Aplikasi Pohon dalam Parsing Bahasa Pemrograman

Teresa - 13516133[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]teresachen14217@gmail.com

***Abstract*—Saat ini bahasa pemograman sudah banyak jumlahnya dan sudah banyak pula yang memakainya. Bahasa pemrograman tersebut akan digunakan untuk membuat program, namun sebelum kode yang telah dibuat oleh programmer di-compile, kode tersebut harus di-parsing menggunakan parser. Parser pada umumnya menerapkan pohon untuk menentukan apakah suatu bahasa adalah benar atau tidak. Terdapat banyak sekali algoritma yang digunakan pada parser, dan pada makalah ini bentuk representasi umum dari algoritma tersebut akan dibahas.**

***Keywords*—Representasi, Parser, Pemrograman, Pohon.**

I. PENDAHULUAN

Pemrograman adalah suatu proses untuk mengambil suatu algoritma dan mengubahnya menjadi program dengan membuat kode yang menggambarkan algoritma tersebut dengan suatu bahasa pemrograman [1]. Bahasa pemrograman terbagi menjadi tiga tingkatan sesuai dengan abstraksinya, yaitu bahasa mesin, bahasa *assembly*, dan bahasa tingkat tinggi. Bahasa mesin sesuai namanya adalah bahasa yang hanya dapat dimengerti oleh mesin itu sendiri. Bahasa mesin tersusun dari suatu rangkaian string yang hanya terdiri dari angka 0 dan 1 yang menandakan ada atau tidaknya arus listrik pada mesin, dan bahasa tersebut akan berbeda-beda tergantung dari mesinnya. Bahasa mesin ini sangat susah dipahami oleh manusia, dan dengan bahasa ini sangat susah untuk men-debug suatu program.

Lalu ada bahasa *assembly*, yaitu bahasa yang sudah terabstraksi dari bahasa mesin dengan menggunakan simbol-simbol, seperti ADD, LOAD, dan lain-lain. Namun, bahasa ini juga agak susah dimengerti manusia karena bahasa ini hanya mengabstraksi langkah demi langkah, sehingga kita tidak tahu algoritma apa yang dilakukannya secara keseluruhan. Yang terakhir adalah bahasa pemrograman tingkat tinggi, yaitu bahasa pemrograman yang diabstraksi sedemikian rupa sehingga menyerupai bahasa manusia, umumnya berbahasa inggris [2]. Bahasa pemrograman tingkat tinggi yang sudah ada saat ini sangat banyak, namun bahasa pemrograman tingkat-tinggi yang paling awal dibuat adalah bahasa FORTRAN yang dibuat pada tahun 1954.

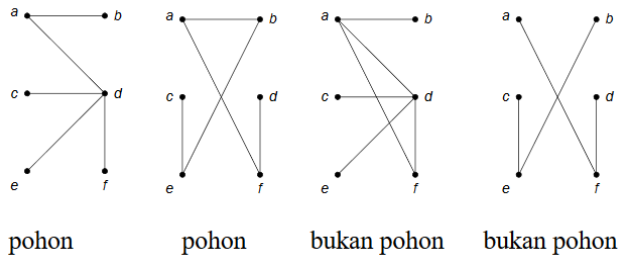
Untuk membuat suatu program dari suatu kode, pastilah membutuhkan *compiler*. *Compiler* akan mengubah kode program yang telah ditulis menjadi suatu program yang dapat dieksekusi oleh komputer. Namun, sebelum *compiler* membuart program yang dapat dieksekusi, *compiler* harus tahu apakah kode tersebut ditulis sesuai dengan tata bahasa dari suatu bahasa tersebut. Dalam ilmu komputer, tahap ini disebut tahap *parsing*.

Pada makalah ini penulis akan membahas bagaimana pohon diterapkan dalam algoritma *parsing*. Tujuan dari makalah ini adalah untuk membagi ilmu kepada pembaca terkait penggunaan pohon dalam algoritma *parsing* dalam pemrograman.

