BAB 2 STUDI LITERATUR

Pada bagian studi literatur ini dijelaskan hal-hal yang dibutuhkan untuk memahami laporan penelitian. Selain itu, juga diberikan hasil studi literatur pada bagian ini.

2.1. Sistem HOL

Pada bagian ini dijelaskan definisi sistem HOL, perintah-perintah ML, *term-term* dari HOL, dan pembuktian berorientasi *goal* pada bagian ini. Berkaitan dengan kata formalisasi yang sering digunakan di dalam tulisan ini, formalisasi dalam sistem HOL mempunyai arti proses-proses yang dibutuhkan untuk mengubah sesuatu[1] yang informal menjadi sesuatu yang formal dalam sistem HOL. Formal berarti mengikuti aturan sintaksis yang telah ditetapkan sebelumnya dan bermakna tunggal (maknanya tidak ambigu).

2.1.1. Definisi Sistem HOL

Berdasarkan [2], dinyatakan bahwa "*automated theorem proving* merupakan pembuktian teorema matematika oleh program komputer". Pembuktian teorema matematika secara umum oleh program komputer yang sepenuhnya otomatis mungkin bisa gagal karena pencarian cara-cara pembuktian secara eksponensial melebihi batas kemampuannya. Oleh karena itu, program komputer yang melakukan pembuktian tidak sepenuhnya otomatis dan perlu tetap diberikan petunjuk-petunjuk oleh pengguna sistem supaya pencariannya lebih terarah. Program komputer yang semacam itu disebut dengan pembukti teorema yang interaktif (*interactive theorem prover*).

Salah satu kegunaan pembukti teorema di bidang industri adalah dalam desain sirkuit terintegrasi dan verifikasi. Sejak kesalahan Pentium FDIV, unit *floating point* yang kompleks dari mikroprosesor modern dirancang dengan lebih cermat [2]. Sejak prosesor-prosesor sekarang, pembukti teorema digunakan untuk memverifikasi hal tersebut dan operasi-operasi lainnya.

---

[1] Yang dimaksud dengan sesuatu dapat berupa definisi, aksioma, lemma, atau pun teorema.

Beberapa pembukti teorema yang dapat digunakan, misalnya HOL dan Isabelle. Sistem HOL adalah sebuah sistem pengembangan pembuktian yang ditujukan untuk aplikasi baik perangkat keras maupun perangkat lunak, dan mendukung logika bertingkat (*higher-order logic*) [3], serta dapat digunakan di antaranya untuk membuat definisi dan membuktikan teorema secara mekanis. Anggota-anggota dari keluarga sistem HOL, yaitu HOL88 dan HOL4 [4]. Dalam penelitian ini, digunakan HOL4 dan versinya adalah Kananaskis-4.

2.1.2. Perintah-Perintah ML

Sistem dalam keluarga HOL menggunakan bahasa pemrograman ML (Meta Language) atau pendahulunya sebagai bahasa pengantar untuk sistem pembukti teorema [4]. Perintah-perintah ML dapat ditulis secara interaktif atau pun melalui *script*. Beberapa perintah ML [1] yang digunakan dalam penelitian ini adalah sebagai berikut.

- `load` digunakan untuk membuka sebuah pustaka teori yang sudah ada berdasarkan namanya. Perintah ini hanya digunakan secara interaktif.
- `open` digunakan untuk membuka sekumpulan pustaka teori yang sudah ada berdasarkan namanya.
- `new_theory` digunakan untuk membuat sebuah teori baru dan memberikan nama kepadanya.
- `Hol_datatype` digunakan untuk membuat sebuah tipe data baru yang kemudian tipe data itu dapat dipakai di dalam *term* dari HOL.
- `Define` digunakan untuk membuat sebuah definisi baru.
- `new_recursive_definition` digunakan untuk membuat sebuah definisi baru yang bersifat rekursif. Terdapat tiga parameter yang diperlukan perintah ini, yaitu nama definisi, aksioma rekursif, dan isi dari definisi. Misalkan akan dibuat sebuah definisi rekursif terhadap sebuah daftar (*list*), aksioma rekursif yang sudah ada dan dapat digunakan adalah `list_Axiom`. Berdasarkan pengalaman penulis, nama definisi rekursif yang sudah ada tidak dapat ditimpa dengan nama definisi rekursif baru yang sama.

