Pemanfaatan Graf dalam Membentuk *Grammar* pada Sistem Pengenalan Suara

Ivan Jonathan / 13516059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516059@std.stei.itb.ac.id

***Abstract*— Di era modern ini, tersebar banyak sekali aplikasi yang ada di desktop maupun perangkat *mobile.* Salah satu aplikasi yang sedang naik daun adalah asisten pribadi digital atau sering disebut sebagai *digital personal assistant.* Asisten pribadi bertugas untuk melakukan serangkaian tugas yang diperintahkan oleh pengguna dengan memanfaatkan sistem pengenalan suara atau sistem pengenalan dengan teks. Asisten pribadi dapat melakukan hal seperti menyetel musik, melakukan pencarian di peramban, menyetel alarm, membuat pengingat, atau bahkan asisten pribadi juga bisa diajak berbincang dengan kita dan masih banyak lagi yang lain yang dapat dilakukannya. Semua ini bisa dilakukan karena asisten pribadi ditenagai dengan Kecerdasan Buatan (*Artifical Intelligence*) dan khususnya dapat mengenali bahasa manusia sebagai bahasa alami. Pengenalan bahasa dapat dilakukan dengan ilmu dasar pada Matematika Diskrit yaitu Graf. Proses pengenalan bahasa dengan Graf ini dapat dimanfaatkan agar asisten pribadi mengerti apa yang dimaksud oleh kita sebagai manusia sehingga perintah tersebut dapat diolah dan dikerjakan oleh asisten pribadi. Salah satu contoh asisten pribadi yang terkenal saat ini adalah Siri buatan Apple, Amazon dengan Alexa-nya, Cortana oleh Microsoft, Google Assistant yang dirancang oleh Google, dan masih banyak lagi.**

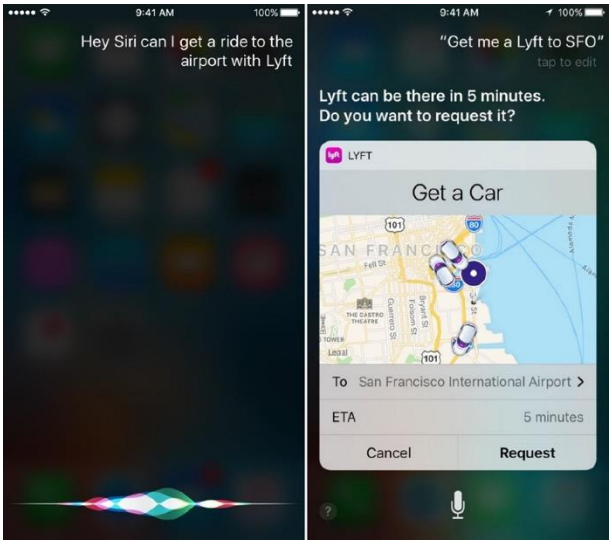
***Keywords*— *Grammar, Semantics, Syntax*, Graf, Asisten pribadi.**

I. PENDAHULUAN

Perkembangan teknologi yang sangat pesat membuat para perusahaan besar seperti Google, Apple, Microsoft, dan lainnya berlomba-lomba mengembangkan piranti lunak berbasis *A.I* agar dapat membantu pekerjaan manusia. Dengan memanfaatkan sumber daya yang mereka miliki, piranti lunak yang dikembangkan oleh perusahaan besar biasanya terintegrasi dengan basis data yang besar yang menyimpan berbagai informasi.

Informasi adalah salah satu kata penting dalam Ilmu Komputer. Asisten pribadi dapat diintegrasikan dengan basis data luar jaringan (*offline database*) atau dengan basis data dalam jaringan (*online database*) yang biasanya digunakan oleh perusahaan seperti Microsoft pada asisten pribadi *Cortana* milik mereka. Basis data ini memudahkan pencarian informasi oleh asisten pribadi untuk menampilkan informasi yang diminta oleh pengguna. Metode interaksi asisten pribadi dilakukan dengan berbagai cara yaitu pengenalan bahasa berbasis teks (*text-based recognition*) dan pengenalan bahasa berbasis suara (*voice-based recognition*).

