepada ahli dalam bidang yang berkaitan.

1. Perancangan sistem

Dilakukan analisis serta seleksi materi Metode Formal yang cocok untuk diajarkan serta ditentukan perangkat terbaik yang akan digunakan sebagai dasar untuk pembuatan sistem.

1. Implementasi sistem

Dilakukan implementasi sistem dengan menggunakan perangkat yang sudah ditentukan dan menggunakan materi yang sudah terseleksi.

## Jadwal Pelaksanaan Tugas Akhir

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tahapan Pengerjaaan | Feb | Mar | Apr | Mei | Jun | Jul | Aug | Sep | Okt |
| Studi Literatur |  |  |  |  |  |  |  |  |  |
| Perancangan Sistem |  |  |  |  |  |  |  |  |  |
| Implementasi Sistem |  |  |  |  |  |  |  |  |  |

Tabel I‑1. Jadwal Pengerjaan Tugas Akhir

# STUDI LITERATUR

## Red-Black Tree

### Binary Tree

### Red-Black Tree

### Algoritma insersi Okazaki

### Algoritma penghapusan Matt Might

## Metode Formal

## Pengenalan Metode Formal

Metode Formal merupakan sebuah teknik penerapan prinsip-prinsip matematika dalam memodelkan dan menganalisis sistem ICT (Teknologi Komunikasi dan Informasi) (Baier & Katoen, 2008). Dengan kata lain, Metode Formal didefinisikan sebagai ilmu matematika untuk sistem perangkat keras dan perangkat lunak komputer (Holloway, 1997). Menurut Monin, sebenarnya istilah “Teknik Formal” lebih pantas untuk digunakan dibandingkan “Metode Formal” karena teknik ini belum memiliki metodologi yang baku. Namun karena istilah Metode Formal lebih populer maka untuk selanjutnya dalam tulisan ini istilah Metode Formal akan menjadi istilah yang digunakan (Monin, 2003).

Metode Formal merupakan salah satu teknik verifikasi yang “sangat direkomendasikan” untuk pengembangan perangkat lunak dan sistem kritikal oleh standar praktik terbaik yang dimiliki oleh IEC (Komisi Elektroteknik Internasional) dan ESA (Agensi Antariksa Eropa) (Baier & Katoen, 2008). Hasil laporan investigasi yang dilakukan oleh FAA (Otoritas Penerbangan Federal Amerika) dan NASA (Administrasi Aeronautika dan Antariksa Nasional Amerika) mengenai penggunaan Metode Formal menunjukkan bahwa Metode Formal haruslah menjadi bagian dari pendidikan seluruh ilmuwan komputasi dan insinyur sistem perangkat lunak, seperti bagaimana ilmu matematika terapan merupakan ilmu yang wajib dimiliki oleh setiap insinyur lainnya.

#### Alasan penggunaan Metode Formal

##### Fatalitas kesalahan sistem perangkat lunak

Kesalahan pada unit pembagian bilangan titik mengambang (*floating point*) milik Intel pada tahun 90an menyebabkan kerugian mencapai 470 juta dolar Amerika untuk mengganti seluruh prosesor yang cacat dan juga merusak reputasi Intel sebagai pembuat cip komputer yang bisa diandalkan. Kesalahan perangkat lunak dalam sistem penanganan bagasi memundurkan pembukaan sebuah bandara di Denver selama 9 bulan yang menyebabkan kerugian sekitar 1.1 juta Dolar Amerika per hari. Kesalahan selama 24 jam pada sebuah sistem pemesanan tiket daring internasional akan menyebabkan kebangkrutan perusahaan tersebut karena hilangnya pesanan. Bila kesalahan terjadi pada sistem keamanan, akibat yang ditimbulkan bisa menjadi bencana. Kecacatan fatal pada perangkat lunak kontrol pada misil Ariadne-5, wahana antariksa Mars Pathfinder, serta pesawat-pesawat milik Airbus sudah menjadi berita utama di seluruh dunia dan menjadi kasus-kasus yang terkenal. Perangkat lunak juga digunakan pada kontrol proses sistem yang kritikal pada pabrik kimia, pembangkit listrik tenaga nuklir, sistem lalu lintas, sistem penghalau badai, serta sistem penting lainnya yang bisa menimbulkan bencana dan kerugian besar jika terjadi kesalahan pada sistem tersebut. Salah satu contohnya adalah sebuah kecacatan perangkat lunak pada mesin terapi radiasi Therac-25 yang menyebabkan kematian 6 pasien kanker di antara 1985 dan 1987 disebabkan oleh overdosis paparan radiasi. Meningkatnya ketergantungan aplikasi kritikal pada pemrosesan informasi menyebabkan pentingnya meningkatkan reliabilitas dalam proses desain sistem ICT.

