Generator Pohon Untuk Grammar Berbasis Algoritma Couke Younger Kasami

Wijanarto
Fakutas Ilmu Komputer
Universitas Dian Nuswantoro
Semarang, Indonesia
wijanarto.udinus@gmail.com

Ajib Susanto
Fakutas Ilmu Komputer
Universitas Dian Nuswantoro
Semarang, Indonesia
ajibsusanto@gmail.com

***Abstract*—Syntactic analysis is a series of processes in order to validate a string that is received by a language. In this section it is often difficult to be explained properly, especially when making the rules up into a tree decline valid received by language. This study aims to produce an application that can generate information in the form of image files visual and textual representations of trees from the strings of the received language, a decrease in the rules of grammar, as well as the basic code of the image file that can be compiled and customization of its own. The algorithm chosen is CYK algorithm (Cocke Younger Kasami) for parsing the context free grammar (CFG) in the form of Chomsky Normal Form (CNF). CYK algorithm is very easy to understand and implement, therefore, appropriate given the decline in the teaching of automata to understand the rules to produce a valid tree of the input string. In the study of automata, understanding the process of reduction in the rules to be a tree (parse tree) is important to be taught well**

Kata kunci — generator; tree; cocke younger kasami; grammar; algorithm

I. PENDAHULUAN

Otomata dan teori bahasa dikenal sangat rumit dan lambat perkembangannnya, baik dari segi algoritma dan teori maupun implementasinya. Namun keberadaannya menjadi sangat sentral di dunia komputasi, karena tanpanya komputer tidak dapat berkembang seperti saat ini. Otomata dan teori bahasa sulit untuk dipahami secara manual karena algoritma dan pendekatan yang sangat rumit untuk dipelajari [1]. *Grammar* merupakan salah satu bagian dari ilmu otomata, yang implementasinya adalah bagaimana merepresentasikan secara linier dari suatu grammar atau dikenal dengan istilah *parsing*. Parsing merupakan bentuk representasi linier yang menghasilkan suatu pohon (*Parsing Tree*) [2][3][4]. Pemodelan bahasa ke dalam suatu bentuk matematis sudah di kenalkan oleh Chomsky [5]. Studi mengenai teknik parsing mulai dari awal, hingga yang modern baik dari teknik bottom up dan top down [4][6][7][8] selalu menarik untuk ditelaah maupun diimplementasikan. Visualisasi parser, dan implementasinya berbasis algoritma CYK [9][10][11] merupakan bukti dari betapa menariknya bahasan mengenai teknik parser. Cocke, Algoritma Younger dan Kasamai (CYK) [12][13][14] adalah suatu algoritma pemrograman dinamis dengan pendekatan *bottom-up*. yang merupakan salah satu pionir pengembang teknik parser yang masih sering diimplementasikan dan dikembangkan hingga saat ini. Berikut algoritma *Cocke-Younger-Kasami (CYK)* seperti ditunjukan pada Gambar 1 berikut.

```
input:
G=(N,Σ,S,→) dalam CNF, string w=a1..an∈ Σ+

CYK(G ,w )
1 for i=1…n do
2  Ti,i := {A∈N | A → ai}
3 for j=2…n do
4   for i=j-1…1 do
5    Ti,j:=∅;
5    for h=i…j-1 do
6     for all A → BC
7       if B ∈ Ti,h and C ∈ Th+1,j then
8          Ti,j:=Ti,j ∪ {A}
10 if S ∈ T1,n then return yes else return no
```

Gambar 1. Algoritma Cocke-Younger-Kasami

Algoritma *Cocke-Younger-Kasami (CYK)* memanfaatkan *Context-Free Grammar* (CFG) tipe 2 dari chosmsky hierarchy of grammar untuk format grammarnya dan menghindari rule string kosong, berikut Tabel 1 merupakan hirarki grammar menurut Chomsky [4],

