Membangun sebuah Combinatory Categorial Grammar (CCG) Supertagger Berbasis Maximum Entropy untuk Bahasa Indonesia

Proposal Tugas Akhir

Kelas TA NLP

Wisnu Adi Nurcahyo
NIM: 1301160479

Program Studi Sarjana Informatika

Fakultas Informatika

Universitas Telkom

Bandung

2019

Lembar Persetujuan

Membangun sebuah Combinatory Categorial Grammar (CCG) Supertagger Berbasis Maximum Entropy untuk Bahasa Indonesia

Building a Combinatory Categorial Grammar (CCG) Supertagger Based on the Maximum Entropy for Bahasa Indonesia

Wisnu Adi Nurcahyo
NIM: 1301160479

Proposal ini diajukan sebagai usulan pembuatan tugas akhir pada
Program Studi Sarjana Informatika
Fakultas Informatika Universitas Telkom

Bandung, 13 November 2019
Menyetujui

Calon Pembimbing 1

Dr. Ade Romadhony, S.T., M.T.
NIP: 06840042

Abstrak

Dalam pemrosesan bahasa alami, combinatory categorial grammar (CCG) merupakan salah satu formalisme tata bahasa yang dapat digunakan untuk membangun sebuah *parser* yang umumnya dikenal sebagai CCG *parser*. CCG *parser* dapat digunakan untuk berbagai macam keperluan dalam pemrosesan bahasa alami. Sebagai contoh, CCG *parser* dapat digunakan untuk memperoleh informasi (*information extraction*) dari suatu kalimat yang kemudian membentuk sebuah *query*. Agar dapat bekerja, CCG *parser* membutuhkan CCG *lexicon*. CCG *lexicon* diperoleh dari proses yang bernama *supertagging*. *Supertagging* adalah proses pelabelan suatu token kata terhadap *supertag*-nya. Perangkat lunak yang melakukan *supertagging* disebut sebagai *supertagger*. Demikian itu, *supertagging* merupakan langkah pertama yang perlu dilakukan sebelum membangun sebuah CCG *parser*. *Supertagger* yang dibangun dalam tugas akhir ini dimaksudkan sebagai produsen CCG *lexicon* bahasa Indonesia untuk riset-riset yang berkenaan dengan CCG di masa yang akan datang.

Kata Kunci: natural language processing, combinatory categorial grammar, supertagger, maximum entropy model, bahasa indonesia, haskell

Daftar Isi

Bab I

Pendahuluan

1.1 Latar Belakang

Riset pemrosesan bahasa natural untuk bahasa Indonesia saat ini terbilang sedikit. Bahkan, masih banyak area riset yang belum tersentuh seperti contohnya *combinatory categorial grammar* (CCG). Riset mengenai CCG untuk bahasa Inggris sudah cukup matang. Adapun untuk bahasa lainnya, seperti bahasa Vietnam, sudah memulai untuk menggunakan CCG di dalam penelitiannya [5]. Umumnya terdapat dua cara yang paling sering digunakan untuk mengembangkan CCG *supertagger* maupun CCG *parser* bahasa lokal yaitu (1) membangun *dataset* CCG *supertag* secara manual maupun semi-otomatis atau (2) melakukan transfer *dataset* dari CCGBank ke dalam bahasa lokal dengan cara melakukan alih bahasa dan bila perlu melakukan penyesuaian untuk *supertag*-nya [2]. Berhubung riset CCG di Indonesia masih kurang peminatnya, tugas akhir ini diangkat untuk menimbulkan minat dengan cara menyediakan *tool* esensial yaitu CCG *supertagger*.