II. TEORI DASAR

A. Pohon

Pohon adalah graf yang khusus. Lebih rincinya lagi pohon adalah graf tak berarah terhubung yang tidak mempunyai sirkuit. Pada gambar berikut, graf ketiga adalah bukan pohon karena memiliki sirkuit dan graf keempat bukan pohon karena mereka saling lepas.



Gambar 1. Pohon dan bukan Pohon.
Sumber : http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).pdf. (Diakses pada tanggal 2 Desember 2017 pukul 09.25 WIB)

Menurut definisi graf, graf didefinisikan sebagai pasangan himpunan (V, E), dengan V adalah himpunan tidak kosong dari simpul pada graf dan E adalah himpunan dari sisi pada graf. Karena pohon adalah graf, maka suatu pohon boleh hanya memiliki satu simpul dan tidak memiliki sisi. Pohon yang didefinisikan di awal adalah pohon bebas.

Sifat-sifat pohon dinyatakan dalam teorema berikut.

Teorema. Misalkan G = (V, E) adalah graf tak berarah sederhana dan jumlah simpulnya *n*. Maka semua pernyataan di bawah ini adalah ekuivalen:

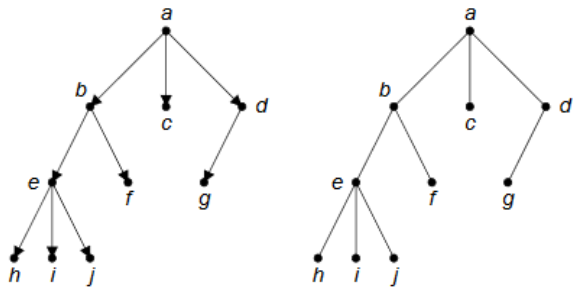
1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.

6. G terhubung dan semua sisinya adalah jebaran (jembatan adalah sisi yang bila dihapus menyebabkan graf terpecah menjadi dua komponen).

Terdapat juga beberapa sifat yang unik pada pohon, yaitu pada pewarnaan simpul pohon, pohon hanya memerlukan dua warna sehingga tidak ada dua simpul bertetangga yang berwarna sama. Lalu ada istilah pohon merentang, yaitu pohon yang didapatkan dari suatu graf G dengan menghilangkan sisi-sisi yang menyebabkan graf G memiliki sirkuit. Pohon merentang minimum adalah pohon merentang yang mempunyai jumlah bobot sisi minimum pada suatu graf G berbobot.

B. Pohon Berakar dan Pohon Terurut

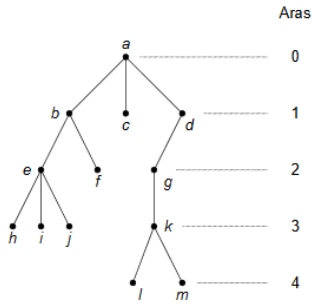
Pohon berakar adalah pohon yang sebuah simpulnya diperlakukan sebagai akar, dan sisi-sisinya diberi arah sehingga menjadi graf berarah. Simpul berupa akar memiliki derajat masuk nol dan simpul lainnya berderajat masuk satu. Diterapkan suatu konvensi sehingga akar selalu berada di atas dan semua sisinya mengarah ke bawah, sehingga arah dari sisi dihilangkan. Sedangkan pohon terurut adalah pohon yang urutan dari anak-anaknya penting, karena penelurusan dari simpul yang berbeda akan menghasilkan anak yang berbeda juga.