TABEL 1. CHOMSKY-HIERARCHY OF GRAMMAR

Type	Description
0	recursively enumerable grammars or unrestricted grammars.
1	context-sensitive grammars.
2	context-free grammars.
3	regular grammars

Algoritma ini dapat menangani *left-recursion,* karena itu, *Context-Free Grammar* harus berada dalam bentuk *Chomsky Normal Form (CNF),* yaitu tidak boleh muncul string kosong, produksi unit, serta produksi *useles*. Berikut grammar G yang dimaksud seperti pada Gambar 2 dibawah ini,

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Gambar 2. Context-Free Grammar dalam Chomsky Normal Form

Dari grammar di atas untuk string *w=baaba* akan di hasilkan matrik atau table seperti yang ditunjukkan oleh Gambar 3,

{S, A, C}	← $X_{1,5}$			
Ø	{S, A, C}			
Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

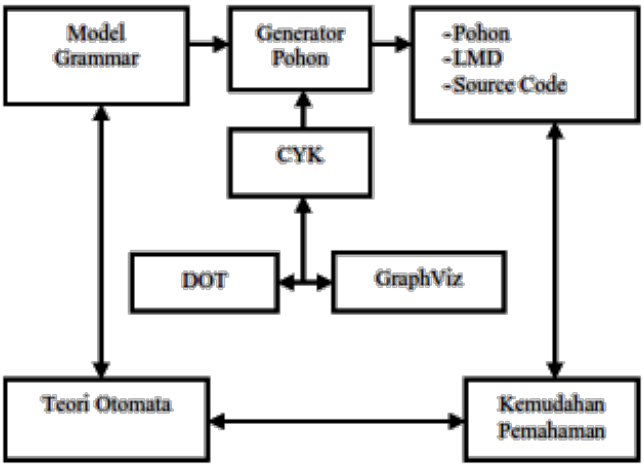
Gambar 3. Hasil akhir algoritma CYK

Hasil akhir dari paper ini adalah suatu prototype generator pohon dari context free grammar dalam bentuk Chomsky Normal Form.

II. MODEL DAN KERANGKA KERJA

A. Kerangka Kerja

Model dan kerangka pikir dalam paper ini seperti di tunjukan pada Gambar 3, adalah menentukan model grammar dan input yang di terima grammar tersebut. *Model Grammar* yang di tentukan merupakan representasi dari mata kuliah teori bahasa dan otomata, bagian parsing atau analisa sintaksis. Dalam hal ini penulis ingin mengimplementasikan model *Contex Free Grammar(CFG)* dalam bentuk *Chomsky Normal Form(CNF)* sebagai aturan dasarnya. Tentu saja diperlukan string *input* untuk menghasilkan pohon dari model yang di maksud yang akan di lakukan oleh *Generator Pohon*..

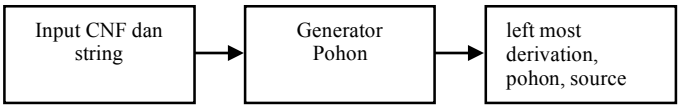


Gambar 4. Kerangka Kerja Generator Pohon

Generator Pohon merupakan inti atau kernel dari kerangka diatas, yang terdiri dari parsing berbasis algoritma *CYK* yang akan membaca rule berbentuk *CNF* dan akan memproduksi tabel turunan secara *left most derivation*, apakah string input dapat di terima oleh model atau tidak. Jika input dapat di terima oleh model dari hasil parsing dengan algoritma CYK, maka hasil penurunan tersebut di oleh menjadi pohon secara logic dalam 3 bentuk yaitu terurut dengan format prefix, infix dan postfix. Pohon yang di hasilkan akan di lakukan traversal dan di masukan ke dalam stack tiap kunjungan ke node. Data dari stack terebut di baca untuk di tulis dalam bahasa *DOT*, yaitu suatu domain specific language yang menghasilkan graph. Untuk merealisasikannya diperlukan antarmuka *GraphViz*, karena kerangka diatas akan diimplementasikan dengan java. *GraphViz* merupakan antar muka berbasis java untuk *DOT*, sedemikian rupa sehingga akan dihasilkan pohon dalam bentuk gambar (dengan format *PNG*). Hasil generator pohon ini diharapkan akan mempermudah pemahaman siswa terhadap ilmu yang diperoleh dalam kelas teori.