- `store_thm` digunakan untuk menyimpan sebuah teorema yang sudah dibuktikan kebenarannya. Terdapat tiga parameter yang diperlukan perintah ini, yaitu nama teorema, isi teorema, dan sekumpulan taktik yang digunakan untuk membuktikan kebenaran teorema.
- `g` digunakan untuk mendaftarkan sebuah *term* baru ke dalam *goalstack*. *Term* tersebut termasuk dalam *goal* awal dan kemudian harus dibuktikan. Perintah ini hanya digunakan secara interaktif.
- `e` digunakan untuk memberikan sekumpulan taktik yang dipakai untuk membuktikan kebenaran *term* yang berada di puncak *goalstack*. Perintah ini hanya digunakan secara interaktif.
- `b` digunakan untuk sekali membatalkan pembuktian tahap terakhir berdasarkan pemberian sekumpulan taktik lewat perintah `e`. Perintah ini hanya digunakan secara interaktif.
- `restart` digunakan untuk mengembalikan kondisi pembuktian *goal* sejauh ini ke kondisi awal *goal*. Perintah ini hanya digunakan secara interaktif.
- `status` digunakan untuk melihat *goal* awal dan *goal*/*subgoal* sejauh ini dari puncak *goalstack*.
- `export_theory` digunakan untuk mengekspor teori saat ini ke dalam sebuah berkas.

2.1.3. *Term-Term* dari HOL

Term dalam logika adalah kata-kata yang digunakan untuk menyatakan pernyataan logika secara formal. *Term-term* dari HOL dapat dilihat pada Tabel 2.1. *Term* dari HOL direpresentasikan di dalam ML oleh tipe abstrak (tipe umum yang dapat mempunyai tipe khusus) yang disebut `term`. *Term-term* tersebut biasanya diletakkan di dalam tanda *double back-quote* (``). Sebagai contoh, *term* ``` ``T`` ``` mempunyai tipe `term` di dalam ML (tipe umum) dan tipe `bool` di dalam HOL (tipe khusus).

Tabel 2.1 *Term-Term* dari HOL

Jenis Term	Notasi HOL	Notasi Standar	Deskripsi
Kebenaran	`T`	$\top$	benar
Ketidakbenaran	`F`	$\bot$	salah
Negasi	`~t`	$\neg t$	tidak t
Disjungsi	t_1 `\/` t_2	$t_1 \vee t_2$	t_1 atau t_2
Konjungsi	t_2 `/\` t_2	$t_1 \wedge t_2$	t_1 dan t_2
Implikasi	t_1 `==>` t_2	$t_1 \rightarrow t_2$	jika t_1, maka t_2
Kesamaan	t_1 `=` t_2	$t_1 = t_2$	t_1 sama dengan t_2
∀-Kuantifikasi	`!x. t`	$\forall x.\ t$	untuk setiap x: t
∃-Kuantifikasi	`?x. t`	$\exists x.\ t$	ada x: t
ε-term	`@x. t`	$\varepsilon\ x.\ t$	sebuah x sedemikian sehingga: t
Kondisi	`if t then` t_1 `else` t_2	$(t \rightarrow t_1, t_2)$	jika t maka t_1, selain itu t_2

2.1.4. Pembuktian Berorientasi *Goal*

Berdasarkan [7], dinyatakan bahwa "gaya dari pembuktian langsung[2] tidaklah alami dan terlalu 'bertingkat rendah' untuk banyak aplikasi". Oleh karena itu, pada awal tahun 1970, Robin Milner membuat sebuah metodologi baru untuk menghasilkan pembuktian dengan menggunakan Taktik [7]. Sebuah Taktik adalah sebuah fungsi yang melakukan dua hal, yaitu sebagai berikut.

- Memecah *goal* menjadi *subgoal-subgoal*.
- Menjaga jejak dari alasan mengapa menyelesaikan *subgoal-subgoal* akan menyelesaikan *goal*.

Beberapa Taktik yang ada antara lain sebagai berikut.

- `PROVE_TAC[]`

  Taktik ini digunakan untuk mencari pembuktian berdasarkan logika tingkat pertama (*first-order logic*) dengan kedalaman tertentu. Taktik ini akan menyelesaikan *goal* secara lengkap jika berhasil, sebaliknya gagal. Dengan menggunakan teorema-teorema yang tersedia dan asumsi-asumsi dari *goal*, Taktik ini melakukan pencarian untuk kemungkinan pembuktian-pembuktian

[2] Pembuktian langsung merupakan metode pembuktian menggunakan tabel kebenaran dengan mencoba semua kemungkinan untuk setiap peubah.

dari *goal*. Teorema-teorema yang digunakan, ditulis di dalam tanda kurung siku. Taktik ini berguna untuk menyelesaikan *goal* secara langsung.