Dengan memanfaatkan media yang ada maka bahasa itu dapat diperoleh tetapi pada makalah ini tidak dibahas bagaimana bahasa itu peroleh. Bahasa yang diperoleh tersebut diterjemahkan agar aplikasi dapat mengenali arti dari bahasa tersebut (*semantics)* walaupun terkadang *syntax* yang diperoleh berbeda. Sehingga, *semantics* yang diperoleh tersebut dikenali dan dimengerti arti dan maksudnya oleh suatu mesin.



Gambar 1 Salah satu aplikasi pengenalan suara yang memproses bahasa alami (https://cdn.macrumors.com/article-new/2016/04/sirithirdpartyintegration.jpg?retina)

Penulis berharap dengan adanya makalah ini, pembaca dapat memperoleh informasi dalam membentuk *Grammar* agar dapat mengenali bahasa alami dengan memanfaatkan ilmu Graf pada Matematika Diskrit agar dapat diterapkan ke dalam aplikasi sejenis.

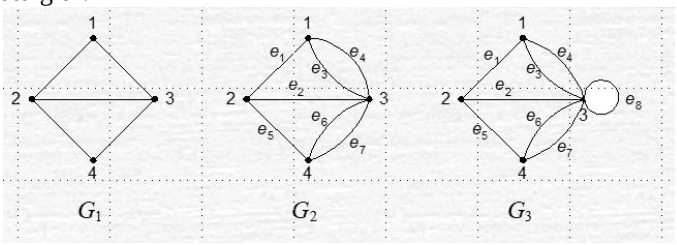
II. LANDASAN TEORI

2.1 Graf

Graf adalah pasangan himpunan yang terdiri atas simpul dan sisi ditulis dengan notasi G = (*V*,*E*) . Dalam hal ini *V* adalah himpunan yang tidak kosong dari simpul-simpul dan *E* adalah sisi-sisi dan himpunan yang boleh kosong.

Simpul pada graf dapat dinomori dengan huruf atau angka demikian pula dengan sisi yang juga dapat dinomori. Pelabelan dengan huruf atau angka dapat dianggap sebagai *state* pada

suatu graf.



Gambar 2.1 (a) G1 adalah graf sederhana, (b) G2 adalah graf ganda, dan (c) G3 adalah graf semu

Gambar 2 memperlihatkan 3 buah graf G1, G2, dan G3. G1 adalah graf dengan himpunan simpul *V* dan himpunan sisi *E* adalah

V = {1,2,3,4}

E = {(1,2), (1,3), (2,3), (2,4), (3,4)}

G2 adalah graf dengan sisi ganda dilihat dari sisi e3 dan sisi e4. Sisi e3 dan e4 dikatakan sisi ganda karena kedua sisi ini menghubungi dua buah simpul yang sama ,yaitu simpul 1 dan simpul 3.

G3 adalah graf dengan sisi gelang atau kalang karena ada simpul yang berawal dan berakhir pada simpul yang sama.

2.2 Jenis-Jenis Graf

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka graf digolonngkan menjadi dua jenis :

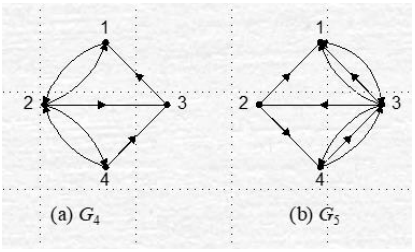
1. Graf sederhana (*simple graph*).
   Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana. G1 pada Gambar 2.1 adalah contoh graf sederhana.

2. Graf tak-sederhana (*unsimple-graph*).
   Graf yang mengandung sisi ganda atau gelang dinamakan graf tak-sederhana (unsimple graph). G2 dan G3 pada Gambar 2.1 adalah contoh graf tak-sederhana.

Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis :

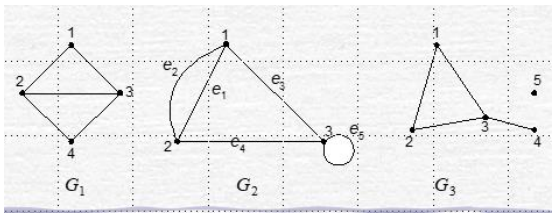
1. Graf tak-berarah (*undirected graph*)
   Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah. Tiga buah graf pada Gambar 2 adalah graf tak-berarah.

2. Graf berarah (*directed graph*)
   Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Dua buah graf pada Gambar 2.2 adalah graf berarah.



Gambar 2.2 (a).G4 adalah graf berarah (b).G5 adalah graf ganda berarah

2.3 TERMINOLOGI GRAF



Gambar 2.3 Terminologi Graf

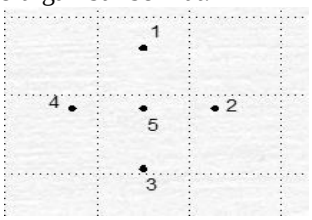
Berikut terminologi graf :

1. Ketetanggaan (*Adjacent*)
   Dua simpul dikatakan bertetangga jika keduanya terhubung dengan sisi. Tinjau graf G1 pada Gambar 2.3 simpul 1 bertetangga dengan 2 juga dengan simpul 3 karena dihubungkan dengan sisi. Perhatikan bila simpul 1 tidak bertetangga dengan simpul 4.
2. Bersisian (*Incident*)

Untuk sembarang sisi e = (vj, vk) dikatakan
e bersisian dengan simpul vj , atau
e bersisian dengan simpul vk

Tinjau graf G1: sisi (2, 3) bersisian dengan simpul 2 dan simpul 3, sisi (2, 4) bersisian dengan simpul 2 dan simpul 4, tetapi sisi (1, 2) tidak bersisian dengan simpul 4.

3. Simpul Terpencil (*Isolated Vertex*)
   Simpul terpencil ialah simpul yang tidak mempunyai sisi yang bersisian dengannya. Tinjau graf G3: simpul 5 adalah simpul terpencil.
4. Graf Kosong (*Null Graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong (Nn), seperti gambar berikut.



Gambar2.4 Graf Kosong

5. Derajat (*Degree*)
   Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.
   Notasi: d(v)
   Tinjau graf G1:
   d(1) = d(4) = 2
   d(2) = d(3) = 3
   Tinjau graf G3: d(5) = 0 -> simpul terpencil
   d(4) = 1 -> simpul anting-anting (pendant vertex)

Tinjau graf G2: d(1) = 3 -> bersisian dengan sisi ganda
d(2) = 4 -> bersisian dengan sisi gelang (loop).

6. Lintasan (*Path*)

Lintasan yang panjangnya n dari simpul awal v0 ke simpul tujuan vn di dalam graf G ialah barisan berselangseling simpul-simpul dan sisi-sisi yang berbentuk v0, e1, v1, e2, v2,... , vn –1, en, vn sedemikian sehingga e1 = (v0, v1), e2 = (v1, v2), ... , en = (vn-1, vn) adalah sisi-sisi dari graf G.

Tinjau graf G1: lintasan 1, 2, 4, 3 adalah lintasan dengan barisan sisi (1,2), (2,4), (4,3).

Panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Lintasan 1, 2, 4, 3 pada G1 memiliki panjang 3.

7. Siklus (*Cycle*)

Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus.

Tinjau graf G1: 1, 2, 3, 1 adalah sebuah sirkuit.