B. Penentuan Model Generator Pohon

Penentuan model standar grammar merupakan jantung dari paper ini, dikarenakan model ini merupakan kerangka utama dari aplikasi yang akan dihasilkan. Model grammar yang di pilih merupakan model Chomsky Norm Form (CNF) [15]. Secara umum arsitektur model grammar yang dipakai adalah seperti dalam Gambar 5 berikut.

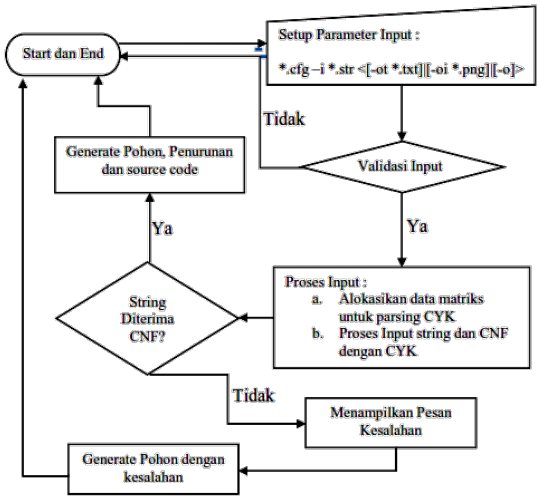


Gambar 5. Model Generator Pohon

Model terdiri dari 3 buah langkah yaitu Notasi CNF dan string input, yang berupa grammar (context free grammar) yang sudah dalam bentuk CNF serta string input yang dapat diterima oleh grammar, keduanya merupakan input yang akan diproses oleh generator pohon dan akan menghasilkan (men-generate) pohon, baik secara visual (image/*.png) maupun textual (*.txt) dan source code pohon untuk dapat dikustomisasi dalam format *.dot.

C. Rancangan Arsitektur

Rancangan arsitektur aplikasi dikembangkan dengan suatu metode atau cara yang beragam, penelitian ini akan menggunakan pendekatan dalam mengembangkan aplikasi yaitu Rapid Application Development (RAD). Adapun rancangan arsitektur secara umum sebagai kerangka pikirnya adalah seperti Gambar 6 sebagai berikut,



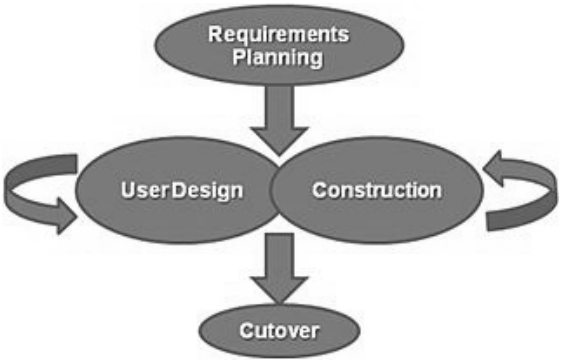
Gambar 6. Rancangan Arsitektur Aplikasi

Dari Gambar 6 dapat diterangkan, input yang berupa file grammar dalam format *Chomsky Norm Form* dan input string

(.cfg, *.str)* akan dibaca atau diparsing dengan algoritma CYK. Jika semua input tidak sesuai (validasi input) maka program selesai. Jika lolos validasi input maka langkah selanjutnya adalah mengalokasikann matrik atau table sesuai algoritma CYK. Setelah string input diperiksa dan diterima oleh grammar maka akan di-generate pohon, aturan penurunan dan source code untuk pohon visual. Jika input string tidak di terima oleh grammar, maka akan ditampilkan pesan kesalahan dan tetap akan di-generate pohon yang terdapat kesalahan tersebut.