Gambar 2. Pohon berakar
Sumber : http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).pdf. (Diakses pada tanggal 2 Desember 2017 pukul 09.53 WIB)

Beberapa terminologi yang digunakan pada pohon berakar adalah sebagai berikut:

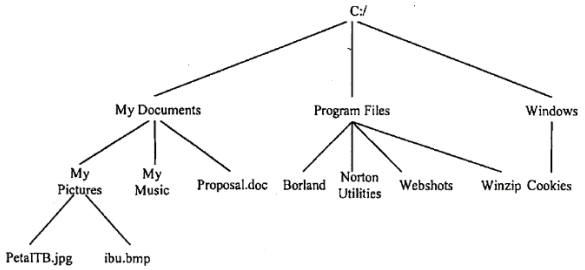
1. Anak : Suatu simpul *y* dikatakan sebagai anak dari simpul *x* bila ada suatu sisi berarah dari simpul *x* ke simpul *y*.
2. Orang Tua : Suatu simpul *y* dikatakan sebagai orang tua dari simpul *x* bila ada suatu sisi berarah dari simpul *y* ke simpul *x*.
3. Lintasan : Lintasan dari simpul v_1 ke v_k adalah runtutan dari simpul v_1, v_2, ..., v_k sedemikian rupa sehingga v_i adalah orang tua dari v_{i+1}.
4. Keturunan : Bila terdapat lintasan dari simpul *x* ke simpul *y* dalam pohon, maka simpul *y* adalah keturunan dari simpul *x*.
5. Leluhur : Bila terdapat lintasan dari simpul *x* ke simpul *y* dalam pohon, maka simpul *x* adalah leluhur dari simpul *y*.
6. Saudara kandung : Dua buah simpul adalah saudara kandung bila memiliki orang tua yang sama.
7. Upapohon : Upapohon pada pohon T dengan akar simpul x adalah upagraf T' = (V', E') dengan V' adalah simpul *x* beserta semua keturunannya dan E' adalan semua sisi pada semua lintasan yang berasal dari *x*.
8. Derajat : Pada pohon, derajat adalah jumlah upapohon pada suatu simpul.
9. Daun : Daun pada pohon adalah simpul yang tidak memiliki anak.
10. Simpul dalam : Simpul pada pohon yang memiliki anak.
11. Aras/Tingkat : Pada akar, arasnya adalah nol, sedangkan pada simpul lainnya, arassnya adalah 1 + panjang lintasan dari akar ke simpul tersebut.
12. Tinggi : Tinggi adalah aras maksimum pada suatu pohon.



Gambar 3. Aras pada pohon
Sumber : http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).pdf. (Diakses pada tanggal 2 Desember 2017 pukul 10.09 WIB)

C. Pohon n-Ary

Pohon *n-ary* adalah pohon yang setiap simpulnya memiliki anak maksimal sebanyak *n*. Bila setiap simpul cabang memiliki *n* buah anak, maka pohon tersebut disebut sebagai pohon teratur atau penuh. Contoh aplikasi dari pohon *n-ary* ini adalah pohon direktori dalam pengarsipan komputer dan juga sebagai pohon *parsing* untuk merepresentasikan struktur bahasa.



Gambar 4. Pohon direktori pada OS *Windows*
Sumber : Buku Matematika Diskrit. 2003

D. Compiler

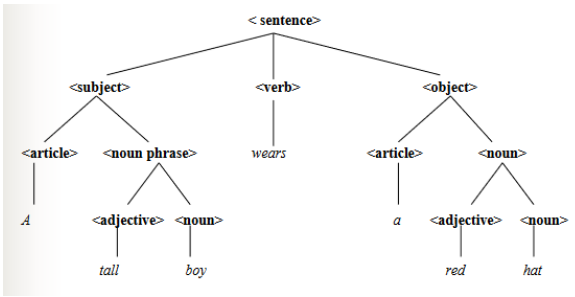
Compiler adalah suatu *software* yang mengubah program yang ditulis dalam bahasa tingkat tinggi ke dalam bahasa mesin, biasanya diubah dulu ke bahasa *assembly* baru diubah lagi ke bahasa mesin. Di dalam *compiler* terdapat dua tahap pengubahan, yaitu tahap analisis dan tahap sintesis. Di dalam tahap analisis terdapat beberapa tahap lagi, yaitu tahap analisis leksikal, analisis sintaks, dan analisis semantik. Pada tahap analisis leksikal, *compiler* akan membaca rangkaian karakter dan mengubahnya menjadi *token*, lalu pada bagian analisis sintaks, compiler mengecek apakah suatu kode tersebut ditulis dengan grammar yang benar menggunakan *token* hasil pengubahan dari analisis leksikal. *Token* tersebut digunakan

