Bab II Proses Spesifikasi dan Verifikasi

Bagian ini akan menerangkan pengertian dari proses spesifikasi dan verifikasi pada perangkat lunak serta penjelasannya dengan yang berkaitan dengan LinguSQL.

2.1 Spesifikasi

Dalam subbab ini akan dijelaskan mengenai pengertian dari spesifikasi, spesifikasi menggunakan Hoare Triple, serta spesifikasi dengan menggunakan Lingu.

2.1.1 Pengertian Spesifikasi

Spesifikasi perangkat lunak adalah hal-hal yang berkaitan tentang ketentuan-ketentuan serta batasan dari suatu perangkat lunak sehingga perangkat lunak tersebut dapat berjalan sesuai dengan apa yang diinginkan [SCH08]. Kegiatan spesifikasi seringkali dilakukan saat tahap-tahap awal pengembangan sistem ataupun pada pertengahan masa pengembangan sistem. Spesifikasi perangkat lunak harus dilakukan secara tepat dan teliti karena jika tidak dilakukan dengan tepat dan teliti maka perangkat lunak akan tidak sesuai dengan harapan. Selain itu verifikasi yang dilakukan setelahnya pun akan tidak berarti perannya terhadap kesesuaian tujuan perangkat lunak tersebut.

Spesifikasi sebuah perangkat lunak harus dijelaskan dalam suatu bahasa yang tidak ambigu [SUH07]. Suatu bahasa yang secara khusus didesain untuk penulisan spesifikasi program disebut *specification language* [PRA04]. Beberapa *specification language* yang ada sekarang ini antara lain; Hoare Triple, UML, Z, B, SDL, VDM, CCS, CSP, dll. *Specification language* yang akan dijelaskan terkait penelitian ini adalah Hoare Triple. Hoare Triple dipilih karena konsepnya akan digunakan pada bahasa spsifikasi Lingu. Penulisan spesifikasi pada Lingu akan mengikuti model Hoare Triple dengan menggunakan tata bahasa sendiri menurut aturan Lingu [SUH07]. Penjelasan mengenai Hoare Triple dan Lingu akan dijelaskan pada subbab-subbab selanjutnya.

2.1.2 Spesifikasi Menggunakan Hoare Triple

Spesifikasi menggunakan Hoare Triple adalah salah satu macam dari teknik spesifikasi yang menggunakan pendekatan *theorem proving* pada saat melakukan verifikasi terhadap spesifikasi perangkat lunak tersebut. Pembuatnya yaitu C.A.R Hoare (1969) menggunakan teknik kalkulus dalam menjelaskan kebenaran program jika ditinjau dari kondisi awal dan kondisi akhir dari suatu sistem [HOA69]. Secara sederhana, formula dari Hoare Triple dapat digambarkan sebagai berikut:

{PRE}P{POST}

Formula di atas dapat dijelaskan sebagai berikut, "Jika kondisi awal **PRE** terpenuhi saat program **P** akan dijalankan, maka kondisi akhir **POST** harus terpenuhi sebagai output dari program **P**".

Implementasi dari program atau fungsi tersebut dikatakan *partially correct* jika kondisi **PRE** terpenuhi saat sebelum program atau fungsi dijalankan, dan ketika program atau fungsi diasumsikan berhenti maka kondisi **POST** akan terpenuhi atau bernilai benar. Kemudian implementasi tersebut dikatakan *totally correct* jika kondisi **PRE** terpenuhi saat sebelum program atau fungsi dijalankan, dan ketika program atau fungsi itu dipastikan akan berhenti maka kondisi **POST** akan terpenuhi atau bernilai benar. Jadi *total correctness* adalah *partial correctness* ditambah dengan *termination* program atau fungsi [JOH88]. Suatu program dikatakan benar dan memenuhi spesifikasi jika memenuhi *total correctness.* Oleh karena itu, untuk membuktikan *total correctness* dapat dilakukan dengan cara membuktikan secara *partial correctness* dan membuktikan bahwa program atau fungsi pasti berhenti. Dalam melakukan pembuktian secara *partial correctness*, ada beberapa cara dan aturan yang dapat digunakan, di antaranya [BAC84] [HUN04] adalah *Consequence rule, Assignment axiom, Sequential composition, Conditional statement.* Gambaran dari semua aturan yang dipakai tersebut dapat dilihat pada Tabel 2.1 berikut ini.