Tugas akhir dengan judul " Membangun sebuah Combinatory Categorial Grammar (CCG) Supertagger Berbasis Maximum Entropy untuk Bahasa Indonesia " berusaha untuk membangun versi awal dari CCG *supertagger* untuk bahasa Indonesia yang mana harapannya dapat menjadi inisiator riset pemrosesan bahasa natural dengan tema CCG sehingga ke depannya akan ada lebih banyak riset mengenai CCG yang tersedia. Acuan utama *paper* yang digunakan dalam membangun *supertagger* ini adalah *paper* yang dipublikasikan oleh Stephen Clark pada tahun 2002 [1]. *Supertagger* yang dimaksud dalam tugas akhir ini akan dibangun dengan menggunakan model Maximum Entropy (MaxEnt) dan implementasinya akan ditulis dalam bahasa pemrograman Haskell. Model MaxEnt digunakan karena keterbatasan *dataset* untuk melakukan *learning*. Selain itu, model MaxEnt sudah terbukti dapat digunakan untuk membangun CCG *supertagger* [1] berdasarkan *paper* yang dipublikasikan oleh Stephen Clark. Adapun bahasa pemrograman Haskell digunakan karena abstraksi bahasanya yang sangat mendekati *category theory* serta kemampuannya yang sangat baik dalam pemrosesan data.

1.2 Perumusan Masalah

Rumusan masalah yang akan diangkat yaitu:

1. Bagaimana proses pembangunan CCG *supertagger* untuk bahasa Indonesia?
2. Bagaimana dengan akurasi CCG *supertagger* apabila dibandingkan dengan POS *tagger* untuk bahasa Indonesia?

1.3 Tujuan

Tujuan yang diharapkan dapat tercapai oleh tugas akhir ini yaitu:

1. Membangun dan merilis CCG *supertagger* pertama untuk bahasa Indonesia.
2. Membuka peluang riset untuk CCG *parser* bahasa Indonesia.

1.4 Batasan Masalah

Hipotesis dari tugas akhir ini yaitu:

1. Memberikan label CCG untuk proses *training* merupakan permasalahan utama dari tugas akhir ini.
2. *Supertagger* yang akan dibangun kemungkinan besar memiliki akurasi yang cenderung rendah.
3. CCG *supertag* sudah dapat digunakan oleh CCG *parser* bahasa Indonesia (apabila ada).

1.5 Rencana Kegiatan

Rencana kegiatan yang akan dilakukan adalah sebagai berikut:

- Studi literatur
- Studi *tools* yang tersedia
- Studi bahasa pemrograman yang akan digunakan
- Perancangan sistem *supertagger*
- Membangun *supertagger*
- Memeriksa hasil

1.6 Jadwal Kegiatan

Laporan proposal ini akan dijadwalkan sesuai dengan tabel 1.1.

Tabel 1.1: Jadwal kegiatan proposal tugas akhir.

No	**Kegiatan**	**Bulan ke-**																							
		1				**2**				**3**				**4**				**5**				**6**			
1	Studi Literatur																								
2	Studi *Tools* yang Tersedia																								
3	Studi Bahasa Pemrograman																								
4	Pengumpulan Data																								
5	Analisis dan Perancangan Sistem																								
6	Implementasi Sistem																								
7	Analisa Hasil Implementasi																								
8	Penulisan Laporan																								

Bab II

Kajian Pustaka

2.1 Categorial Grammar

Categorial Grammar (CG) merupakan sebuah istilah yang mencakup beberapa formalisme terkait yang diajukan untuk sintaks dan semantik dari bahasa alami serta untuk bahasa logis dan matematis [9]. Karakteristik yang paling terlihat dari CG adalah bentuk esktrim dari leksikalismenya di mana beban utama (atau bahkan seluruh beban) sintaksisnya ditanggung oleh leksikon. Konstituen tata bahasa dalam *categorial grammar* dan khususnya semua leksikal diasosiasikan dengan suatu *type* atau "*category*" (dalam *category theory*) yang mendefinisikan potensi mereka untuk dikombinasikan dengan konstituen lain untuk menghasilkan konstituen majemuk. *Category* tersebut adalah salah satu dari sejumlah kecil *category* dasar (seperti NP) atau *functor* (dalam *category theory*).