untuk membuat representasi grammar yang mirip dengan pohon. Tahap ini juga disebut dengan *parsing*. Pada tahap analisis semantik, pohon sintaks digunakan untuk mengecek kekonsistenan *source code* dalam semantic dengan definisi bahasa tersebut.

Lalu pada tahap sintesis, terdapat tiga anak tahapan, yaitu tahap pembuatan kode menengah, optimisasi, dan pembuatan kode (akhir). Pada tahap pembuatan kode menengah, setelah dilakukan analisis secara sintaks dan semantik, *compiler* akan menghasilkan representasi tingkat menengah ata lebih rendah dari *source code* tersebut. Representasi tersebut harus memiliki dua sifat, yaitu mudah dihasilkan dan juga mudah diterjemahkan ke mesin tujuan. Pada tahap optimisasi, *compiler* akan mencoba untuk mengoptimisasi kode tingkat menengah supaya kode target yang dihasilkan menjadi lebih baik, seperti lebih cepat atau kodenya lebih pendek. Pada tahap pembuatan kode akhir, *compiler* akan mengubah kode menengah tersebut menjadi kode dengan bahasa tujuan pada mesin [3].

D. Parsing

Parsing adalah suatu tindakan untuk menganalisis sebuah kalimat berdasarkan *grammar*-nya. *Parsing* biasanya digunakan untuk mengetahui apakah suatu kalimat memiliki tata bahasa yang baik.



Gambar 5. Pohon yang menyatakan struktur kalimat dalam bahasa Inggris
Sumber : http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).pdf. (Diakses pada tanggal 2 Desember 2017 pukul 10.15 WIB)

Menurut referensi [6], terdapat dua jenis *parsing*, yaitu:

1. *Top-down Parsing* : *Parser* berusaha untuk membuat pohon *parse* dari input yang diberikan mulai dari simbol *Start*.
2. *Bottom-up Parsing* : Berawal dari input, *Parser* akan mencoba untuk mengubah input tersebut menjadi pohon *parse* sampai akarnya adalah simbol *Start*.

Metode *Top-down parsing* dipakai pada *Recursive-Descent Parsing* dan LL *Parsing*. Untuk *Recursive-Descent Parsing* biasanya perlu *backtracking*, artinya *parser* terkadang perlu untuk membaca ulang input yang diberikan, namun sangat jarang. *Predictive Parser* adalah *Recursive-Descent Parsing* yang memprediksi produksi apa yang dihasilkannya, sehingga tidak perlu melakukan *backtracking*.

Pada metode *Bottom-up parsing* digunakan *shift-reduce parsing*, yaitu dua langkah khusus yang digunakan, yaitu langkah *shift* dan langkah *reduce*. Pada langkah *shift*, penunjuk pada simbol input akan dimajukan sekali, dan simbol yang sudah dipindah penunjuknya akan dimasukkan sebagai simpul dalam pohon *parse*. Pada langkah *reduce*, bila terdapat tata bahasa yang sudah lengkap pada pohon-pohon *parse* yang paling baru, maka pohon-pohon *parse* tersebut akan disatukan sebagai satu pohon *parse* dengan simbol akar yang baru sesuai dengan tata bahasa yang digunakan.