Tabel 2.1 Aturan Pembuktian Pada Partial Correctness

No.	Nama Aturan	Aturan Pembuktian
1	Consequence Rule	a. PRE' => PRE {PRE}P{POST} ∴ {PRE'}P{POST} b. POST => POST' {PRE}P{POST} ∴ {PRE}P{POST'}
2	Assignment Axiom	∴ {POST[E/x]}x := E {POST(x)}
3	Sequential Composition	{PRE}P1{POST} {POST}P2{POST2} ∴ {PRE}P1;P2{POST2}
4	Conditional Statement	a. {PRE/\ *condition*}P{POST} {PRE/\ ¬*condition*} => {POST} ∴ {PRE}if *condition* then P{POST} b. {PRE/\ *condition*}P1{POST} {PRE/\ ¬*condition*}P2{POST} ∴ {PRE}if *condition* then P1 else P2{POST}

2.1.3 Spesifikasi Menggunakan Lingu

Lingu yang merupakan bahasa abstrak dan bahasa tingkat tinggi (*high level language*) serta didesain khusus pada domain aplikasi transaksi basis data [PRA05] adalah suatu jawaban atas kebutuhan tentang penggunaan metode formal sebagai alat verifikasi suatu perangkat lunak. Bahasa ini dikembangkan atas dasar susahnya penggunaan verifikasi formal ketika program dan spesifikasinya berkembang semakin besar dan kompleks.

Penulisan skrip pada Lingu didasari oleh model Hoare Triple dengan menggunakan tata bahasa tersendiri menurut aturan Lingu [SUH07]. Dalam perkembangannya sendiri, saat ini Lingu sudah memasuki versi yang kedua.

Pada Lingu v2, skrip Lingu dan skrip LinguHOL dijadikan satu setelah pada versi sebelumnya kedua skrip tersebut dipisahkan. Hal ini dikarenakan adanya kebutuhan untuk mempersingkat proses spesifikasi dan juga verifikasi. Skrip LinguHOL sendiri memiliki definisi yaitu skrip Lingu yang akan diverifikasi kebenarannya dengan menggunakan *theorem prover* HOL. Penjelasan skrip Lingu di sini akan ikut dijelaskan dalam penjelasan skrip LinguHOL. Hal ini dikarenakan skrip Lingu v2 merupakan skrip LinguHOL v2.

Dengan adanya versi yang terbaru ini proses verifikasi suatu perangkat lunak akan semakin singkat karena pengembang tidak perlu paham benar mengenai formula matematis dalam pembuatan skrip LinguHOL. Gambar 2.1 berikut ini akan menunjukan contoh penulisan skrip LinguHOL v2.

```
1  val credit_def = define
2
3    'credit (REF account, REF trans, t)
4        =
5        pre T
6        post ~(t IN trans)
7        do
8        /{ ins t trans
9         ; ins <| Nr     := t.Nr   ;
10                Desc   := t.Desc ;
11                Date   := t.Date ;
12                Debit  := 0      ;
13                Credit:= t.Ammount |>
14             account
15          ; delete trans (drop r. r.Nr = t.Nr)
16          /}
17          return void
18          ';
```

Gambar 2.1 Kode LinguHOL v2 [SUH07]

Pada gambar di atas dijelaskan bahwa penulisan skrip Lingu dimulai dengan suatu *header*, yaitu suatu deklarasi nama skrip yang diikuti parameter formal beserta tipenya. Contoh *header* tersebut bisa dilihat pada Gambar 2.2 berikut ini:

```
1  val credit_def = define
2
3    'credit (REF account, REF trans, t)
```

Gambar 2.2 *Header* dan Parameter Formal Skrip Credit pada LinguHOL v2

Sementara itu *syntax* dari header dan isi program pada Lingu v2 adalah sebagai berikut.