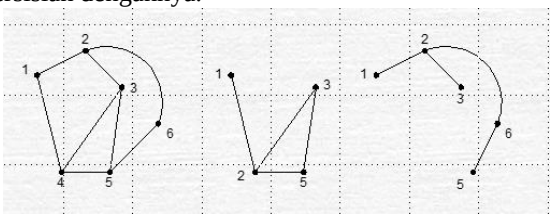
Panjang sirkuit adalah jumlah sisi dalam sirkuit tersebut. Sirkuit 1, 2, 3, 1 pada G1 memiliki panjang 3.

8. Terhubung (*Connected*)

Dua buah simpul v1 dan simpul v2 disebut terhubung jika terdapat lintasan dari v1 ke v2. G disebut graf terhubung (connected graph) jika untuk setiap pasang simpul vi dan vj dalam himpunan V terdapat lintasan dari vi ke vj. Jika tidak, maka G disebut graf tak-terhubung (disconnected graph).

9. Upagraf (*Subgraph*) dan Komplemen Graf

Misalkan G = (V, E) adalah sebuah graf. G1 = (V1, E1) adalah upagraf (subgraph) dari G jika V1 ⊆ V dan E1 ⊆ E. Komplemen dari upagraf G1 terhadap graf G adalah graf G2 = (V2, E2) sedemikian sehingga E2 = E - E1 dan V2 adalah himpunan simpul yang anggota-anggota E2 bersisian dengannya.



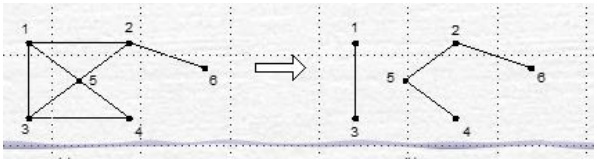
Gambar 2.5 Graf dan Upagraf

10. Upagraf Rentang (*Spanning Subgraph*)

Upagraf G1 = (V1, E1) dari G = (V, E) dikatakan upagraf rentang jika V1 =V (yaitu G1 mengandung semua simpul dari G).

11. Cut-set

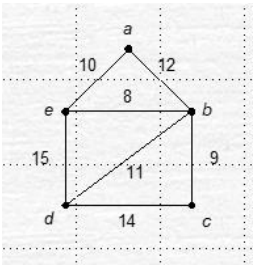
Cut-set dari graf terhubung G adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung. Jadi, cut-set selalu menghasilkan dua buah komponen. Pada graf di bawah, {(1,2), (1,5), (3,5), (3,4)} adalah cut-set. Terdapat banyak cut-set pada sebuah graf terhubung. Himpunan {(1,2), (2,5)} juga adalah cut-set, {(1,3), (1,5), (1,2)} adalah cut-set, {(2,6)} juga cut-set, tetapi {(1,2), (2,5), (4,5)} bukan cut-set sebab himpunan bagiannya, {(1,2), (2,5)} adalah cut-set.



Gambar 2.6 Cut-set

12. Graf berbobot (*Weighted Graph*)

Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga (bobot).



Gambar 2.7 Graf berbobot

2.4 Graf Khusus

Berikut beberapa graf khusus:

1. Graf Lengkap (*Complete Graph*)

Graf lengkap ialah graf sederhana yang setiap simpulnya mempunyai sisi ke semua simpul lainnya. Graf lengkap dengan n buah simpul dilambangkan dengan Kn. Jumlah sisi pada graf lengkap yang terdiri dari n buah simpul adalah n(n – 1)/2.

2. Graf Lingkaran (*Circle Graph*)

Graf lingkaran adalah graf sederhana yang setiap simpulnya berderajat dua. Graf lingkaran dengan n simpul dilambangkan dengan Cn.

3. Graf Teratur (*Regular Graphs*)

Graf yang setiap simpulnya mempunyai derajat yang sama disebut graf teratur. Apabila derajat setiap simpul adalah r, maka graf tersebut disebut sebagai graf teratur derajat r. Jumlah sisi pada graf teratur adalah nr/2.