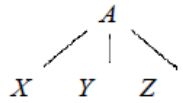
E. Context-Free Grammar

Menurut referensi [7], *Context-Free Grammar* adalah suatu tata bahasa yang dinotasikan dengan G = (V, T, P, S) dimana:

1. V adalah variabel atau nonterminal, setiap variabel merepresentasikan bahasa, atau serangkaian dari terminal.
2. T adalah terminal, yaitu simbol-simbol terdefinisi dalam suatu bahasa.
3. S adalah simbol *Start*, dimana semua CFG berawal.
4. P adalah aturan produksi dalam tata bahasa tersebut. Setiap aturan produksi memiliki:
   a. Variabel yang terdefinisi, dimana variabel ini akan disubstitusikan
   b. Tanda panah →
   c. Serangkaian dari nol atau lebih terminal dan variabel.

F. Pohon Parse

Pohon *parse* adalah pohon yang menunjukkan bagaimana dari simbol *Start* dapat diperoleh rangkaian huruf dari suatu bahasa. Umumnya pohon *parse* ini dibuat dari *Context-Free Grammar*. Contoh dari pohon *parse* adalah bila pada suatu tata bahasa terdapat aturan produksi A → XYZ, maka pohon *parse* dari aturan produksi tersebut akan memiliki simpul dalam A dan tiga anak X, Y, dan Z.



Gambar 6. Pohon *parse* dari aturan A → XYZ
Sumber : Buku *Compilers Principles, Techniques, & Tools*. 2007

Secara formal, pohon *parse* yang terbentuk dari *Context-Free Grammar* memiliki sifat-sifat berikut:

1. Akar dari pohon *parse* diberi simbol *Start*.
2. Setiap daun pada pohon diberi terminal atau e.
3. Setiap simpul dalam pada pohon diberi variabel/non terminal
4. Bila A, suatu nonterminal menandakan sebuah simbol dalam, dan X1, X2, ..., X adalah anak dari simpul tersebut dari kiri ke kanan, maka pasti ada suatu aturan produksi A → X1X2...X dimana X1, X2, ... X masing-masing adalah terminal atau nonterminal. Untuk kasus khusus produksi A → e, maka suatu simpul A hanya mempunyai satu anak e.

IV. PENERAPAN POHON DALAM PARSER

Pada *Parser* terdapat dua metode, yaitu metode *Top-down parsing* dan metode *Bottom-up parsing*. Kedua metode ini membutuhkan pohon sebagai representasinya. Sebagai contoh,

kita akan menggunakan tata bahasa yang dibuat oleh Irie Pascal Tools, sebuah *compiler* bahasa Pascal. Tata bahasa ini sudah dibuat dalam bentuk *Context-Free Grammar*.

The start symbol for this grammar is **program**.

```
actual-parameter = expression | variable-access |
    procedure-identifier | function-identifier

actual-parameter-list = '(' actual-parameter { ',' actual-parameter } ')'

adding-operator = '+' | '-' | 'or' | 'or_else' | 'xor'

array-type = 'array' [ index-type-list ']' 'of' component-type

array-variable = variable-access

assignment-statement = assignment-statement-lhs ':=' expression

assignment-statement-lhs = variable-access | function-identifier | property-designator

base-type = ordinal-type

binary-digit = '0' | '1'

binary-digit-sequence = binary-digit { binary-digit }

binary-integer = '%' binary-digit-sequence

block = declarative-part statement-part

boolean-expression = expression

buffer-variable = file-variable '^' | file-variable '@'

c-string-type = 'cstring' [ max-string-length ]

case-body = case-list-elements [ [ ';' ] case-statement-completer ] |
                case-statement-completer

case-constant = ordinal-constant

case-constant-list = case-specifier { ',' case-specifier }

case-index = ordinal-expression

case-list-element = case-constant-list ':' statement

case-list-elements = case-list-element { ';' case-list-element }

case-specifier = case-constant [ '..' case-constant ]

case-statement = 'case' case-index case-body [ ';' ] 'end'

case-statement-completer = ( 'otherwise' | 'else' ) statement-sequence

character-code = digit-sequence

character-literal = ''' string-element-one ''' |
                    '"' string-element-two '"' |
                    '#' character-code

component-type = type-denoter

component-variable = indexed-variable | field-designator

compound-statement = 'begin' statement-sequence 'end'

conditional-statement = if-statement | case-statement

constant = [ sign ] integer-number |
           [ sign ] real-number |
           [ sign ] constant-identifier |
                    character-literal |
                    string-literal
```

