SNATI
Seminar Nasional
Aplikasi Teknologi Informasi 2004
Yogyakarta, 19 Juni 2004

Sistem Penganalisis Sintaks Otomatis dengan Metode *Generalized-LR Parsing*

Rila Mandala, Antonius Sigit, Rinaldi Munir, Harlili
Laboratorium Ilmu dan Rekayasa Komputasi, Departemen Teknik Informatika,
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung, 40132.
e-mail: {rila, rinaldi, harlili)@if.itb.ac.id

Abstract

The main problem of difficulties in natural language processing is ambiguity. Almost in all level of natural language processing there are ambiguities, including in syntax level. LR-Parser is an efficient method that often be used in developing a programming language compiler for parsing, but unfortunately it can not be used for parsing natural language processing because it can not handle the ambiguity. This paper uses Generalized-LR Parsing, a modified-LR-Parsing method that can handle ambiguity, for parsing a sentence in Bahasa Indonesia. Generalized-LR Parsing use a graph-structured stack to cope the ambigutiy problem. It can successfully produce all parsing-trees for an ambiguous sentence in Bahasa Indonesia.

Keywords: *natural language processing, syntax analysis, ambiguity, Generalized LR-Parsing, parsing-tree*

1. Pendahuluan

Dalam konteks bahasa manusia, sintaksis merupakan pengetahuan mengenai bagaimana menggabungkan kata-kata menjadi kalimat yang benar. Kalimat terdiri atas kumpulan kata yang masing-masing memiliki fungsi dalam kalimat tersebut, misalnya sebagai predikat, subjek, objek, pelengkap, atau keterangan. Salah satu ciri khas yang dihadapi bahasa manusia adalah ketaksaan, yaitu suatu deretan kata dalam satu kalimat dapat memiliki lebih dari satu fungsi. Dengan demikian, kalimat tersebut juga memiliki lebih dari satu makna. Akibatnya, jika kita melakukan analisis sintaks terhadap kalimat tersebut, kita akan mendapatkan lebih dari satu pohon sintaks pula. Hasil dari analisis sintaks sangat diperlukan untuk berbagai aplikasi, seperti mesin penterjemah, *information retrieval*, dan sebagainya.

Makalah ini merupakan penelitian bagaimana melakukan proses analisis sintaks dengan menggunakan metode *LR-Parsing* yang telah dimodifikasi. Modifikasi dilakukan agar metode tersebut mampu menganalisis tata bahasa yang memiliki ketaksaan. Penelitian difokuskan pada bagaimana menemukan semua pohon sintaks yang mungkin pada tata bahasa yang memiliki ketaksaan (*ambiguity*). Hasil modifikasi terhadap metode *LR*-Parsing ini akan diterapkan untuk menganalisis tata bahasa Indonesia.

2. Kalimat dan Ketaksaaan

Menurut [ALW98], kalimat dasar memiliki ciri-ciri sebagai berikut:

- Terdiri atas satu klausa.

- Memiliki predikat dan subjek (sebagai unsur wajib).
- Susunan unsur-unsurnya menurut aturan yang paling umum.
- Tidak mengandung pertanyaan ataupun pengingkaran.

Dengan demikian, kalimat dasar dapat disamakan dengan kalimat tunggal deklaratif (kalimat berita) yang urutan unsur-unsurnya paling lazim.

Salah satu ciri kalimat dasar adalah susunan unsur-unsurnya menurut aturan yang paling umum. Kalimat dalam bahasa Indonesia sekurang-kurangnya memiliki dua gatra, yaitu predikat dan subjek. Berdasarkan hal tersebut, pola umum kalimat dasar dalam bahasa Indonesia adalah S - P - (O) - (Pel) - (Ket). Berdasarkan pola umum tersebut, dapat diturunkan enam pola kalimat dasar, yaitu:

1. S – P
2. S – P – O
3. S – P – Pel
4. S – P – Ket
5. S – P – O – Pel
6. S – P – O - Ket

Dari pola-pola kalimat dasar tersebut dapat diturunkan pola-pola kalimat misalnya sebagai berikut:

1. <Kal> ➔ <FN> <FN>
2. <Kal> ➔ <FN> <FV>
3. <Kal> ➔ <FN> <FAdj>
4. <Kal> ➔ <FN> <FNum>
5. <Kal> ➔ <FN> <FPrep>
6. <Kal> ➔ <FV> <FV>
7. <Kal> ➔ <FV> <FAdj>