ProgDecl	→ ProgName (FormalParam , . . .) = ProgDeclRHS
ProgDeclRHS	→ **pre** (Assertion) → **post** (AssertionRet) → **do** /{ Instr /} **return** (Expr)

Gambar 2.3 *Syntax header* dan isi program skrip Lingu v2

Parameter formal yang dipakai dalam skrip Lingu v2 dapat berupa primitif (*Boolean, integer, string*) atau *record. Keyword* REF sebelum parameter formal menunjukan parameter *passed by reference.* Tanpa *keyword* tersebut, memiliki arti parameter memiliki tipe *pass by value. Passed by reference* [SUH07] parameter berarti nilai dari argument *method* pemanggil merupakan alamat memori argument tersebut. Sedangkan *pass by value* [SUH07] parameter adalah nilai dari argument *method* pemanggil merupakan hasil salinan dari nilai argument program yang dipanggil. *Syntax* dari parameter serta variabelnya akan dijelaskan dalam Gambar 2.4 berikut ini.

FormalParam	→ REF? Var
Var	→ Identifier | (Identifier : HOLtype)
ActualParam	→ REF Var \\ pass-by-reference | Expr \\ pass-by-value

Gambar 2.4 *Syntax* parameter serta variable pada skrip Lingu v2

Kemudian dalam mendefinisikan kondisi awal dan kondisi akhir pada Lingu v2, digunakan *assertion. Assertion* ini harus berupa HOL *term* yang berupa variabel, konstanta, fungsi, dan λ-abstraksi(\x.t) [NN05]. *Syntax* dari *assertion* dijelaskan pada gambar 2.5 dan notasi yang digunakan pada HOL dapat dilihat pada Tabel 2.2 berikut ini.

Assertion	→ HOLTerm \\ ret tidak diperbolehkan
AssertionRet	→ HOLTerm \\ ret diperbolehkan

Gambar 2.5 *syntax assertion* pada Lingu v2

Tabel 2.2 Notasi HOL

Terms of the HOL Logic			
Kind of term	***HOL notation***	***Standard Notation***	***Description***
Truth	T	T	True
Falsity	F	⊥	False
Negation	~*t*	¬ t	Not t
Disjunction	$t_1 \backslash / t_2$	$t_1 \vee t_2$	t_1 *or* t_2
Conjunction	$t_1 /\backslash t_2$	$t_1 \wedge t_2$	t_1 *and* t_2
Implication	$t_1 ==> t_2$	$t_1 \Rightarrow t_2$	t_1 *implies* t_2
Equality	$t_1 = t_2$	$t_1 = t_2$	t_1 *equals* t_2
∀-quantification	*!x.t*	∀*x.t*	For all x : t
∃-quantification	*?x.t*	∃*x.t*	For some x : t
ε-term	*@x.t*	ε*x.t*	An x such that : t
Conditional	If *t* then t_1 else t_2	(t ⟶ t_1, t_2)	If t then t_1 else t_2

Ekspresi pada Lingu v2 terbagi menjadi dua tipe yaitu ekspresi sederhana dan juga ekspresi tabel. Ekspresi tabel digunakan untuk *query* pada basis data. Operator yang tersedia untuk digunakan dapat dilihat pada lampiran. Berikut akan dijelaskan *syntax* dari ekspresi Lingu v2 pada gambar 2.6 berikut ini.

```
Expr        → SimpleExpr | TableExpr
SimpleExpr  → Constant | Var
              | SimplExpr.FieldName
              | UnOp SimpleExpr
              | SimpleExpr BinOp SimpleExpr
              | <| FieldName := SimplExpr, . . . , FieldName := SimplExpr |>
TableExpr   → empty
              | select Expr ( map Var . simpleExpr ) ( only Var . TabelExpr )
              | TableExpr union TableExpr
              | SimpleExpr IN Expr
              | ALLof Expr ( satisfy Var . Expr )
              | SOMEof Expr ( satisfy Var . Expr )
```