Gambar 7. Sebagian kecil dari CFG bahasa Pascal dari Irie Pascal Tools
Sumber : http://www.irietools.com/iriepascal/linux/progref526.html#appendix_h_grammar. (Diakses pada tanggal 2 Desember 2017 pukul 22.15 WIB)

Beberapa hal yang perlu diperhatikan dalam notasi ini adalah pembuat CFG mengganti tanda panah dengan tanda sama dengan (=), notasi {} untuk menandakan bahwa produksi di bagian dalam notasi tersebut boleh tidak ada, notas () yang menandakan bahwa dalam notasi tersebut boleh dipilih salah satu, dan masih ada lagi. *Context-Free Grammar* ini dipakai oleh *compiler* Irie untuk melakukan *parsing* pada suatu kode, apakah tata bahasa pada kode tersebut sudah benar atau belum.

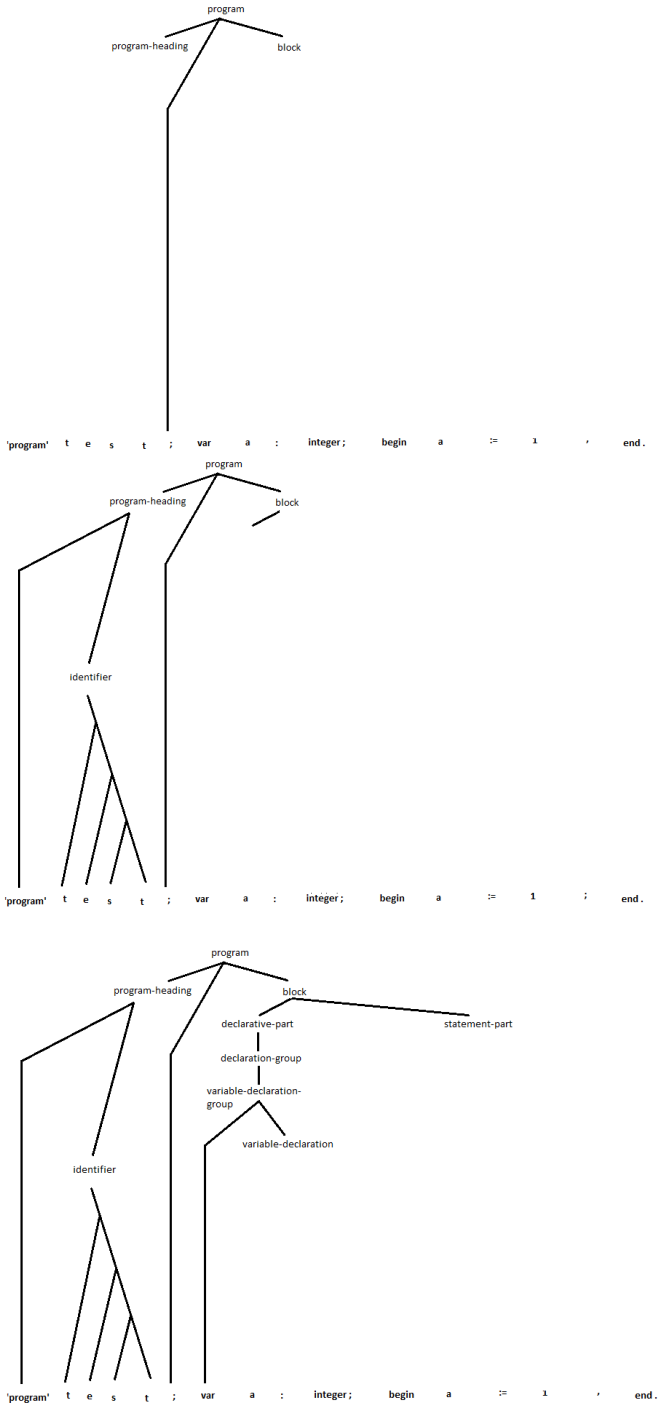
Ambil sebuah contoh kode program yang akan di-*parsing* adalah sebagai berikut:

```
1  program test;
2  var
3      a : integer;
4  begin
5      a := 1;
6  end.
```