Ada beberapa notasi berbeda untuk *category* dalam merepresentasikan *directional*-nya. Notasi yang paling umum digunakan adalah "*slash notation*" yang dipelopori oleh Bar-Hilel, Lambek, dan kemudian dimodifikasi dalam kelompok teori yang dibedakan sebagai tata bahasa "*combinatory*" *categorial grammar* (CCG). Sebagai contoh, *category* (S\NP)/NP merupakan suatu *functor* yang memiliki dua buah notasi *slash* yaitu \ dan /. Masing-masing notasi *slash* tersebut merepresentasikan *directionality* yang berbeda. Notasi *forward slash*, /, mengindikasikan bahwa argumen dari suatu *functor* X/Y ada di bagian kanan atau dengan kata lain Y. Adapun *backward slash*, \, mengindikasikan bahwa argumen dari suatu *functor* X\Y ada di bagian kiri atau dengan kata lain X. Demikian itu, penggunaan notasi *slash* yang tepat sangat penting dikarenakan hal ini dapat mempengaruhi konstituen dari hasil "kombinasi" *category*-nya.

2.2 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) merupakan salah satu formalisme tata bahasa yang gaya aturannya diturunkan dari *categorial grammar* dengan beberapa penambahan aturan dan istilah baru [10]. Di CCG, *category* dapat dipasangkan dengan *semantic representation*. Dalam hal ini, *se-*

$$
\begin{aligned}
\text{Pamungkas} &\vdash \text{NP} : \mathit{pamungkas}' \\
\text{Setyo} &\vdash \text{NP} : \mathit{setyo}' \\
\text{dan} &\vdash \text{CONJ} : \lambda x.\lambda y.\lambda f.\ (f\ x) \wedge (f\ y) \\
\text{menyukai} &\vdash (\text{S}\backslash\text{NP})/\text{NP} : \lambda x.\lambda y.\ \mathit{suka}(y, x) \\
\text{rendang} &\vdash \text{NP} : \mathit{rendang}'
\end{aligned}
$$

Gambar 2.1: Kamus yang memetakan token kata ke bentuk CCG *lexicon*-nya.

mantic representation yang dimaksud adalah abstraksi fungsi lambda (dalam *lambda calculus*, *lambda function*). Sebagai contoh, *category* (S\NP)/NP dapat dipasangkan dengan fungsi lambda $\lambda x.fx$ sehingga dapat ditulis menjadi (S\NP)/NP : $\lambda x.fx$. Adapun pemetaan dari suatu token kata ke *category*-nya menggunakan notasi $\vdash$. Sebagai contoh, anggap saja kita memiliki kamus pemetaan seperti pada Gambar 2.1. Apabila kita memiliki kalimat "Pamungkas dan Setyo menyukai rendang", maka kita dapatkan:

Pamungkas	dan	Setyo	menyukai	rendang
NP	CONJ	NP	(S\NP)/NP	NP
$: \mathit{pamungkas}'$	$: \lambda x.\lambda y.\lambda f.\ (f\ x) \wedge (f\ y)$	$: \mathit{setyo}'$	$: \lambda x.\lambda y.\ \mathit{suka}(y, x)$	$: \mathit{rendang}'$

Ada beberapa operasi yang dapat dilakukan dalam CCG. *Operand* dari operasi yang dimaksud adalah *category*. Berdasarkan contoh di atas, akan ada tiga operasi yang dijalankan yaitu *coordination*, *forward application*, dan *type rising*. Untuk mendapatkan hasil yang diinginkan, kita lakukan *type rising* sebelum *forward application* di akhir. Sehingga, kita dapatkan:

Pamungkas	dan	Setyo	menyukai	rendang
NP	CONJ	NP	(S\NP)/NP	NP
$: \mathit{pamungkas}'$	$: \lambda x.\lambda y.\lambda f.\ (f\ x) \wedge (f\ y)$	$: \mathit{setyo}'$	$: \lambda x.\lambda y.\ \mathit{suka}(y, x)$	$: \mathit{rendang}'$
		$<\&>$		$>$
NP			S\NP	
$: \lambda f.\ (f\ \mathit{pamungkas}') \wedge (f\ \mathit{setyo}')$			$: \lambda y.\ \mathit{suka}(y, \mathit{rendang}')$	
$> T$				
S/(S\NP)				
$: \lambda f.\ (f\ \mathit{pamungkas}') \wedge (f\ \mathit{setyo}')$				
$>$				
S				
$: \mathit{suka}(\mathit{pamungkas}', \mathit{rendang}') \wedge \mathit{suka}(\mathit{setyo}', \mathit{rendang}')$				

Berdasarkan hasil evaluasi tersebut, kita dapatkan *query* 2.1 yang diperoleh dari kalimat "Pamungkas dan Setyo menyukai rendang". Demikian itu, komputer dapat melakukan komputasi berdasarkan *query* yang telah diperoleh.