III. GRAF DALAM MEMBENTUK *GRAMMAR*

A. Grammar

Grammar adalah suatu tata bahasa yang dilihat sebagai suatu aturan (*rule*) yang dapat menentukan apakah suatu bahasa tersebut dapat diterima atau tidak. Kita mulai dengan sebuah contoh misalkan ada lima buah produksi yaitu sebagai berikut :

S -> AB
A -> a
A -> Aa
B -> b
B -> bB

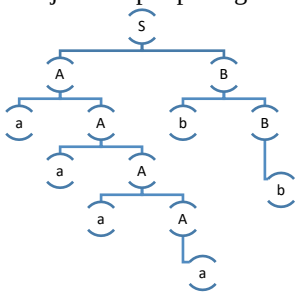
Disini, S, A , dan B adalah suatu peubah (*variable*). S adalah

start variable, sedangkan untuk huruf kecil a dan b adalah sebuah *terminal*. Perbedaan antara *terminal* dan *variable* adalah pada *variable* masih dapat diturunkan sedangkan pada *terminal* tidak. Misalkan kita ingin menurunkan sebuah string dari lima aturan diatas menjadi {a,b}*. Algoritmanya dimulai dari *start variable* yaitu S lalu lihat pada *rule* apakah S dapat diturunkan atau tidak , jika ya maka dapat diturunkan, sehingga pada akhirnya hanya tersisa *terminal* saja.

Sebagai contoh, *string* aaaaabb dapat diturunkan dengan cara

S -> AB
  -> aAB
-> aAbB
-> aaAbB
-> aaaAbB
-> aaaabB
-> aaaabb

Jika digambarkan dengan graf maka akan membentuk graf tak -berarah terhubung yang tidak punya sirkuit dan sering disebut sebagai pohon. Graf dibentuk dengan cara membuat *variable* dan/atau *terminal* menjadi simpul pada graf.



Gambar 3.1 Pohon untuk string aaaabb

Dari kesimpulan diatas didapat bahwa *Grammar* mempunyai 4 buah *tuple* dinotasikan sebagai G = (V, ∑, R, S).

1. V adalah sebuah himpunan terbatas yang anggota-anggotanya disebut sebagai *variables*.
2. ∑ adalah sebuah himpunan terbatas yang anggota-anggotanya disebut sebagai terminals.
3. S adalah *start variable* merupakan anggota dari V.
4. R adalah himpunan terbatas yang anggota-anggotanya disebut sebagai *rules*. Bentuknya adalah A -> w, A ∈ V dan w ∈ (V ∪ Σ)*.

Pada contoh diatas,

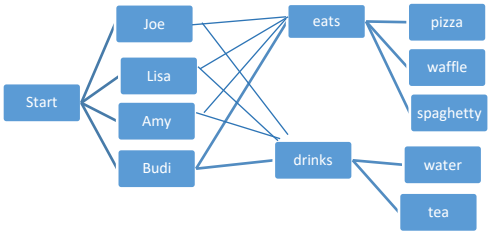
G = ({S,A,B}, {a,b}, {S -> AB, A -> a, A -> aA , B -> b, B -> bB}, S).

Teori *Grammar* ini dapat kita manfaatkan pada pengenalan ucapan , contoh kecilnya seperti “Joe eats pizza”, “Budi eats waffle” atau “I want to fly from Chicago to Miami”.

Pada contoh pertama dan kedua, kita dapat membuat kalimat tersebut menjadi lebih *general* contohnya dengan membuat aturan produksi. Misalnya

<Start> -> <Person><EatsOrDrinks>
<Person> -> “Joe” | “Budi” | “Amy” | “Lisa”
<EatsOrDrinks> -> “eats” <Food> | “drinks” <Drink>
<Food> -> “pizza” | “waffle” | “spaghetty”
<Drink> -> “water” | “tea”

Sehingga, jika *string* berupa “Joe drinks tea” diterima tetapi jika *string* berupa “Joe drinks pizza” itu tidak diterima. Lalu, jika digambarkan ke dalam sebuah graf maka akan terlihat seperti :