Gambar 8. Contoh program kecil dalam bahasa Pascal
Sumber : Pribadi

A. Metode Top-down Parsing

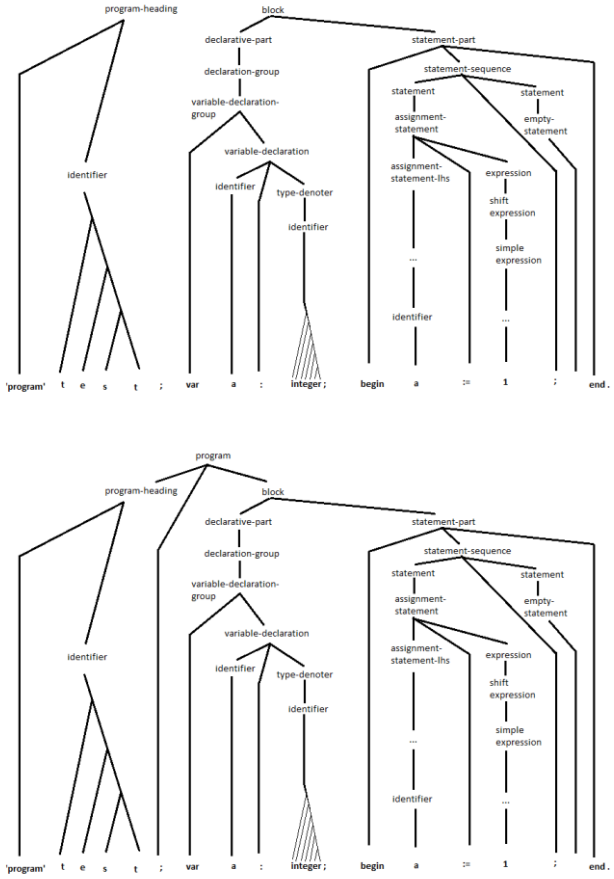
Pada grammar ini, simbol *Start*-nya adalah nonterminal <program>. Dari nonterminal program akan beranak menjadi 3 anak, yaitu <program-heading>, ';', dan <program-block>. Selanjutnya program akan beranak sesuai dengan tata bahasa dari Irie Pascal Tools ini. Namun, terdapat variabel yang akan menghasilkan satu variabel lainnya, sehingga perubahan yang seperti ini hanya memakan waktu dan langkah, sehingga pada pohon parse yang telah dibuat tidak dapat ditampilkan proses keseluruhannya karena terlalu panjang. *Compiler* akan mencoba semua kemungkinan yang terjadi pada aturan produksi sebelum mencapai terminal. Berikut adalah gambar urutan dan hasil pembuatan pohon *parse*-nya.



Gambar 9. Proses pembuatan pohon *parse* dengan metode *top-down*, Catatan: terdapat proses yang tidak bisa ditulis karena terlalu panjang
Sumber : Pribadi

B. Metode Bottom-up Parsing

Dengan metode ini, suatu pohon dapat dibuat dengan mengubah suatu terminal menjadi variabel yang menghasilkan terminal tersebut pada aturan produksinya. Akan dicoba semua kemungkinan, kemudian ketika terbentuk pohon-pohon terpisah yang dapat digabungkan, pohon-pohon tersebut akan digabungkan dengan akarnya yang baru adalah variabel yang dapat menghasilkan anak-anak tersebut. Pini akan menghasilkan pohon *parse* yang sama dan prosesnya terlalu panjang untuk digambarkan sebagai pohon. Berikut adalah gambar urutan dan hasil pembuatan pohon *parse*-nya.