Adapun gatra predikat, subjek, objek, pelengkap, dan keterangan dapat dibentuk dari pola misalnya seperti di bawah ini:

- <FN> ➔ <N> <FV>
- <FV> ➔ <V> <FN>
- <FAdj> ➔ <Adj> <Adv>
- <FNum> ➔ <Num> <Adv>

Suatu kalimat dikatakan memiliki ketaksaan apabila kalimat tersebut memiliki lebih dari satu makna. Makna suatu kalimat ditentukan oleh fungsi sintaksis deretan kata pembentuknya. Berarti, dalam kalimat yang taksa, suatu deretan kata dapat memiliki fungsi sintaksis lebih dari satu. Akibatnya, pola kalimat yang terbentuk pun lebih dari satu pula. Kita ambil contoh kalimat “Saya melihat seseorang menggunakan teropong.” Kalimat tersebut memiliki dua makna, apakah *saya* ataukah *seseorang* yang menggunakan teropong.

3. Tata Bahasa

Secara formal, tata bahasa bebas konteks didefinisikan sebagai G = (V,T,P,S), di mana:

V = himpunan simbol nonterminal
T = himpunan simbol terminal
P = himpunan aturan produksi
S = simbol awal

Sebagai contoh, tata bahasa Indonesia dapat didefinisikan sebagai berikut:

G = (V,T,P,S)

di mana:

V = {predikat, subjek, objek, frasa nominal, frasa verbal, …}

T = {saya, melihat, kendaraan, di, jalan, …}
P = <kalimat> ➔ <subjek> <predikat> <objek>
<subjek> ➔ <frasa nominal>
<predikat> ➔ <frasa verbal>
<objek> ➔ <frasa nominal>
…
S = <kalimat>

Pada aturan produksi tata bahasa bebas konteks berlaku dua hal sebagai berikut:

- Ruas kiri aturan produksi terdiri atas satu simbol nonterminal.
- Ruas kanan berupa string yang dibentuk dari simbol terminal dan nonterminal.

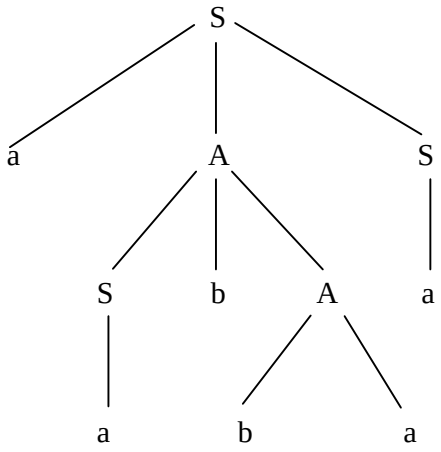
Proses penurunan kalimat dari simbol awal suatu tata bahasa dapat digambarkan sebagai rangkaian penerapan aturan produksi yang dimiliki oleh tata bahasa tersebut. Proses penerapan aturan produksi selalu dimulai terhadap simbol awal dari tata bahasa. Sebagai contoh, sebuah tata bahasa memiliki aturan-aturan produksi sebagai berikut:

<S> ➔ a <A> <S>
<S> ➔ a
<A> ➔ <S> b <A>
<A> ➔ <S> <S>
<A> ➔ b a

Proses penurunan terhadap ‘a a b b a a’ dapat dilakukan antara lain dengan cara sebagai berikut:

1. <S> ➔ a <A> <S> ➔ a <S> b <A> <S> ➔ a a b <A> <S> ➔ a a b b a <S> ➔ a a b b a a
2. <S> ➔ a <A> <S> ➔ a <A> a ➔ a <S> b <A> a ➔ a <S> b b a a ➔ a a b b a a

Pada cara pertama, aturan produksi selalu diterapkan terhadap simbol nonterminal paling kiri sehingga cara ini disebut *leftmost derivation*. Sebaliknya, pada cara kedua, aturan produksi selalu diterapkan terhadap simbol nonterminal paling kanan sehingga cara ini disebut *rightmost derivation*. Kedua cara di atas menghasilkan pohon sintaks sebagai berikut:



Gambar 1. Contoh Pohon Sintaks

Sebuah tata bahasa yang memiliki ketaksaan (*ambiguity*) akan memiliki lebih dari satu *leftmost derivation* atau lebih dari satu *rightmost derivation*.