Gambar 3.2

Lalu, pada contoh ketiga “I want to fly from Chicago to Miami” juga dapat dibuat lebih *general* dengan produksi-produksi sebagai berikut :

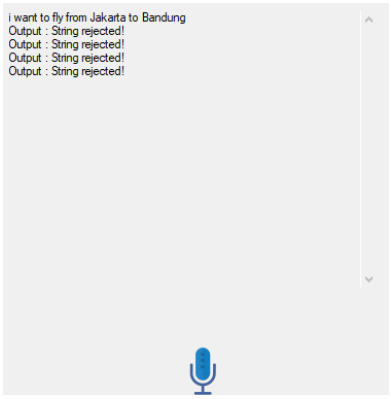
<Start> -> “I want to fly from” <From_To>
<From_To> -> <Destination> “to” <Destination>
<Destination> -> “Chicago” | “Jakarta” | “Miami” | “Bandung”

Sehingga, *string* diluar produksi diatas tidak akan diterima. Agar terlihat lebih “nyata” dan lebih aplikatif mari kita membuat *Grammar* pada file *XML (eXtensible Markup Languange).* Kita ambil contoh aturan misalnya contoh ketiga untuk dibuat ke dalam XML. Produksi-produksi diatas ekivalen dengan Gambar 3.3.

```
<?xml version="1.0" encoding="utf-8"?>
<grammar version="1.0" xml:lang="en-US"
         xmlns="http://www.w3.org/2001/06/grammar"
         tag-format="semantics/1.0" root="flights">

  <rule id="flights">
      i want to fly from
      <ruleref uri ="#from_to"/>
   </rule>

  <rule id="from_to">
      <ruleref uri="#destination"/>
      to
      <ruleref uri="#destination"/>
  </rule>

  <rule id="destination">
    <one-of>
      <item> Chicago </item>
      <item> Jakarta </item>
      <item> Bandung </item>
      <item> Miami </item>
  </one-of>
  </rule>
</grammar>
```

Gambar 3.3

Penulis telah membuat aplikasi untuk mengecek masukan berupa suara dengan *rules* diatas yang telah dimasukkan ke dalam file. Jika penulis berkata sesuatu selain aturan diatas maka akan muncul tulisan “Output : String rejected!” sedangkan bila penulis berkata sesuatu yang sesuai atau mengikuti *rules* diatas maka akan muncul tulisan yang barusan diucapkan. Misalnya, bila penulis berkata “I want to fly from Jakarta to Bandung” maka akan muncul tulisan seperti pada Gambar 3.4 sedangkan bila penulis berkata selain *rules* diatas maka akan muncul tulisan “Output : String rejected!” seperti pada Gambar 3.4 pada line kedua dan seterusnya.



Gambar 3.4 Tampilan aplikasi dengan menggunakan Grammar pada Gambar 3.3

B. Semantics

Sebelumnya, telah dibahas bagaimana pembentukan *Grammar* dengan menggunakan graf. Hal-hal yang kita lakukan tadi sebenarnya hanya membuat struktur dari sebuah ekspresi atau statement tetapi makna atau arti dari ekspresi tersebut masih belum ada sehingga disebut sebagai *Syntax*. Lalu, arti dari sebuah bahasa, ekspresi, atau statement inilah yang kita sebut sebagai *Semantics*. Contoh sederhananya untuk membuat *Semantics* misalnya kita punya *Grammar*

G = (V,T,P,S) dengan V,T,P,S sebagai berikut :

V = Himpunan *variable* yang terdiri atas {Operand1,Operand2, Operator, Start}.

T = Himpunan *terminal* yang terdiri atas {1,2,3,4,5,6,7,8,9,0,+,-,*,/}.