##### Reliabilitas pengembangan perangkat lunak

Pengembangan perangkat lunak saat ini sudah terkenal sebagai sebuah proses yang lambat dalam memberikan hasil serta sulit diprediksi dan tidak bisa diandalkan dalam operasi (Holloway, 1997). Menurut sebuah artikel yang ditulis pada 1994 oleh Wyatt Gibbs, “Studi menunjukkan bahwa untuk setiap 6 sistem perangkat lunak skala besar baru yang dioperasikan, 2 akan dibatalkan. Rata-rata waktu pengembangan perangkat lunak melampaui jadwal yang ditentukan sebanyak 50%. Perangkat lunak yang lebih besar bahkan membutuhkan waktu yang lebih lama lagi. 75% dari seluruh sistem skala besar memiliki kesalahan dalam operasi yang menyebabkan mereka tidak berfungsi sebagaimana mestinya atau bahkan tidak digunakan sama sekali.” Dibandingkan dengan disiplin keinsinyuran lainnya, teknik perangkat lunak terlihat sangat buruk. Namun ini tidak terlalu mengejutkan karena setidaknya dalam dua aspek, perangkat lunak berbeda dengan objek fisik, material, dan sistem yang ditangani oleh ilmu teknik pada umumnya.

Pertama, pada sistem fisik perubahan yang halus pada masukan akan menghasilkan perubahan yang halus pada keluaran. Dengan kata lain, sistem fisik merupakan sebuah sistem yang kontinu. Hal ini memungkinkan perilaku sistem untuk ditentukan hanya dengan memberikan beberapa masukan yang kemudian diterjemahkan menggunakan interpolasi dan ekstrapolasi untuk menentukan perilaku sistem pada masukan yang tidak dites. Sistem perangkat lunak, berbeda dengan sistem fisik, merupakan sebuah sistem yang diskontinu. Perubahan kecil pada masukan bisa menghasilkan perubahan yang signifikan pada beberapa penentuan keputusan dalam perangkat lunak dan menyebabkan keluaran yang sangat berbeda. Hasilnya, interpolasi atau ekstrapolasi tidak bisa digunakan untuk memprediksi keluaran dari masukan yang tidak dites karena risiko yang tinggi dan bisa menyebabkan hasil yang tidak diinginkan. Selain itu sistem komputer semakin hari memiliki kompleksitas yang semakin tinggi. Dengan naiknya kompleksitas maka semakin banyak pula kemungkinan kecacatan desain yang akan terjadi. Pada sisi baiknya, sistem perangkat lunak cenderung tidak memiliki keausan tidak seperti sistem fisik. Sehingga jika sistem perangkat lunak sudah dibuktikan reliabilitasnya maka reliabilitas itu bisa bertahan selama bertahun-tahun tanpa mengalami keusangan.

#### Contoh pengaplikasian Metode Formal

Salah satu eksperimen skala besar pengaplikasian Metode Formal dilakukan pada proyek CICS yang dilakukan oleh IBM (Taman Huxley, Inggris) dalam kerja sama dengan Universitas Oxford (Baier & Katoen, 2008). Tujuan dari proyek ini adalah untuk melakukan restrukturisasi mayor terhadap sebuah sistem manajemen transaksi besar yang sudah berjalan. Dalam keseluruhan sistem terdapat 800.000 baris kode yang tertuliskan dalam bahasa Assembly dan Pias, sebuah bahasa tingkat tinggi khusus. Ada 268.000 baris kode yang dimodifikasi dan ditulis ulang dan di dalam baris-baris kode itu 37.000 diantaranya diberikan spesifikasi formal menggunakan notasi Z. Berbagai prosedur pengukuran dilakukan untuk mengevaluasi dampak Metode Formal terhadap produktivitas dan kualitas. Hasilnya adalah sebagai berikut:

1. Biaya pengembangan berkurang 9 persen
2. Terjadi 2.5 kali lebih sedikit error pada bagian program yang dikembangkan menggunakan notasi Z dalam 8 bulan pertama setelah instalasi
3. Error yang dilaporkan memiliki keseriusan lebih rendah

Eksperimen ini merupakan sebuah eksperimen yang menarik karena banyaknya kode yang terlibat. Namun, eksperimen ini memiliki cakupan yang terbatas karena eksperimen hanya melakukan eksperimen terhadap notasi formal Z serta tidak memperhitungkan teknik-teknik pembuktian yang dilakukan.