4. Metode LR-Parsing

LR-Parsing merupakan salah satu metode analisis sintaks dengan cara *bottom-up parsing* secara deterministik tanpa proses runut balik. Makna *L* berarti *left to right* (melakukan pembacaan string dari kiri ke kanan) dan *R* berarti menghasilkan *right parse*. Pada metode ini, sebuah tabel *parsing* yang berisikan *grammar* (tata bahasa) diperlukan untuk menentukan aksi apa yang harus dilakukan (*shift* atau *reduce*) [TOM86]. Tata bahasa seperti ini, yang dapat menghasilkan *right parsing* secara deterministik, disebut tata bahasa LR (*LR grammar*).

Sebagai contoh, jika diberikan aturan produksi tata bahasa sebagai berikut:

1. <Kal> ➔ <N> <FV>
2. <FV> ➔ <V> <N>

Tabel *Parsing* yang berpadanan dengan aturan produksi tata bahasa di atas adalah seperti pada table 1.

Tabel 1. Contoh Tabel Parsing

State	**N**	**V**	**$**		**FV**	**Kal**
0	shift 2					1
1			acc			
2		shift 4			3	
3			reduce 1			
4	shift 5					
5			reduce 2			
	action table				*goto table*	

Keterangan:

Shift n : ambil kategori yang sedang dibaca, letakkan ke dalam *stack*, dan ubah *state* menjadi n.

Reduce n : ambil tumpukan dari *stack* dan satukan dengan menggunakan aturan ke-n, letakkan kembali hasilnya ke *stack*, ubah *state* sesuai dengan *goto table.*

Jika diberikan kalimat masukan "*Saya makan nasi*", maka proses *parsing* terhadap kalimat tersebut adalah seperti dalam table 2.

Tabel 2. Tahapan Proses Parsing Deterministik

No.	***Stack***	**Kata yang Dibaca**	**Aksi**
1.	0	saya	shift 2
2.	0 N 2	makan	shift 4
3.	0 N 2 V 4	nasi	shift 5
4.	0 N 2 V 4 N 5	$	reduce 2
5.	0 N 2 FV 3	$	reduce 1
6.	0 Kal 1	$	acc

Keterangan:

1. Pada *state* 0, kata yang dibaca adalah "*saya*" yang merupakan nomina (N), aksi yang dilakukan (shift 2) adalah letakkan "N" dan "2" ke *stack*, pindah ke *state* 2.
2. Pada *state* 2, kata yang dibaca adalah "makan", aksi yang dilakukan (shift 4) adalah letakkan "V" dan "4" ke *stack*, pindah ke *state* 4.
3. Pada *state* 4, kata yang dibaca adalah "nasi", aksi yang dilakukan (shift 5) adalah letakkan "N" dan "5" ke *stack*, pindah ke *state* 5.

4. Pada *state* 5, kalimat telah selesai dibaca. Aksi yang dilakukan (reduce 2) adalah mengambil isi *stack* (V dan N) dan mereduksinya dengan aturan produksi 2:
   <FV> ➔ <V> <N>
   State sekarang adalah 2 (lihat isi *stack* paling atas). Pada *goto table* kolom "FV", aksi yang dilakukan adalah pindah ke *state* 3, letakkan "FV" dan "3" ke *stack*.
5. Pada *state* 3 aksi yang dilakukan (reduce 1) adalah mengambil isi *stack* (N dan FV) dan mereduksinya dengan aturan produksi 1:
   <Kal> ➔ <N> <FV>
   State sekarang adalah 0. Pada *goto table* kolom "Kal", aksi yang dilakukan adalah pindah ke *state* 1, letakkan "S" dan "1" ke *stack*.
6. Pada *state* 1 kolom $, tanda "acc" berarti kalimat diterima tata bahasa.

Semua tata bahasa LR tidak memiliki ketaksaan dalam struktur bahasanya. Oleh karenanya, tata bahasa LR tidak dapat digunakan untuk merepresentasikan tata bahasa manusia.

5. Generalized-LR-Parsing

Bahasa pemrograman tidak mengenal adanya ketaksaan, artinya sebuah deretan simbol masukan pasti hanya menghasilkan sebuah pohon sintaks saja. Sebaliknya, pada bahasa manusia sering dijumpai struktur kalimat yang taksa. Akibatnya, bila kita melakukan proses analisis sintaks terhadap masukan string yang berupa kalimat bahasa manusia, kita mungkin akan mendapatkan lebih dari sebuah pohon sintaks.