P = Himpunan produksi yang terdiri atas

<Start> -> <Operand1><Operator><Operand2>

<Operand1> -> 0|1|2|3|4|5|6|7|8|9

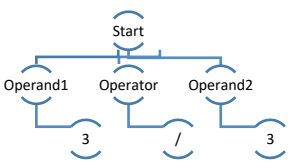
<Operand2> -> 0|1|2|3|4|5|6|7|8|9

<Operator> -> * | + | - | /

Jika dituliskan ke dalam file maka akan berbentuk seperti pada Gambar 3.5, karena kode terlalu panjang maka tidak akan ditampilkan semua. Lalu, bila digambarkan *Grammar* dengan graf maka akan terlihat seperti pada Gambar 3.6.

```
<?xml version="1.0" encoding="utf-8"?>
<grammar version="1.0" xml:lang="en-US"
         xmlns="http://www.w3.org/2001/06/grammar"
         tag-format="semantics/1.0" root="calculate">

  <rule id="calculate">
    <one-of>
      <item repeat="0-1">What is the answer of </item>
      <item repeat="0-1">How much is </item>
    </one-of>
    <ruleref uri ="#Number" type="application/srgs+xml"/>
    <tag>out.number1=rules.latest();</tag>
    <ruleref uri ="#Operator" type="application/srgs+xml"/>
    <tag>out.operator=rules.latest();</tag>
    <ruleref uri ="#Number" type="application/srgs+xml"/>
    <tag>out.number2=rules.latest();</tag>
  </rule>

  <rule id="Number">
    <one-of>
      <item>
        zero <tag>out = 0; </tag>
      </item>
      <item>
        one <tag>out = 1; </tag>
      </item>
      <item>
        two <tag>out = 2; </tag>
      </item>
      <item>
        three <tag>out = 3; </tag>
      </item>
      <item>
```

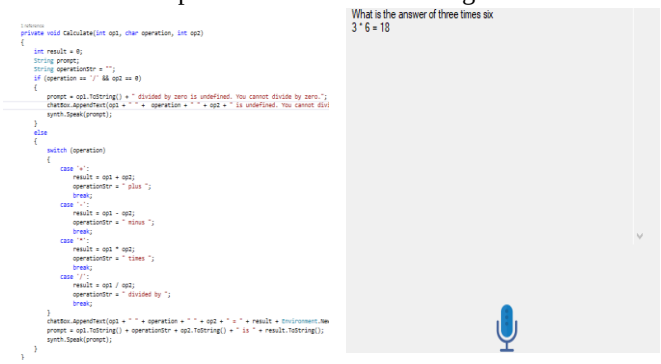
Gambar 3.5



Gambar 3.6 Misalkan string masukan 3 / 3

Bila diamati dengan seksama pada Gambar 3.5, masukan *string* yang diterima apabila menyebutkan kata “three plus five” maka akan diterima lalu untuk kata “What is the answer of” dan “How much is” adalah bentuk *closure* sehingga apabila itu tidak diujarkan oleh penulis maka masukan tetap diterima bila memenuhi produksi diatas. *Semantics* pada *Grammar* diatas bisa dilihat yaitu number1, operator, dan number2. Maksudnya adalah bila kita mengucapkan kata “Three plus five” maka secara kata “Three” akan diartikan sebagai number1, kata “plus” akan diartikan sebagai operator, lalu pada kata “five” akan diartikan sebagai number2 sehingga sekarang kita berhasil membentuk *Grammar* dengan membuat bagian-bagiannya menjadi sesuatu yang mempunyai arti.

Dengan begitu, bila kita gunakan *semantics value* pada number1, operator, dan number2 dapat kita manfaatkan untuk melakukan operasi matematis sehingga program komputer dapat menunjukkan hasil kalkulasi kepada pengguna. Dengan sedikit modifikasi penulis telah membuat fungsi untuk



Gambar 3.7 Fungsi untuk menghitung dari nilai semantik yang didapat yaitu number1, operator, dan number2

mengolah nilai semantik tersebut agar program dapat menghitung sesuatu yang diujarkan oleh pengguna.