#### Kelemahan Metode Formal

Metode Formal bukan merupakan sebuah teknik adiguna yang bisa digunakan untuk memverifikasi seluruh program yang dibuat. Menurut Jean-Francois Monin (Monin, 2003), beberapa kelemahan dari teknik ini yang harus diperhatikan adalah:

1. Selalu ada jarak antara spesifikasi format yang tertulis dengan objek yang direpresentasikan. Hal ini serupa seperti yang terjadi pada ilmu fisika. Tidak bisa dibuktikan bahwa hukum-hukum fisika benar-benar merepresentasikan dunia nyata namun bisa diyakini bahwa hukum-hukum itu cukup dekat dengan dunia nyata untuk tujuan saat ini.
2. Dibutuhkan waktu untuk memahami dan menggunakan notasi yang dipakai. Keyakinan terhadap kebenaran dari sebuah spesifikasi program hanya bisa diketahui dengan proses analisis yang mendalam. Metode Formal juga membutuhkan aspek teori serta kemampuan untuk memanipulasi berbagai teknik matematika yang ada.
3. Lebih banyak waktu yang akan dihabiskan pada tahap awal program (spesifikasi, desain). Eksperimen menunjukkan bahwa waktu yang dihabiskan pada tahap awal ini biasanya akan dikompensasi dengan waktu yang berkurang pada tahap akhir (tes, integrasi). Formalisasi membuka di awal beberapa masalah yang biasanya hanya akan ditemukan pada proses akhir seperti saat *debugging* yang mungkin akan membutuhkan penanganan yang lebih besar karena sulitnya mengubah program yang sudah dibuat secara besar-besaran. Berbagai kesulitan yang ditemukan dalam melakukan formalisasi sebenarnya merupakan refleksi terhadap kesulitan proyek yang dilakukan namun proses modelisasi menunjukkan kompleksitas permasalahan yang tidak terlihat pada pandangan pertama.

##### Minimnya penggunaan Metode Formal

Pada kenyataannya setelah 40 tahun Metode Formal dengan segala kelebihan dan kekurangannya masih sangat jauh dari pengaplikasian pada pemrograman sehari-hari. Menurut Ricardo Pena (Peña, 2017), ada beberapa alasan atas terjadinya situasi ini:

1. Dibutuhkan waktu dan usaha yang cukup besar untuk memformalisasi spesifikasi pada kebutuhan dengan menuliskan prakondisi dan pos kondisi untuk setiap kebutuhan.
2. Dibutuhkan usaha yang lebih besar lagi untuk menentukan *loop invariant* (variabel yang selalu konstan selama sebuah *loop* berjalan) serta menentukan asumsi lain yang kritikal dalam program.
3. Bahkan setelah menuliskan seluruh hal tersebut, untuk menuliskan pembuktian program tersebut secara manual dibutuhkan ruang bahkan mencapai 5 sampai 10 kali panjang program yang dibuktikan.

Pada umumnya, verifikasi formal bisa memberikan manfaat yang jelas namun investasi usaha yang dibutuhkan untuk melakukan hal tersebut sangatlah tinggi. Hal ini menyebabkan Metode Formal jarang digunakan dan hanya dilakukan untuk program kritikal yang akan memberikan kerugian yang sangat besar untuk setiap kesalahan yang terjadi sehingga investasi yang besar untuk melakukan Metode Formal bisa dijustifikasi. Usaha besar yang dibutuhkan untuk melakukan Metode Formal ini harus dikurangi untuk bisa memungkinkan Metode Formal untuk digunakan lebih luas lagi kepada lebih banyak pemrogram yang membutuhkannya.