Pada proses analisis sintaks dengan *LR-Parsing*, tabel *LR-Parsing* dibentuk dari *LR Grammars* yang sifatnya selalu tidak taksa. Jadi, tabel *LR-Parsing* tidak akan pernah memiliki entri ganda (*multiple entry*). Permasalahan mengenai entri ganda ini akan muncul apabila kita melakukan proses analisis sintaks terhadap tata bahasa yang memiliki ketaksaan di dalamnya. Akibatnya, kita harus melakukan modifikasi terhadap *LR-Parsing* agar mampu menangani munculnya entri ganda ini.

Bila tabel *parsing* memiliki entri ganda, proses *parsing* tidak dapat dilakukan secara deterministik, tetapi haruslah secara nondeterministik. Sebagai contoh, jika diberikan tata bahasa sebagai berikut:

1. <Kal> ➔ <N> <FPrep>
2. <Kal> ➔ <Kal> <V>
3. <FN> ➔ <N> <V>
4. <FPrep> ➔ <Prep> <FN>

Tabel *Parsing* yang berpadanan dengan tata bahasa di atas adalah sebagai berikut:

Tabel 3. Contoh Tabel Parsing dengan Entri Ganda

State	**N**	**V**	**Prep**	**$**		**FN**	**FV**	**FPrep**	**Kal**
0	sh 2								1
1		sh 3		acc					
2		re 4 sh 3	sh 5	re 4				4	
3		re 2		re 2					
4		re 1		re 1					
5	sh 2	sh 3				2			
	Action table						*Goto table*		

Dengan demikian, jika kita berada pada *state* 2 membaca "V", proses *parsing* akan dipecah menjadi dua proses, *reduce* 4 dan *shift* 3. Selanjutnya, kedua proses tersebut berjalan terpisah untuk menemukan pohon sintaks dari kalimat masukan.

Ada beberapa teknik yang dapat digunakan agar *LR-Parsing* dapat menangani entri ganda ini, yaitu dengan *stack list*, *tree-structured stack*, dan *graph-structured stack* [TOM86].

Stack list merupakan ide yang paling sederhana untuk menangani entri ganda secara nondeterministik dengan menggunakan prinsip paralelisme semu secara *breath-first search*. Pada prinsipnya, sejumlah proses dijalankan secara paralel. Setiap proses memiliki sebuah *stack* yang karakteristiknya sama dengan *stack* pada metode *LR-Parsing* standar. Bila ditemukan entri ganda, proses tersebut dipecah menjadi sejumlah proses sesuai dengan jumlah entrinya dengan cara mereplikasi *stack* dari proses tersebut. Semua proses berjalan secara sinkron, artinya "melihat" kata yang sama pada waktu yang sama.

Dengan teknik ini, ada kemungkinan lebih dari satu *stack* yang identik. Akibatnya, mungkin terjadi redundansi aksi. Hal ini tentunya akan merugikan dari segi performansi dan efisiensi. Selain itu, jumlah *stack* yang dihasilkan bertambah secara eksponensial terhadap jumlah ketaksaan yang ditemukan.

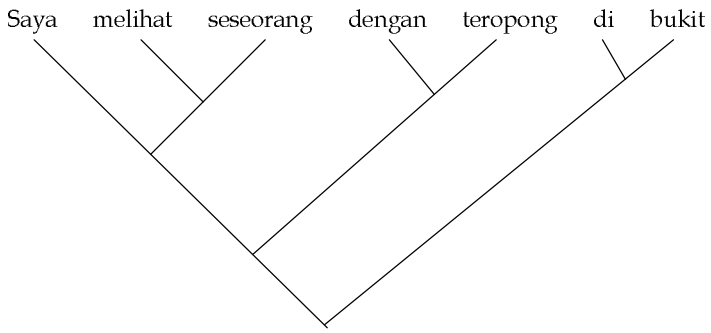
Telah disebutkan bahwa pada teknik menggunakan *stack list*, mungkin terjadi redundansi aksi yang disebabkan oleh adanya lebih dari satu *stack* yang identik. Untuk mengatasi aksi redundan tersebut, proses-proses yang memiliki aksi yang sama tersebut digabung menjadi satu dengan cara mengkombinasikan *stack* dari proses tersebut. Akibatnya, *stack-stack* tersebut akan direpresentasikan sebagai pohon (*tree*). Secara umum, sistem akan mengelola sejumlah pohon secara paralel sehingga *stack-stack* tersebut akan direpresentasikan sebagai hutan (*forest*).