Sehingga, saat penulis berkata “What is the answer of three multiplied by six” maka akan mengeluarkan output dan komputer merespon dengan jawaban seperti pada Gambar 3.7. Dengan memperoleh nilai semantik, program dapat mengerti maksud dari suatu kalimat yang diujarkan oleh pengguna. Lalu, pada contoh lain misalkan pada Gambar 3.3 juga dapat dibuat semantiknya misalnya memperoleh semantik daerah asal dan daerah tujuan. Sehingga, bila *string*-nya adalah “I want to fly from Jakarta to Bandung”, maka didapat nilai semantik untuk daerah asal adalah Jakarta dan nilai semantik daerah tujuan adalah Bandung.

Proses membentuk semantiknya mirip seperti pada contoh sebelumnya. Walaupun terlihat menguntungkan, tetapi sebenarnya mendefinisikan semantik adalah proses yang sulit karena terkadang terdapat kalimat yang ambigu.

IV. KESIMPULAN

Ilmu Graf pada Matematika Diskrit sangat bermanfaat di bidang *Computer Science*. Salah satunya ialah membentuk bahasa. Dengan membentuk bahasa, program dapat mengenali ucapan pengguna sehingga manfaatnya ialah manusia dapat berinteraksi dengan komputer via suara atau *gesture* lainnya. Manfaat ini mungkin belum terasa bagi kita dengan kondisi tubuh yang lengkap namun akan terasa sangat bermanfaat bagi

orang-orang yang mengalami masalah disabilitas fisik contohnya bila ada program yang dapat menerima bahasa "gerakkan mouse ke tepi kanan atas layar" lalu komputer menerima masukan tersebut lalu melakukan perintah dengan menggerakkan mouse pada layar komputer bergerak ke tepi kanan layar. Sehingga mereka masih bisa melakukan pekerjaan mereka dengan baik.

V. LAMPIRAN

Penulis juga sertakan lampiran berupa link dari aplikasi kecil yang telah penulis buat pada makalah ini

https://drive.google.com/drive/folders/1p1Ejpy5DMgsLcJjyWT_5dxRk9s72tPtF?usp=sharing

VI. UCAPAN TERIMAKASIH

Puji dan Syukur kepada Tuhan Yang Maha Esa, karena berkat dan rahmat-nya makalah ini dapat diselesaikan dengan baik. Penulis juga berterimakasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. , Bapak Drs. Judhi Santoso, M.Sc. , dan Ibu Dra. Harlili, M.Sc. selaku dosen pengajar IF2120 Matematika Diskrit atas segala bimbingan dan ilmu yang telah diberikan kepada penulis. Selain itu, penulis juga berterimakasih kepada sumber-sumber yang ada serta pihak yang telah membantu penulis dalam menyelesaikan makalah ini. Semoga makalah ini dapat berguna bagi pembaca.

REFERENCES

[1] https://en.wikipedia.org/wiki/Virtual_assistant_(artificial_intelligence)
Waktu akses : 1 Desember 2017 pukul 18.01 WIB.

[2] Rinaldi Munir, Diktat Kuliah IF2120: Matematika Diskrit. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2006.
Waktu akses : 1 Desember 2017 pukul 19.38 WIB.

[3] https://msdn.microsoft.com/en-us/library/system.speech.recognition.speechrecognizer.speechrecognized(v=vs.110).aspx
Waktu akses : 2 Desember 2017 pukul 12.23 WIB.

[4] https://msdn.microsoft.com/en-us/library/office/hh538497(v=office.14).aspx
Waktu akses : 2 Desember 2017 pukul 14.10 WIB

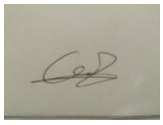
[5] Aniel, Michiel, Introduction to Theory of Computation. 2014.
Waktu akses : 2 Desember 2017 pukul 10:34 WIB.

[6] http://cs.umw.edu/~finlayson/class/spring13/cpsc401/notes/06-syntax-semantics.html
Waktu akses : 2 Desember 2017 pukul 13.30 WIB

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2017



Ivan Jonathan, 13516059