- `RW_TAC[]`

  Taktik ini digunakan untuk menyederhanakan *goal* (menulisnya kembali dalam bentuk yang lebih sederhana) dengan aturan penyederhanaan yang diberikan dalam himpunan yang dinamakan `simpset`. Jika tidak berhasil menyederhanakan *goal*, maka *goal* akan tetap seperti semula. Beberapa `simpset` yang tersedia yaitu `arith_ss` (berhubungan dengan aritmatika), `bool_ss` (berhubungan dengan boolean), dan `std_ss` (standar).

- `REWRITE_TAC[]`

  Taktik ini digunakan untuk menyederhanakan *goal* dengan menuliskannya kembali dengan teorema-teorema yang diberikan secara eksplisit di dalam tanda kurung siku dan berbagai aturan penulisan kembali yang *built-in*.

- `Induct_on `x``

  Taktik ini menerapkan aturan induksi matematika terhadap *goal* dengan cara memecahnya menjadi kasus dasar dan kasus induksi. Induksi dilakukan terhadap peubah *x*.

- `Cases_on `x``

  Taktik ini digunakan untuk menganalisis kasus pada peubah *x*. Selain itu, dapat juga digunakan untuk menentukan peubah yang tersembunyi di dalam prefiks *quantifier* terhadap peubah *x*.

- `STRIP_TAC`

  Taktik ini digunakan untuk memecah sebuah *goal* dengan membuang salah satu penghubung terluar atau menjadikannya asumsi. Penghubung yang dimaksud adalah `!`, `/\`, `~`, atau `==>`.

- `WEAKEN_TAC funcBool`

  Taktik ini digunakan untuk menghapus asumsi pertama dari *goal* yang memenuhi fungsi `funcBool`. Fungsi itu menerima sebuah parameter berupa *term* dan mengeluarkan nilai *boolean*.

Dalam hal melakukan pembuktian menggunakan Taktik, sejauh ini belum ada algoritma yang pasti. Sering terdapat kesulitan untuk melakukan pembuktian

menggunakan Taktik, tetapi bukan berarti tidak bisa dibuktikan. Hal ini dikarenakan sistem HOL itu *sound* [6]. Definisi *sound* adalah jika terdapat cara untuk membuktikan kebenaran sebuah sekuen atau pernyataan yang ingin dibuktikan, maka hubungan *semantic entailment* antara premis dan kesimpulan dari sekuen itu terpenuhi [5].

Berdasarkan eksperimen-eksperimen yang dilakukan, beberapa pedoman yang dapat digunakan untuk melakukan pembuktian menggunakan Taktik adalah sebagai berikut.

- Pembuktian sebaiknya dilakukan secara bertahap dan dilakukan pada mode interaktif HOL terlebih dahulu. Sebaiknya jangan mencoba mencari pembuktian yang paling efisien pada awal pembuktian karena yang dicari adalah hanya cara-cara pembuktian, bukan pembuktian yang paling efisien. Setelah berhasil menemukan cara-cara pembuktian, barulah boleh menemukan pembuktian yang paling efisien.
- Cara-cara pembuktian sebaiknya ditulis terlebih dahulu pada tempat yang berbeda dengan jendela mode interaktif HOL. Setelah ditulis pada tempat yang berbeda, barulah cara yang dimaksud disalin ke jendela tersebut.
- Jika *goal* terlihat cukup sederhana, coba gunakan Taktik `PROVE_TAC[]`.
- Jika di dalam *goal* terdapat penggunaan-penggunaan definisi atau teorema yang sudah ada, coba gunakan `RW_TAC[]` atau `REWRITE_TAC[]` dengan definisi atau teorema yang dimaksud ditulis di dalam tanda kurung siku.
- Definisi dan teorema yang sudah ada dapat dikombinasikan di dalam tanda kurung siku Taktik-Taktik di atas. Terkadang penggunaan dua atau lebih Taktik yang sama dengan definisi atau teorema yang berbeda-beda dapat menghasilkan cara pembuktian yang berbeda, dibandingkan dengan jika semua definisi dan teorema yang dimaksud dikombinasikan sekaligus sehingga cukup digunakan Taktik sekali saja.
- Jika di dalam *goal* terdapat peubah, coba gunakan `Induct_on` atau `Cases_on` terhadap peubah itu. Taktik ini dapat membuat *subgoal* bertambah banyak, tetapi memberikan efek samping yang menarik yaitu *subgoal* yang harus dibuktikan semakin berbentuk sederhana atau lebih mudah untuk dibuktikan.