Dengan teknik ini, jumlah aksi yang dilakukan akan lebih sedikit jika dibandingkan dengan jumlah aksi pada teknik menggunakan *stack list*. Walaupun demikian, jumlah cabang pohon yang dihasilkan (yang mencerminkan *stack* yang memiliki aksi redundan tadi) tetap bertumbuh secara eksponensial. Dengan kata lain, walaupun teknik *tree-structured stack* lebih baik daripada *stack list* dilihat dari sisi kompleksitas waktu, dari sisi kompleksitas ruang *tree-structured stack* tidaklah lebih baik.

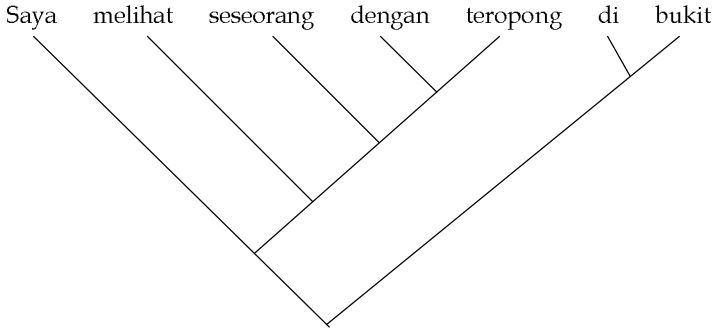
Ide dari *graph-structured stack* ialah tidak diperlukan adanya replikasi dari keseluruhan *stack* untuk merepresentasikan proses yang memiliki entri ganda. Replikasi hanya dilakukan terhadap sebagian dari *stack* tersebut sedemikian sehingga proses *parsing* dengan cara yang sama terhadap suatu bagian kalimat hanya dilakukan satu kali (*one-pass*). Hal ini disebabkan saat proses-proses mem-*parsing* bagian yang sama dari suatu kalimat, proses-proses tersebut berada pada keadaan (*state*) yang sama sehingga akan digabungkan menjadi sebuah proses. Teknik ini dilihat dari sisi kompleksitas ruang lebih baik daripada kedua teknik sebelumnya.

6. Contoh Keluaran

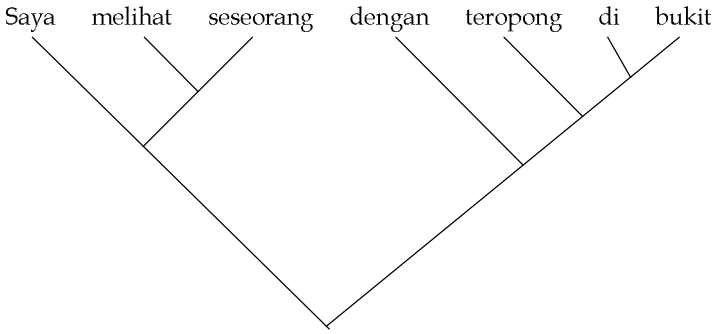
Sebagai contoh, proses analisis sintaks terhadap kalimat masukan "*Saya melihat seseorang dengan teropong di atas bukit*" akan menghasilkan pohon sintaks sebagai berikut ini.