### Spesifikasi dan Verifikasi Program

Pada dasarnya, dua komponen paling utama dalam melakukan Metode Formal adalah spesifikasi dan verifikasi. Spesifikasi merupakan formalisasi atau penulisan kebutuhan yang dimiliki dalam notasi yang formal. Verifikasi adalah proses untuk membuktikan kebenaran suatu program terhadap spesifikasi yang sudah dituliskan sebelumnya.

#### Spesifikasi Program

Representasi paling sederhana untuk spesifikasi program adalah pasangan (prakondisi, pos kondisi) (Monin, 2003). Prakondisi adalah asumsi mengenai kondisi yang relevan yang terjadi sebelum eksekusi program. Pos kondisi adalah asumsi mengenai hasil yang diinginkan setelah program yang dilakukan. Asumsi-asumsi tersebut dituliskan menggunakan formula logis yang memiliki arti matematis sehingga bisa dilakukan kalkulasi matematika terhadap asumsi tersebut. Verifikasi kemudian merupakan sebuah proses untuk membuktikan bahwa sebuah program yang memenuhi prakondisi harus melakukan aksi yang pada akhirnya memenuhi pos kondisi.

Berbagai properti yang dituliskan dalam spesifikasi bisa cukup elementer seperti menyatakan bahwa hasil tidak pernah melampaui nilai tertentu, program akan selalu berakhir (tidak berjalan selamanya), dan seterusnya. Verifikasi bergantung pada spesifikasi dalam menentukan apa yang program harus dan tidak boleh lakukan. Kesalahan hanya ditemukan jika program tidak memenuhi spesifikasi tertentu. Sistem dianggap “benar” jika sistem mampu memenuhi seluruh spesifikasi. Jadi kebenaran suatu program selalu merupakan sebuah properti yang relatif terhadap spesifikasi bukan merupakan sebuah properti yang absolut pada sistem.

#### Verifikasi Program

Menurut Michael Huth (Huth & Ryan, 2004), ada berbagai pendekatan dalam melakukan verifikasi:

1. Berbasis Bukti dan Berbasis Model. Dalam Verifikasi Berbasis Bukti, deskripsi sistem dituliskan dalam kumpulan formula Γ dan spesifikasi dituliskan dalam formula yang lain φ. Metode verifikasi kemudian mencoba mencari bukti bahwa Γ |− φ atau dengan kata lain Γ mengimplikasikan φ. Hal ini biasanya membutuhkan panduan dan keahlian pengguna. Dalam pendekatan berbasis model, sistem direpresentasikan sebagai sebuah model M dalam teori logika yang sesuai. Spesifikasi kembali direpresentasikan sebagai formula φ dan metode verifikasi menentukan apakah model M memenuhi φ (ditulis M φ). Komputasi ini biasanya bisa dilakukan secara otomatis untuk model berhingga.
2. Derajat otomatisasi. Verifikasi memiliki berbagai derajat untuk seberapa otomatis metode verifikasi bisa dijalankan mulai dari dilakukan dengan manual secara keseluruhan ataupun otomatis. Banyak teknik yang menggunakan komputer berada di antara dua ekstrem itu.
3. Properti atau keseluruhan. Spesifikasi mungkin hanya mendeskripsikan sebagian properti dari sebuah sistem atau mungkin keseluruhan perilaku. Dibutuhkan usaha yang jauh lebih berat untuk memverifikasi spesifikasi yang mendeskripsikan keseluruhan perilaku sistem.
4. Domain aplikasi. Domain aplikasi bisa memiliki banyak arti mulai dari apakah verifikasi dilakukan pada perangkat lunak atau perangkat keras, berurutan atau paralel, memiliki akhir atau reaktif, dan sebagainya. Pada dasarnya verifikasi pada perangkat keras jauh lebih vital untuk dilakukan seawal mungkin karena biaya untuk mengganti bagian kode yang salah pada sistem perangkat keras jauh lebih tinggi dibandingkan dengan sistem perangkat lunak.
5. Sebelum atau sesudah pengembangan. Verifikasi akan memberikan manfaat yang lebih baik jika dilakukan di awal pengembangan sistem karena error yang ditangkap akan bisa ditangani dengan biaya yang lebih murah.