D. Rapid Application Development

Teknik pembangunan sistem yang digunakan memanfaatkan pendekatan object oriented programming, dengan teknik Rapid Application Development (RAD). Disamping karena kemudahannya, teknik ini juga sangat cepat dalam membangun sistem skala menengah ke atas.



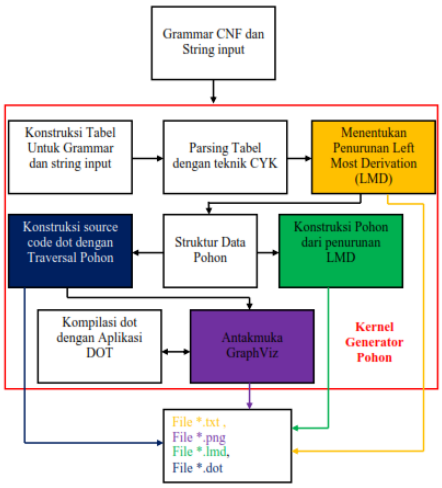
Gambar 7. Fase Rapid Application Development

Fase pengembangan sistem dengan metode RAD dibagi menjadi: (1) ***Fase Planning***, untuk menentukan tujuan, fungsionalitas dan scope yang akan dikerjakan, (2) ***Fase User Design***, yaitu menentukan interface dan bagaimana system akan bekerja dalam bentuk prototype, (3) ***Fase Construction***, Prototype di konversi menjadi aplikasi yang sudah berfungsi, dengan pengkodea dan pengembangan fungsionalitas aplikasi, (4) ***Fase Cutover***, merupakan fase terakhir dimana kegiatan utamanya adalah mencobakan pada pemakai dan melatih pemakai [16].

III. IMPLEMENTASI DAN HASIL

A. Implementasi Model

Dari paparan model, arsitektur dan kerangka kerja, maka dihasilkan sebuah aplikasi generator pohon. Generator yang merupakan kernel dari model sebagai pemroses utama, dituangkan dalam arsitektur yang dirancang pada Gambar 6, yang tediri dari 5 blok yang saling berhubungan satu dengan yang lain, Aplikasi *DOT*, antarmuka *GraphViz*, Algoritma CYK, Struktur Data Pohon dan Koreksi Kesalahan Grammar. Keluaran dari model berupa 4 file yang terdiri dari file text (**.txt*) dan gambar (**.png*) untuk menyimpan pohon yang dihasilkan, file text (**.lmd*) untuk menyimpan penurunan pohon dalam susunan *left most derivation,* serta file source code dari file gambar (**.dot*) yang dapat dikustomisasi kemudian jika diperlukan. Berikut Gambar 8, yang merupakan *building block* kernel dari model generator pohon berada dalam kotak dengan outline warna merah.



Gambar 8. Kernel Model Generator Pohon

Grammar *G* dalam bentuk chomsky normal form serta suatu string *w* sebagai menjadi masukan bagi kernel generator (kotak warna merah). Berdasarkan grammar *G* dan string *w* akan dikonstruksi tabelnya, setelah itu untuk dilakukan parsing dengan teknik *CYK*. Hasil dari parsing berupa suatu *Left Most Derivatio*n (*LMD*) sesuai aturan dalam algoritma yang terpilih (kotak warna kuning). *LMD* ini merupakan hasil generator yang disimpan dalam struktur data untuk menghasilkan pohon dalam bentuk tekstual (kotak warna hijau). Dalam struktur data yang disimpan pula nanti akan di-generate suatu source code dalam bahasa dot (kotak warna biru). Sementara pohon dalam bentuk grafik baru di-generate dengan menggunakan aplikasi dot dan antarmuka *GraphViz*, setelah mendapatkan source code (kotak warna ungu). Dengan demikian kernel menghasilkan 4 file yaitu hasil penurunan grammar (*LMD*), pohon dalam bentuk tekstual dan grafik serta source code pembentuk pohon dalam bahasa dot. Semua bagian kernel model generator pohon diimplementasikan dengan bahasa pemrograman java dan aplikasi *DOT*.