Kegiatan tersebut merupakan apa yang disebut dengan CCG *parsing*. Untuk dapat melakukan parsing, CCG *lexicon* diperlukan. Untuk mendapatkan CCG *lexicon* kita dapat menggunakan CCG *supertagger* yang akan melakukan pelabelan suatu token kata ke CCG *lexicon* berdasarkan pemetaannya.

$$suka(pamungkas', rendang') \wedge suka(setyo', rendang') \tag{2.1}$$

2.3 Category Theory

Category Theory (CT) merupakan formalisme yang dapat digunakan untuk memformalkan struktur matematis [3]. CT mempelajari *category* yang merupakan sebuah representasi dari suatu abstraksi konsep matematis. Suatu *category* memiliki kumpulan *object* dan *morphism*. Untuk mempermudah pemahaman mengenai CT, kita akan gunakan *category of set* (kategori dari himpunan) sebagai contoh. Dalam *category of set*, *object*-nya adalah himpunan dan *morphism*-nya (terkadang disebut dengan *arrow*) adalah fungsi (*function*, sebuah pemetaan). Kemudian, pemetaan dari suatu *category* C ke *category* D yang dipetakan oleh F ($F : C \to D$) disebut sebagai *functor*.

2.4 Lambda Calculus

Lambda calculus (λ-*calculus*) merupakan sebuah formalisme yang dikembangkan oleh Alonzo Church sebagai alat yang digunakan untuk memahami konsep komputasi yang efektif [8]. Formalisme λ-*calculus* cukup populer dan bahkan dijadikan sebagai pondasi teori bagi paradigma pemrograman *functional programming*. Konsep utama dari λ-*calculus* adalah apa yang disebut dengan *expression*. Suatu *expression* dalam λ-*calculus* terdiri dari tiga bagian yaitu *lambda notation* (λ), *argument* (seperti a, b, c, x, dan lain-lain), dan *body* yang dipisahkan dengan tanda titik. Sebagai contoh, fungsi lambda $\lambda x.x$ merupakan sebuah fungsi identitas yang mengambil argumen x kemudian mengembalikan nilai x itu sendiri. Dalam hal ini, terlihat bahwa notasi λ merupakan sebuah penanda bagi suatu fungsi lambda. Kemudian, pengubah x setelah notasi λ merupakan argumen dari fungsi tersebut. Selanjutnya, tanda titik merupakan pemisah antara *head* dan *body* fungsi lambda. Terakhir, setelah tanda titik adalah *body* dari suatu fungsi lambda yang mana berupa *expression*.

Untuk mempermudah pemahaman, λ-*calculus* dapat diperlakukan seperti fungsi tanpa nama. Sebagai contoh, fungsi lambda ($\lambda x.x+5$) apabila diberikan nilai 2 sehingga menjadi ($\lambda x.x + 5)2$ akan dievaluasi menjadi $\lambda(2).(2) + 5$. Demikian itu, nilai yang dikembalikan oleh fungsi tersebut adalah 7. Sama seperti fungsi pada umumnya, konsep ini bernama *substition* (substitusi). Memahami λ-*calculus* dirasa perlu berhubung dalam tugas akhir ini λ-*calculus* digunakan sebagai bentuk formal di *category* dalam konteks CCG *lexicon*. Meskipun λ-*calculus* tidak sesederhana yang dijelaskan sebelumnya, setidaknya memahami

λ-*calculus* seperti ini sudah cukup untuk dapat membangun *supertagger* yang ada di tugas akhir ini.