Menurut Nikki Vazou (Vazou et al., 2017), kemungkinan perbedaan pendekatan lain adalah apakah verifikasi dilakukan secara intrinsik atau ekstrinsik. Verifikasi intrinsik dilakukan terhadap program secara langsung sedangkan verifikasi ekstrinsik membutuhkan kode khusus lain yang khusus dituliskan untuk tujuan verifikasi. Contoh dari sistem verifikasi yang memiliki perbedaan ini adalah Liquid Haskell dan Coq. Liquid Haskell mampu menjalankan verifikasi intrinsik karena spesifikasi dituliskan bersamaan dengan program dan SMT Solver dapat dilakukan untuk menganalisis isi program sekaligus dengan spesifikasi yang tertuliskan secara otomatis. Pada Coq pembuktian harus dilakukan secara manual oleh pengguna sehingga pembuktian harus dituliskan oleh pengguna sendiri dalam bahasa Coq. Karena pembuktian pada Liquid Haskell merupakan pembuktian implisit yang dilakukan oleh SMT Solver sedangkan pembuktian pada Coq dituliskan secara eksplisit oleh pengguna, maka pembuktian yang ditulis pada verifikasi ekstrinsik jauh lebih mudah dibaca dan digunakan. Hal ini juga dikarenakan narasi pembuktian hanya akan tertulis dalam bahasa verifikasi tersebut tidak bercampur dengan kode implementasi program.

## Haskell

# <RENCANA PENYELESAIAN MASALAH>

## 

## 

## 

Tujuan utama penulisan bab ini adalah untuk menguraikan rencana penyelesaian masalah tugas akhir yang merupakan bab penutup Laporan Tugas Akhir I. Gantilah judul sesuai konteks persoalan, misalnya: *Analisis Masalah dan Rancangan Solusi Pembangunan Sistem ABC*. Bab ini mencakup antara lain:

1. Deskripsi dan analisis persoalan yang terkait dengan Rumusan Masalah, misalnya menjelaskan secara detail latar belakang dan masalah yang menjadi dasar munculnya topik, menunjukkan gap/celah antara kondisi saat ini dengan kondisi yang diharapkan, dan kaitan antara sistem/aplikasi yang dikembangkan dengan sistem/aplikasi lain yang terkait.
2. Analisis solusi yang terdiri dari pilihan alternatif solusi yang dapat digunakan untuk setiap permasalahan berdasarkan hasil studi literatur atau survei, pemilihan solusi beserta justifikasinya.
3. Deskripsi umum solusi yang dipilih, mencakup:
   1. Langkah-langkah yang dibutuhkan untuk mencapai solusi masalah tugas akhir, berikut penjelasannya.
   2. Modul/subsistem/komponen yang akan dikembangkan untuk menyelesaikan masalah, berikut penjelasannya.
   3. Alur umum algoritma atau langkah-langkah pengembangan sistem dan penjelasannya.
   4. Penggunaan kakas yang diperlukan

Dianjurkan untuk menggunakan diagram sebagai pendukung penjelasan bagian ini.

DAFTAR PUSTAKA

Baier, C., & Katoen, J.-P. (2008). Principles Of Model Checking. In *MIT Press*.

Holloway, C. M. (1997). Why engineers should consider formal methods. *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, *1*(October). https://doi.org/10.1109/dasc.1997.635021

Huth, M., & Ryan, M. (2004). *Logic in CS: Modelling and Reasoning about Systems* (Vol. 66, Issue 1).

Monin, J.-F. (2003). Understanding Formal Methods. In *Understanding Formal Methods*. https://doi.org/10.1007/978-1-4471-0043-0

Peña, R. (2017). An introduction to liquid Haskell. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, *237*, 68–80. https://doi.org/10.4204/EPTCS.237.5

Sabry, A. (1998). What is a purely functional language? *Journal of Functional Programming*, *8*(1), 1–22. https://doi.org/10.1017/S0956796897002943

Turner, D. A. (1985). Miranda: A non-strict functional language with polymorphic types. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *201 LNCS*, 1–16. https://doi.org/10.1007/3-540-15975-4\_26

Vazou, N., Lampropoulos, L., & Polakow, J. (2017). A tale of two provers: Verifying monoidal string matching in liquid Haskell and Coq. *ACM SIGPLAN Notices*, *52*(10), 63–74. https://doi.org/10.1145/3122955.3122963

1. Contoh Judul Lampiran
   1. Contoh Judul Anak Lampiran

Contoh anak lampiran