Gambar 2.6 Syntax Ekspresi Lingu v2

Syntax instruksi yang terdapat pada Lingu juga tergambarkan pada gambar 2.7 berikut ini.

```
Instr       → skip
              | Expr /: Expr
              | /{ Instr ; . . . /}
              | /@ ProgName (ActualParam , . . .)
              | Expr /@= ProgName (ActualParam , . . .)
              | if (Expr) then /{ Instr /} else /{ Instr /}
              | let
                    Var = Expr and
                    . . .
                    In /{ Instr /}
              | assert (Assertion)
              | ins SimpleExpr Expr
              | insert Expr Expr (map Var . SimpleExpr)(only Var . Expr)
              | del SimpleExpr Expr
              | delete Expr (drop Var , Expr)
              | update Expr (map Var . SimpleExpr)(only Var . Expr)
```

Gambar 2.7 *Syntax* Instruksi pada skrip Lingu v2

Perbedaan yang lain antara Lingu v1 dengan Lingu v2 adalah pada Lingu v2 sebuah program yang dipanggil tidak dapat digunakan untuk proses verifikasi [SUH07]. Lingu v2 menggunakan bahasa TEST, turunan dari $\mathcal{L}_0$ [PRA05] untuk membuat unit

testing guna evaluasi suatu fungsi. Penulisan validasi pada Lingu v2 juga menggunakan TEST sehingga pada Lingu v2 tidak terdapat skrip validasi. Untuk selanjutnya pengecekan kebenaran program dilakukan pada bagian *assert* dengan ditentukan dahulu kondisi awal dan kondisi akhir sesuai dengan ketentuan **assert** (pre ==> post). *Syntax* TEST dapat dilihat pada gambar 2.8.

Test-Suite	→ TEST Label? Test-Instr -- SUITE Label? /{Test-Suite; . . . ; Test-Suite /}
Label	→ /:: Identifier

Gambar 2.8 *Syntax* TEST pada Lingu v2

Sementara itu *syntax* untuk Text-Instr adalah Instr dengan ketentuan:

1. Boleh menggunakan pemanggilan **whitebox**

   Whitebox (ProgramCall)

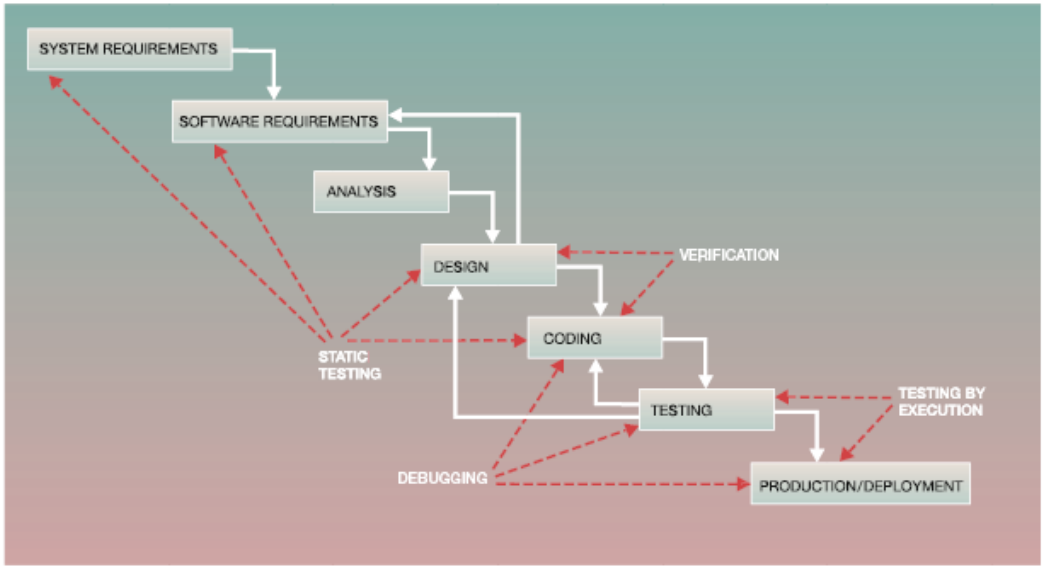
2. Untuk **assert** terbatas pada assert(Expr)

2.2 Verifikasi

Pada subbab ini akan dijelaskan tentang pengertian verifikasi, verifikasi dengan menggunakan metode formal, serta verifikasi dengan menggunakan HOL.