2.5 Supertagging

Supertagging merupakan proses yang memetakan suatu token kata ke bentuk *supertag*-nya. *Supertagging* awalnya diajukan untuk LTAG (*Lexicalized Tree-Adjoining Grammars*) yang memiliki konsep *compliate locally, simplify globally* (CLSG) [6]. CCG *supertagging* artinya proses pemetaan dari suatu token kata ke dalam bentuk CCG *supertag* atau dapat juga disebut sebagai CCG *lexicon*. *Supertag* mirip dengan POS *tag*. Perbedaannya, *supertag* memiliki bentuk formalisme yang lebih kompleks dari POS *tag*. Hal ini dikarenakan *supertag* menyimpan informasi lain selain tanda gramatikalnya (NP, NN, VB, dan sebagainya). Sebagai contoh, CCG *supertag* memiliki bentuk dengan format:

$$< \text{categorial grammar tag} > : < \text{semantic representation} >$$

Kita dapat menggunakan notasi $\vdash$ untuk memetakan token kata ke bentuk CCG *supertag*-nya. Anggap kita memiliki kata kerja "menyukai" yang akan dipetakan ke (S\NP)/NP : $\lambda x.\lambda y.\ suka(y, x)$, kita dapatkan:

$$\text{menyukai} \ \vdash \ (S\backslash NP)/NP \ : \ \lambda x.\lambda y.\ suka(y, x)$$

Adapun sebuah *tool* yang melakukan proses *supertagging* ini dinamakan *supertagger*. *Supertagger* sederhananya mengambil daftar token kata yang kemudian untuk setiap token kata tersebut "dilabelkan" dengan *supertag*-nya.

2.6 Maximum Entropy Model

Maximum Entropy (MaxEnt) merupakan model statistik yang dapat digunakan untuk melakukan *train* korpus teranotasi dengan Part-Of-Speech (POS) *tag* [7]. Salah satu aplikasi MaxEnt adalah POS *tagger* yang mana memiliki hasil akurasi yang lebih baik ketimbang *state-of-the-art*. Adapun model probabilitasnya didefinisikan dalam $\mathcal{H} \times \mathcal{T}$, dimana $\mathcal{H}$ adalah himpunan dari kemungkinan kata dan konteks *tag*, atau "*history*" (riwayat), dan $\mathcal{T}$ adalah himpunan dari *tag* yang diizinkan. Model probabilitas dari suatu *history* h bersama dengan *tag* t didefinisikan dalam persamaan 2.2.

$$p(h, t) = \pi\mu \prod_{j=1}^{k} a_j^{f_j(h,t)} \tag{2.2}$$

dimana π merupakan konstan normalisasi, $\{\mu, a_1, \ldots, a_k\}$ merupakan parameter model positif, dan $\{f_1, \ldots, f_k\}$ merupakan apa yang kita sebut sebagai "*feature*", dimana $f_j(h, t) \in \{0, 1\}$.

Meskipun MaxEnt dipernalkan untuk POS *tagging*, MaxEnt dapat pula digunakan untuk *supertagging*. Stephen Clark 2002 mempublikasikan literatur *supertagging* untuk CCG yang mana MaxEnt merupakan model yang digunakan. Adapun persamaan modelnya dapat dilihat di persamaan 2.3.

$$p(c|h) = \frac{1}{Z(h)} e^{\Sigma_i \lambda_i f_i(c,h)} \tag{2.3}$$

dimana c merupakan *category*, h merupakan *context*, fungsi $f_i(c,h)$ merupakan "*feature*" dari suatu *category* dan *context*, dan $Z(h)$ merupakan konstan normalisasinya. Adapun contoh "*feature*" yang dimaksud dapat dilihat di persamaan 2.4.

$$f_j(c,h) = \begin{cases} 1 & , \text{if } merupakan\_kata\_yang(h) = true \ \& \ c = \text{NP/N} \\ 0 & , \text{selainnya} \end{cases} \tag{2.4}$$