- Jika penggunaan beberapa Taktik selain `PROVE_TAC[]` sepertinya mengalami kebuntuan, coba gunakan `PROVE_TAC[]` terlebih dahulu walaupun *goal* yang terlihat tidak cukup sederhana. Terkadang cara ini bisa berhasil, tetapi sayangnya sering dilupakan karena *goal* yang terlihat tidak sederhana dan Taktik ini sering gagal melakukan pembuktian.
- Jika pembuktian-pembuktian *subgoal* salah atau mengalami kebuntuan, coba gunakan perintah `b`. Perintah ini dapat digunakan sekali atau beberapa kali. Jika pembuktian sudah cukup panjang dan perintah `b` tidak dapat digunakan lagi (terdapat batasan untuk kembali ke kondisi *goal* sebelumnya), gunakan perintah `restart`.
- Setelah berhasil membuktikan *goal*, sebaiknya coba buktikan sekali lagi dari awal. Hal ini bertujuan untuk menemukan cara-cara pembuktian yang lebih efisien dengan menggunakan lebih sedikit Taktik, untuk membuat suatu pemikiran yang lebih terarah, dan untuk membiasakan diri melakukan pembuktian menggunakan Taktik.
- Jika semua Taktik, definisi, dan teorema yang diketahui telah digunakan dan ditemui kebuntuan dalam pembuktian, cobalah untuk melihat kode sumber dari pustaka-pustaka terkait. Tidak ada salahnya mencoba semua definisi dan teorema yang terdapat di dalam pustaka terkait mulai dari awal sampai bukti ditemukan, atau setidaknya mendapatkan jalan keluar, walaupun nama definisi atau pun teorema di dalam pustaka tidak terlalu mencerminkan apa yang dimaksudnya. Selain itu, membaca buku-buku atau referensi-referensi (seperti [7] dan [8]) yang berisi contoh-contoh cara pembuktian menggunakan Taktik atau pun yang membahas detail pembuktian manual dari teorema bisa sangat membantu.

2.2. Rangkuman Teori *Graph*

Pada bagian ini diberikan definisi-definisi dan teorema dari teori graph yang dipilih dalam penelitian ini, serta aplikasi-aplikasi dari *graph*. Definisi-definisi dan teorema tersebut (termasuk penggolongan kategorinya) berasal dari [8] dan semuanya masih dalam bentuk informal.

2.2.1. Definisi-Definisi dari Teori *Graph*

Terdapat 13 buah definisi yang dipilih, yang dapat dibagi menjadi tiga kategori, yaitu pengenalan *graph*, terminologi *graph*, dan isomorfisme dari *graph*. Kategori pengenalan *graph* berisi definisi-definisi yang berhubungan dengan tipe-tipe *graph*. Kategori terminologi *graph* berisi definisi-definisi yang berhubungan dengan istilah-istilah umum dalam *graph*, misalnya derajat dari verteks. Kategori isomorfisme dari *graph* berisi sebuah definisi dari isomorfisme dua buah *graph*.

Nomor definisi dari teori *graph* di dalam laporan dan kode sumber mengikuti aturan "2.<nomor_kategori>.<nomor_definisi_dalam_kategori>" (untuk selanjutnya tanda kutip dua hanya untuk kejelasan). "<nomor_kategori>" merupakan sebuah bilangan bulat positif yang menyatakan nomor kategori. Kategori pengenalan *graph* memiliki nomor 1, kategori terminologi *graph* memiliki nomor 2, dan kategori isomorfisme dari *graph* memiliki nomor 3. "<nomor_definisi_dalam_kategori>" merupakan sebuah bilangan bulat positif yang menyatakan urutan definisi dalam kategori yang dimaksud. Sebagai contoh, nomor definisi 2.1.3 menyatakan definisi tersebut berada di dalam kategori pengenalan *graph* dan merupakan definisi ketiga dalam kategori itu.

Definisi 2.1.1 menyatakan bahwa sebuah *graph* sederhana $G = (V, E)$ terdiri dari V, sebuah himpunan tidak kosong dari verteks-verteks, dan E, sebuah daftar pasangan tidak berurut dari elemen-elemen berbeda dari V disebut sisi. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *graph* sederhana.