Gambar 9. Proses pembuatan pohon *parse* dengan metode *bottom-up*, Catatan: terdapat proses yang tidak bisa ditulis karena terlalu panjang
Sumber : Pribadi

C. Bila Suatu Kode Tidak Sesuai Dengan Bahasa

Kondisi dimana suatu kode tidak sesuai dengan tata bahasa akan berbeda dengan metodenya. Suatu kode tidak akan sesuai pada metode *Top-down parsing* bila suatu input pada *parser* tidak memiliki langkah berikutnya. Suatu kode tidak akan sesuai pada metode *Bottom-up parsing* bila dari pohon *parse* yang telah terbentuk, dengan aturan produksi yang ada tidak akan pernah mencapai simbol *Start*.

D. Contoh Penggunaan Metode dalam Parser

Sebagai contoh, Hyacc dan LRSTAR adalah *parser generator* yang menggunakan algoritma LR, yang termasuk dalam metode *bottom-up* [7][8]. Sementara itu, ANTLR adalah *parser generator* yang menggunakan algoritma LL, yang termasuk dalam metode *top-down* [9]. Selain itu masih banyak lagi *parser generator* yang menggunakan kedua metode di atas.

V. KESIMPULAN

Penggunaan pohon pada representasi parsing sangatlah berguna untuk mengetahui apakah suatu kode masuk dalam tata bahasa yang digunakannya. Terdapat dua metode pembuatan pohon pada *parsing*, yaitu metode *top-down parsing* dan *bottom-up parsing*, yang memiliki capaian berbeda untuk mengetahui suatu kode masuk dalam tata bahasanya.

VI. UCAPAN TERIMA KASIH

Pertama, penulis berterima kasih dan mengucap syukur kepada Tuhan Yang Maha Esa karena berkat-Nya sehingga penulis dapat menyelesaikan makalah ini dengan baik. Selanjutnya penulis berterima kasih kepada Pak Rinaldi sebagai dosen mata kuliah Matematika Diskrit dalam mengajarkan materi bab pohon. Penulis juga berterima kasih kepada kakak-kakak tingkat atas karena makalah mereka menjadi acuan bagi penulis untuk menulis makalah dengan baik.

DAFTAR PUSTAKA

[1] http://interactivepython.org/runestone/static/pythonds/Introduction/WhatIsProgramming.html, Diakses pada tanggal 1 Desember 2017 pukul 23.20 WIB.
[2] Navabi, Faye (1997). *Chapter 2 Problem Solving* [Powerpoint slides]. Diambil dari http://www.cs.ccsu.edu/%7Emarkov/ccsu_courses/213Syllabus.html.
[3] Aho, Alfred V, Monica S. Lam, Ravi Sethi, dan Jeffrey D. Ullman. *Compilers Principles, Techniques, & Tools 2nd Ed*. London : Pearson, 2007, ch 1, 2, 4.
[4] https://www.pcmag.com/encyclopedia/term/40105/compiler, Diakses pada tanggal 2 Desember 2017 pukul 08.35 WIB.
[5] http://www.dictionary.com/browse/parse, Diakses pada tanggal 2 Desember 2017 pukul 10.37 WIB.
[6] https://www.tutorialspoint.com/compiler_design/compiler_design_types_of_parsing.htm, Diakses pada tanggal 2 Desember 2017 pukul 11.13 WIB.
[7] http://hyacc.sourceforge.net/, Diakses pada tanggal 3 Desember 2017 pukul 08.59 WIB.
[8] https://web.archive.org/web/20130710085600/http://lrstar.org/, Diakses pada tanggal 3 Desember 2017, pukul 09.09 WIB.
[9] Turon, Aaron, *SML.NJ Language Processing Tools: User Guide*, Chicago : University of Chicago, 2006, p.11.

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

Teresa, 13516133