2.2.1 Pengertian Verifikasi

Verifikasi perangkat lunak adalah kegiatan untuk membuktikan atau menunjukan suatu perangkat lunak benar-benar bekerja menurut spesifikasi perangkat lunak tersebut yang sudah didefinisikan sebelumnya [HAI02]. Verifikasi pada beberapa metodologi perangkat lunak seperti *watervall, iterative, spiral,* dan lain-lain, biasanya dilakukan pada tahap *design* dan *coding*. Hal itu tergambar pada Gambar 2.9 berikut ini.



Gambar 2.9 Aktifitas Verifikasi pada beberapa Metodologi Pengembangan Perangkat Lunak [HAI02]

Dalam pelaksanaannya, sebuah proses verifikasi berjalan tergantung dari spesifikasi pada perangkat lunak yang ingin diperiksa. Jika spesifikasi perangkat lunak didefinisikan secara formal, maka verifikasi terhadap perangkat lunak tersebut harus dilakukan menggunakan metode formal [JOHA05] agar spesifikasi yang telah dibuat diterjemahkan menjadi formula matematis untuk selanjutnya dilakukan pembuktian [SUH07].

2.2.2 Verifikasi Menggunakan Metode Formal

Verifikasi dengan menggunakan metode formal sudah menjadi populer dimulai pada tahun 1994 [OUI08]. Ketika itu, proses verifikasi masih dijalankan secara konvensional atau hanya dengan *testing* saja. Pada tahun itu terjadi kasus yang terkenal dengan nama "Pentium Bug" yang mengakibatkan perusahaan besar sekelas Intel melakukan penarikan kembali produk chip mereka dan mengalami kerugian sebesar $475 juta. Dan sejak saat itu verifikasi terhadap perangkat keras dengan menggunakan metode formal mulai dilaksanakan oleh industri. Oleh karena keperluan yang serupa, verifikasi dengan menggunakan metode formal juga dilaksanakan terhadap perangkat lunak.

Secara umum, verifikasi dengan mengunakan metode formal ingin menggambarkan suatu perangkat lunak secara logikal yang menunjukan bahwa semua input yang diinginkan dalam kondisi awal **PRE** akan menghasilkan semua output seperti yang dijelaskan dan kondisi akhir **POST**.

Dalam pelaksanaanya, verifikasi dengan menggunakan metode formal lebih sulit untuk dilakukan. Hal ini dikarenakan perlunya usaha ekstra dalam melakukan spesifikasi sebelumnya ke dalam formula matematis. Walaupun begitu, idealnya jika verifikasi dengan menggunakan metode formal dapat menjamin kebenaran suatu perangkat lunak, maka para pengembang seharusnya tidak perlu lagi melakukan verifikasi dengan menggunakan teknik yang lainnya [ESA95]. Akan tetapi kesalahan pada manusia mungkin juga ada. Untuk mengantispasi hal tersebut perlu adanya tim independen yang memeriksa kembali proses verifikasi tersebut.

Verifikasi dengan menggunakan metode formal saat ini memiliki dua metode yang sudah cukup popular. Metode-metode itu ialah *model checking* dan *theorem proving*. *Model checking* awalnya digunakan untuk verifikasi pada perangkat keras, namun belakangan ini teknik ini juga diaplikasikan kepada perangkat lunak. *Model checking* cocok digunakan pada domain yang terbatas. Teknik ini tidak memerlukan spesifikasi yang berupa formula matematis. Hal ini yang menyebabkan teknik ini tidak bisa dijalankan pada perangkat lunak yang memiliki struktur data yang kompleks. Contoh alat yang berdasarkan teknik ini antara lain; SPIN, Bandera, JavaPathfinder, Verisoft,dll. Sementara itu teknik *theorem proving* seperti dijelaskan pada bab-bab sebelumnya bahwa teknik ini bekerja berdasarkan variasi dari teori Hoare *logic*. Sama seperti pada teknik *model checking,* teknik ini juga pada awalnya digunakan pada perangkat keras. Teknik ini cocok digunakan dalam domain yang tidak terbatas atau kompleks. Tidak seperti *model checking*, *theorem proving* tidak membutuhkan pengecekan properti dari tiap-tiap *state* dari program. Beberapa contoh alat yang berdasarkan teori ini antara lain; ACL2, Coq, HOL, Isabelle, dll. Secara garis besar, *theroem proving* sangat bagus digunakan pada kondisi program yang kompleks dan berskala besar. Sebaliknya dengan *model checking*, teknik ini akan efektif jika

dilakukan pada program yang tidak memiliki struktur data yang kompleks [OUI08]. Pada Lingu dan LinguSQL,metode *theorem proving* HOL digunakan. Hal ini dikarenakan Lingu dikhususkan untuk perangkat lunak basis data yang sangat krusial jika digunakan.