Definisi 2.1.2 menyatakan bahwa sebuah *multigraph* $G = (V, E)$ terdiri dari sebuah himpunan V dari verteks-verteks, sebuah daftar E dari sisi-sisi, dan sebuah fungsi f dari E ke $\{\{u, v\} \mid u, v \in V, u \neq v\}$. Sisi e_1 dan e_2 disebut sisi paralel jika $f(e_1) = f(e_2)$. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *multigraph*.

Definisi 2.1.3 menyatakan bahwa sebuah *pseudograph* $G = (V, E)$ terdiri dari sebuah himpunan V dari verteks-verteks, sebuah daftar E dari sisi-sisi, dan sebuah fungsi f dari E ke $\{\{u, v\} \mid u, v \in V\}$. Sebuah sisi disebut sebuah putaran

jika $f(e) = \{u, u\} = \{u\}$ untuk sebagian $u \in V$. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *pseudograph*.

Definisi 2.1.4 menyatakan bahwa sebuah *graph* berarah (V, E) terdiri dari sebuah himpunan verteks-verteks V dan sebuah daftar sisi-sisi E yang pasangan-pasangannya berurut dari elemen-elemen dari V. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *graph* berarah.

Definisi 2.1.5 menyatakan bahwa sebuah *multigraph* berarah $G = (V, E)$ terdiri dari sebuah himpunan V dari verteks-verteks, sebuah daftar E dari sisi-sisi, dan sebuah fungsi f dari E ke $\{\{u, v\} \mid u, v \in V\}$. Sisi-sisi e_1 dan e_2 merupakan sisi paralel jika $f(e_1) = f(e_2)$. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *multigraph* berarah.

Tabel 2.2 berikut merupakan tabel yang berisi rangkuman dari tipe-tipe graph yang ada dari [8].

Tabel 2.2 Tipe-Tipe *Graph*

Tipe	Sisi	Sisi jamak diperbolehkan?	Putaran diperbolehkan?
Graph sederhana	Tidak berarah	Tidak	Tidak
Multigraph	Tidak berarah	Ya	Tidak
Pseudograph	Tidak berarah	Ya	Ya
Graph berarah	Berarah	Tidak	Ya
Multigraph berarah	Berarah	Ya	Ya

Definisi 2.2.1 menyatakan bahwa dua buah verteks u dan v pada sebuah *graph* tidak berarah G disebut berdekatan (atau bertetangga) di G jika $\{u, v\}$ adalah sebuah sisi dari G. Jika $e = \{u, v\}$, sisi e disebut kejadian antara verteks u dan v. Sisi e juga dikatakan penghubung u dan v. Verteks u dan v disebut titik-titik akhir dari sisi $\{u, v\}$. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan ketetanggaan antara dua buah verteks pada sebuah *graph* tidak berarah.

Definisi 2.2.2 menyatakan bahwa derajat dari sebuah verteks pada sebuah *graph* tidak berarah adalah banyak sisi-sisi yang terjadi dengannya, kecuali bahwa sebuah putaran pada sebuah verteks memberikan dua kali ke derajat dari verteks

itu. Derajat dari verteks v dinotasikan oleh deg(v). Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan derajat dari sebuah verteks pada sebuah *graph* tidak berarah.

Definisi 2.2.3 menyatakan bahwa jika (u, v) adalah sebuah sisi dari *graph* G dengan sisi-sisi berarah, u dikatakan berdekatan ke v dan v dikatakan berdekatan dari u. Verteks u dikatakan verteks mula-mula dari (u, v), dan v dikatakan verteks pangkalan atau akhir dari (u, v). Verteks mula-mula dan verteks pangkalan dari sebuah putaran adalah sama. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan ketetanggaan antara dua buah verteks pada sebuah *graph* atau *multigraph* berarah.

Definisi 2.2.4 menyatakan bahwa pada sebuah *graph* dengan sisi-sisi berarah, derajat-dalam dari sebuah verteks v, dinotasikan oleh $\deg^-(v)$, merupakan banyak sisi dengan v sebagai verteks pangkalannya. Derajat-luar dari v, dinotasikan oleh $\deg^+(v)$, merupakan banyak sisi dengan v sebagai verteks mula-mulanya. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan derajat-dalam dan derajat-luar dari sebuah verteks pada sebuah *graph* atau *multigraph* berarah.