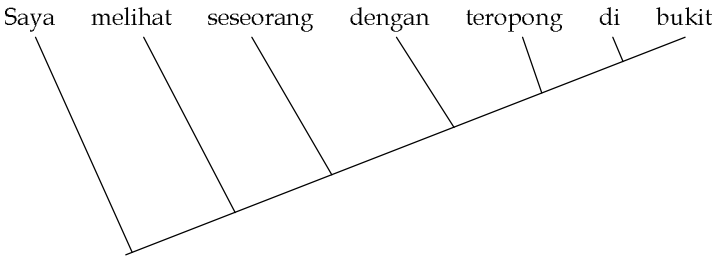
Gambar 2. Pohon Sintaks Pertama



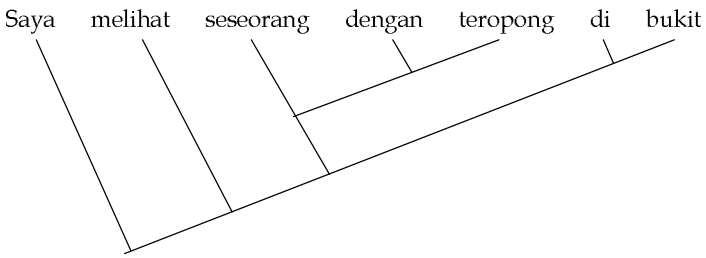
Gambar 3. Pohon Sintaks Kedua



Gambar 4. Pohon Sintaks Ketiga



Gambar 5. Pohon Sintaks Keempat



Gambar 6. Pohon Sintaks Kelima

Masing-masing pohon sintaks di atas dapat diinterpretasikan sebagai berikut:

1. Pohon pertama menggambarkan bahwa:
   - Saya melihat seseorang
   - Saya menggunakan teropong
   - Saya berada di bukit
2. Pohon kedua menggambarkan bahwa:
   - Saya melihat seseorang
   - Seseorang menggunakan teropong
3. Pohon ketiga menggambarkan bahwa:
   - Saya melihat seseorang
   - Saya melihat teropong
   - Teropong berada di bukit
4. Pohon keempat menggambarkan bahwa:
   - Tidak ada informasi yang bisa didapatkan dari pohon sintaks ini
5. Pohon kelima menggambarkan bahwa:
   - Saya melihat seseorang
   - Seseorang menggunakan teropong
   - Saya berada di bukit

7. Kesimpulan

a. Metode *LR-Parsing* dapat digunakan untuk menganalisis sintaks tata bahasa Indonesia, ditunjukkan dengan terbentuknya lebih dari satu pohon sintaks sebagai hasil analisis tersebut. Analisis sintaks dapat melakukan pengelompokan kata yang membantu proses interpretasi makna.
b. Pendefinisian tata bahasa yang digunakan sangat menentukan keberhasilan proses analisis sintaks.
c. Penambahan aturan produksi dalam tata bahasa yang digunakan dapat menimbulkan ketaksaan yang seharusnya tidak ada.
d. Ketiadaan tanda baca dalam kalimat yang digunakan sebagai masukan memperbesar peluang terjadinya ketaksaan.

8. Ucapan Terima Kasih

Penelitian yang dilakukan oleh penulis di Laboratorium Ilmu dan Rekayasa Komputasi, Departemen Teknik Informatika ITB ini sebagian didanai oleh dana RUT (Riset Unggulan Terpadu) IX dari KMNRT (Kantor Menteri Negara Riset dan Teknologi), oleh karena itu penulis mengucapkan terima kasih.

Daftar Pustaka

[AHO72] Aho, Alfred V. dan Jeffrey D. Ullman. 1972. *The Theory Of Parsing, Traslation, And Compiling Volume I: Parsing*. Prentice-Hall, Inc.

[ALW98] Alwi, Hasan, Soenjono Dardjowidjojo, Hans Lapoliwa, Anton M. Moeliono. 1998. *Tata Bahasa Baku Bahasa Indonesia*. Edisi Ketiga. Jakarta: Balai Pustaka.

[ARI00] Arifin, E. Zaenal dan S. Amran Tasai. 2000. *Cermat Berbahasa Indonesia*. Cetakan IV. Jakarta: Akademika Pressindo.

[HOP79] Hopcroft, John E. dan Jeffrey D. Ullman. 1979. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Publishing Company.

[MOL88] Moll, Robert N., Michael A. Arbib, A. J. Kfoury. 1988. *An Introduction To Formal Language Theory*. Springer-Verlag.

[OBE89] Obermeier, Klaus K. 1989. *Natural Language Processing Technologies In Artificial Intelligence: The Science And Industry Perspective*. Ellis Horwood Limited.

[SAG81] Sager, Naomi. 1981. *Natural Language Information Processing: A Computer Grammar of English and It's Applications*. Addison-Wesley Publishing Company.

[SET96] Sethi, Ravi. 1996. *Programming Language: Concepts & Constructs*. 2nd Edition. Addison Wesley Longman, Inc.

[TOM86] Masaru, Tomita. 1986. *Efficient Parsing For Natural Language: A Fast Algorithm For Practical Systems*. Kluwer Academic Publishers.