B. Hasil

Implementasi dari model dan kerangka kerja diatas adalah jika terdapat suatu grammar *G* dengan 5 rule dalam Gambar 9, Rule *S* menghasilkan konkatenasi non terminal *B* dan *S* atau *D* dan *S* atau terminal *a*.

```
S -> BS | DS | a.
A -> b.
B -> SA.
C -> c.
D -> SC.
```

Gambar 9. Grammar CFG Dalam Chomsky Normal Form

Rule *A* menghasilkan terminal *b*, rule *B* menghasilkan konkatenasi non terminal *S* dan *A*. Rule *C* menghasilkan terminal *c* dan terakhir rule *D* menghasillkan konkatenasi non terminal *S* dan *C*. String input *w* untuk grammar *G* diatas adalah "*abaca"*. Dengan menggunakan teknik CYK maka akan didapatkan left most derivation seperti berikut,

```
S=> S
S=> DS
D=> SCS
S=> BSCS
B=> SASCS
S=> aASCS
A=> abSCS
S=> abaCS
C=> abacS
S=> abaca
```

Gambar 10. Left Most Derivation String *abaca*

Hasil konstruksi tabel disusun dalam struktur data akan membentuk urutan baik dalam *preorder*, *inorder* ataupun *postorder* seperti berikut,

```
Preorder:(S(D (S (B S(a) A(b)) S(a)) C(c)) S(a))
Inorder:((((a(S) B b(A)) S a(S)) D c(C)) S a(S))
Postorder:((((a(S) b(A)B) a(S)S) c(C) D) a(S) S)
```

Gambar 11. Pohon dalam preorder, inorder dan postorder

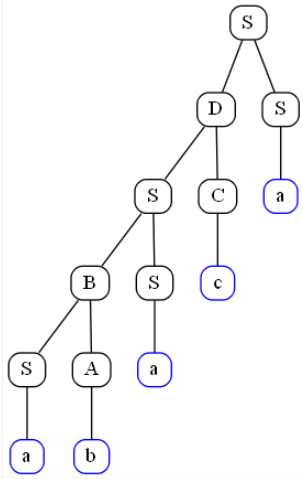
Jika dilakukan traversal pada pohon *preorder* diatas maka akan dapat dikonstruksi suatu source code dalam bahasa DOT seperti ditunjukkan Gambar 12.

```
graph G {
node0[label=S,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node1[label=D,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node2[label=S,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node3[label=B,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node4[label=S,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node5[label=a,shape=box,style=rounded,height=0.02,
width=0.01,color=blue];
node6[label=A,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node7[label=b,shape=box,style=rounded,height=0.02,
width=0.01,color=blue];
node8[label=S,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node9[label=a,shape=box,style=rounded,height=0.02,
width=0.01,color=blue];
node10[label=C,shape=box,style=rounded,height=0.02,
width=0.01, color=black];
node11[label=c,shape=box,style=rounded,height=0.02,
width=0.01,color=blue];
node12[label=S,shape=box,style=rounded,height=0.02,
width=0.01, color=black];

node0 -- node1;
node0 -- node12;
node1 -- node2;
node1 -- node10;
node2 -- node3;
node2 -- node8;
node3 -- node4;
node3 -- node6;
node4 -- node5;
node6 -- node7;
node8 -- node9;
node10 -- node11;
}
```

Gambar 12. Source Code DOT Dari Preorder

Berdasarkan source code diatas maka kita dengan mudah akan mengkonstruksin suatu tree dalam bentuk grafik dengan menggunakan aplikasi *dot* dengan antar muka *GraphViz* seperti Gambar 13 dibawah ini.