Definisi 2.2.5 menyatakan bahwa sebuah *graph* sederhana G disebut dua-pihak (*bipartite*) jika himpunan verteks V dapat dipartisi ke dalam dua himpunan saling lepas V_1 dan V_2 sedemikian sehingga setiap sisi pada *graph* menghubungkan sebuah verteks di V_1 dan sebuah verteks di V_2 (sehingga tidak ada sisi pada G menghubungkan dua verteks di V_1 atau dua verteks di V_2). Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan *graph* sederhana yang merupakan *graph* dua-pihak.

Definisi 2.2.6 menyatakan bahwa sebuah *graph*-bagian dari sebuah *graph* $G = (V, E)$ adalah *graph* $H = (W, F)$ di mana $W \subseteq V$ dan F subdaftar dari E. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *graph*-bagian.

Definisi 2.2.7 menyatakan bahwa gabungan dari dua *graph* sederhana $G_1 = (V_1, E_1)$ dan $G_2 = (V_2, E_2)$ adalah *graph* sederhana dengan himpunan verteks $V_1 \cup V_2$ dan daftar sisi $E_1 + E_2$. Gabungan G_1 dan G_2 dinotasikan oleh $G_1 \cup G_2$. Secara

umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan sebuah *graph*-gabungan.

Definisi 2.3.1 menyatakan bahwa *graph* sederhana $G_1 = (V_1, E_1)$ dan $G_2 = (V_2, E_2)$ adalah isomorfik jika terdapat sebuah fungsi f bersifat satu-satu dan pada dari V_1 ke V_2 dengan sifat bahwa a dan b berdekatan di G_1 jika dan hanya jika $f(a)$ dan $f(b)$ berdekatan di G_2, untuk setiap a dan b di V_1. Fungsi f tersebut disebut isomorfisme. Secara umum, definisi ini mengatakan hal-hal yang dibutuhkan untuk mendefinisikan isomorfisme dari *graph*.

Berdasarkan eksperimen pada subbab 3.3, ada beberapa definisi asli dari [8] yang perlu mengalami penyesuaian. Definisi-definisi tersebut adalah semua definisi dalam kategori pengenalan *graph*, Definisi 2.2.6, dan Definisi 2.2.7. Penyesuaian yang dilakukan adalah struktur *graph* yang menggunakan himpunan sisi diubah menjadi daftar sisi.

2.2.2. Teorema dari Teori *Graph*

Terdapat sebuah teorema yang dipilih, yang tergolong dalam kategori, terminologi *graph*. Kategori terminologi *graph* berisi definisi-definisi yang berhubungan dengan istilah-istilah umum dalam *graph*, misalnya derajat dari verteks.

Nomor teorema dari teori *graph* di dalam laporan dan kode sumber mengikuti aturan "2.<nomor_kategori>.<nomor_teorema_dalam_kategori>". "<nomor_kategori>" merupakan sebuah bilangan bulat positif yang menyatakan nomor kategori. Kategori terminologi *graph* memiliki nomor 2. "<nomor_teorema_dalam_kategori>" merupakan sebuah bilangan bulat positif yang menyatakan urutan teorema dalam kategori yang dimaksud. Sebagai contoh, nomor teorema 2.2.1 menyatakan teorema tersebut berada di dalam kategori terminologi *graph* dan merupakan teorema pertama dalam kategori itu.

Teorema 2.2.1 menyatakan bahwa misalkan $G = (V, E)$ merupakan sebuah *graph* tidak berarah dengan e sisi, maka

$$2e = \sum_{v \in V} \deg(v)$$

Teorema ini dikenal juga dengan nama teorema jabat tangan.

2.2.3. Aplikasi-Aplikasi dari *Graph*

Aplikasi-aplikasi dari *graph* [8] antara lain sebagai berikut.

- Menyelesaikan masalah terkenal jembatan Königsberg. Ide dasar penyelesaian masalah ini diperkenalkan pada abad kedelapan belas oleh Leonhard Euler, seorang matematikawan terkenal Swiss.
- Untuk menentukan apakah sebuah sirkuit dapat diimplementasikan pada sebuah papan sirkuit planar.
- Untuk membedakan antara dua buah campuran kimia dengan formula molekul yang sama, tetapi berbeda struktur.
- Untuk mempelajari struktur dari World Wide Web (WWW).
- Untuk mengetahui apakah dua buah komputer terhubung oleh sebuah jaringan komunikasi.