2.2.3 Verifikasi Menggunakan HOL

Sistem HOL adalah *theorem prover* yang berdasarkan logika tingkat tinggi (*higher-order logic*) [PIS99]. HOL dibangun di atas *Meta-Language* (ML) dan pertama kali diciptakan oleh Mike Gordon dalam versi pertamanya yaitu HOL88. Pada awalnya HOL digunakan khusus dalam melakukan verifikasi terhadap perangkat keras. Namun karena mengikuti perkembangan kebutuhan akan verifikasi, maka HOL dipakai juga dalam perangkat lunak. Adapun logika tingat tinggi (*higher-order logic*) merupakan versi dari predikat kalkulus dengan tiga keutamaan [SUH07] yaitu:

1. Variabel dapat berupa fungsi atau predikat
2. Logika memiliki tipe tersendiri
3. Tidak ada perbedaan secara sintak untuk formula (*term* bertipe **bool** memenuhi keseluruhan aturan).

Term pada logika HOL direpresentasikan dalam ML sebagai tipe abstrak yang disebut **term.** Term memiliki 4 tipe yaitu; variabel, konstanta, fungsi aplikasi ''t1 t2'', λ-abstraksi ''\x.t''. Term biasanya ditulis di antara dua kutip terbalik (''). Sebagai contoh, ekspresi ''x/\y==>z'' akan dievaluasi oleh ML sebagai x/\y => z. notasi yang digunakan dalam HOL dalam mendukung sistem logika dapat dilihat pada Tabel 2.2. Term pada HOL dapat juga dikonstruksikan dengan fungsi konstruktor ML. Dalam ML, fungsi **mk_var** mengkonstruksikan sebuah variable *string* dan sebuah tipe. Konstruktor **mk_conj** mengkonstruksi konjungsi dan konstruktor **mk_imp** mengkonstruksi sebuah implikasi.

Teorema direpresentasikan dalam HOL dengan suatu nilai yang memiliki tipe abstrak **thm**. Yang dilakukan untuk membentuk suatu teorema hanyalah dengan membangun suatu bukti. Pembuktian dalam HOL dilakukan dengan mengaplikasikan fungsi-

fungsi pada ML yang merepresentasikan aturan inferensi kepada aksioma-aksioma atau teorema-teorema sebelumnya. Pembuktian dalam HOL sangat fleksibel dan mendukung pembuktian maju (*forward proof*) dan juga pembuktian mundur (*backward proof*) dengan jalan membuat teorema dan menggunakan aturan inferensi terhadap teorema yang telah dibuat. Dalam pembuktian mundur (*backward proof*) pengguna harus mendefinisikan sekelompok teorema sebagai suatu *goal* [PIS99].

Bentuk umum [SUH07] dari teorema adalah t1, . . . , tn | - t dimana t1, . . . , tn adalah bentuk **term** Boolean yang disebut asumsi dan t adalah **term** Boolean yang disebut kesimpulan. Ada teorema yang menyatakan bahwa jika seluruh asumsi benar, maka kesimpulannya juga benar. Kondisi tersebut sama bentuknya dengan bentuk **term** tunggal tunggal (t1/\ . . . /\ tn) ==> t. Teorema tanpa asumsi dicetak dengan bentuk | - t.

HOL memiliki lima teorema dan delapan aturan inferensi primitif. Ketika sistem HOL dibangun, kedelapan aturan inferensi primitif telah didefiniskan dan kelima aksima terikat dengan nama ML-nya. Semua teorema yang lainnya dibuktikan dengan menggunakan aturan inferensi saat sistem dibuat. Ini menjadi penjelasan mengapa membangun HOL memakan waktu yang lama.