Gambar 13. Grafik Pohon String *abaca*

IV. KESIMPULAN

Dari paparan implementasi dan hasil terhadap model dan kerangka kerja yang dibuat, maka sementara dapat disimpulkan, bahwa generator dapat mempermudah visualisasi konsep suatu bahasa ke dalam bentuk baik tekstual maupun grafis, dalam hal ini adalah grammar *CFG* dalam bentuk *CNF* dengan algoritma *CYK* sehingga dapat divisualisasikan dengan tepat dan komunikatif mulai dari pembentukan grammar, string input, penurunan (*LMD*), pohon bahkan source code yang mudah dikustomisasi. Kedepan di harapkan dapat dibuat suatu framework yang lebih lengkap dengan fitur pilihan berbagai macam algoritma yang sejenis untuk mengenerate pohon.

DAFTAR PUSTAKA

[1] Wantah Satria, Sri Handayaningsih, 2013 , Pembuatan Media Pembelajaran Untuk Proses Konversi Pada Finate Automata Berbasis Multimedia, Jurnal Sarjana Teknik Informatika Volume 1 Nomor 1, Juni 2013 , e-ISSN: 2338-5197

[2] Alferd V Aho, Jeffery D Ullman, 1973, The Theory of Parsing, Translation and Compiling. New York : Prentice Hall Englewood Cliffs, 1973. 0-13-914564-8.

[3] Andrew W Appel, Maia Ginsburg, 1998, Modern Compiler Implementation In C. New York : CAMBRIDGE UNIVERSITY PRESS

[4] Dick Grune and Ceriel J.H. Jacobs, *Parsing Techniques - A Practical Guide*, 2nd ed., David Gries and Fred P. Schneider, Eds. New York, United States of America: Springer, 2008

[5] Alfred V Aho, Monica S Lam, Ravi Sethi , Jeffrey D Ullman, 2007, Compilers : principles, techniques, and tools Second Edition. New York : Pearson Education Addison Wesly

[6] Stephen A. Blythe, Michael C. James, and Susan H. Rodger, "LLparse and LRparse: Visual and Interactive Tools for Parsing," *Proceedings of the Twenty-fifth SIGCSE Technical Symposium on Computer Science Education*, pp. 208-212, 1994

[7] Terence Parr, 2010, Language Implementation Patterns Create Your Own Domain-Specific and General Programming Languages. Raleigh, North Carolina Dallas, Texas : The Pragmatic Bookshelf.

[8] Terence Parr, Kathleen Fisher, 2011, LL(*): the foundation of the ANTLR parser generator. s.l. : Vol 11 ACM SIGPLAN Notices - PLDI.

[9] Shamshad Ali, "CYK Algorithm," *International Journal of Scientific Research Engineering & Technology (IJSRET)* , vol. I, no. 5, pp. 1-4, August 2012

[10] Nathan Bodenstab, "Efficient Implementation Of The CKY Algorithm," Computational Linguistics, Final Project Paper 2009

[11] Martin Lange, Hans Leiß, 2009, To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm, *Informatica Didactica 8*

[12] John Cocke and J. T. Schwartz, 1970**,** Programming Languages And Their Compilers, Preliminary Notes Second Revised Version, Apri1, Courant Institute of Mathematical Sciences,New York University

[13] Daniel M. Younger , 1967, Recognition And Parsing Of Context-Free Languages In Time n^3 , Information And Control 10, 189-208 (1967)

[14] T. Kasami, K. Torii, 1969, A Syntax-Analysis Procedure For Unambiguous Context-Free Grammars, Journal Of The Acm (JACM) , Volume 16 Issue 3

[15] Piotr Skrzypczak, "Parallel Parsing of Context-Free Grammars," Blekinge Institute of Technology, Karlskorna, Master Thesis MCS-2011-28, 2011

[16] Ian Somerville, 2011, Software engineering, 9th edition, Pearson Education, Addison-Wesly, Boston, Massachusetts