Bab III

Perancangan Sistem

Sebelum dapat membangun CCG *supertagger*, sebuah *dataset* diperlukan untuk melatih model *classifier* MaxEnt. Berhubung *dataset* CCG *supertag* untuk bahasa Indonesia belum tersedia, tidak ada pilihan lain selain membuat *dataset*-nya terlebih dahulu. Salah satu cara untuk membuat *dataset* adalah dengan memanfaatkan POS *tag* dari suatu lema yang kemudian diberikan *supertag*-nya secara manual. Setelah *dataset* memiliki *supertag* yang cukup, barulah proses *train* MaxEnt dapat dilakukan. Model yang telah dilatih akan disimpan ke dalam format JSON khusus agar ke depannya dapat digunakan tanpa perlu melakukan *train* lagi. Setelah itu, diperlukan program yang dapat memanfaatkan model yang telah dilatih untuk membuktikan bahwasannya model yang telah dibangun dapat menghasilkan *supertag* yang sesuai. Adapun langkah terakhirnya adalah dengan menyediakan dukungan *web application* berbasis RESTful API agar siapapun dapat memanfaatkan *supertagger* ini tanpa perlu terhalang oleh batasan bahasa pemrograman yang digunakan.

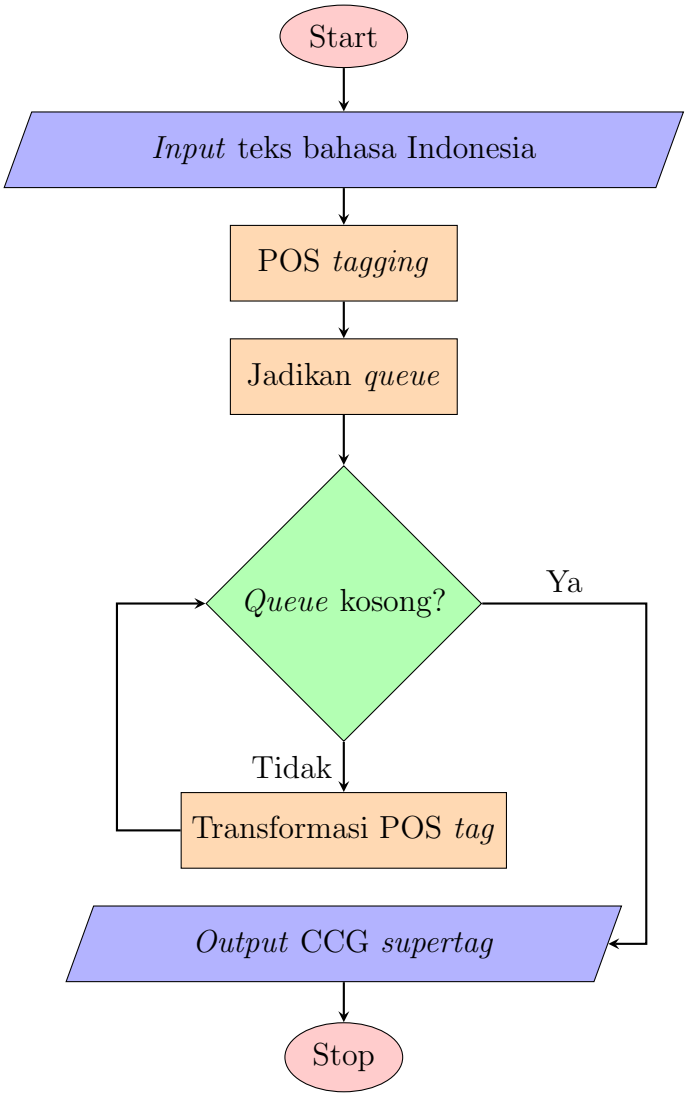
3.1 Pembuatan Dataset

Dataset untuk melakukan *train* CCG sayangnya belum tersedia. Karenanya, kita akan membuatkan *dataset* secara semi otomatis dengan memanfaatkan POS *tagger* untuk bahasa Indonesia. Secara formal, *flowchart* untuk proses pembuatan *dataset* dapat dilihat pada Gambar 3.1.

3.1.1 Input Teks Bahasa Indonesia

Pada bagian *input* dalam pembuatan *dataset*, kita dapat mengambil teks bahasa Indonesia dari Indonesian Treebank[1] dan/atau dari beberapa contoh artikel yang terdapat di *website* Wikipedia Indonesia[2]. Indonesian Treebank terdapat setidaknya 1000 kalimat yang dirasa cukup untuk melakukan *training* menggunakan model MaxEnt. Apa yang perlu dilakukan setelah mengambil *input* adalah membersihkan bentuk *tree*-nya untuk kemudian diambilkan kalimat yang sebenarnya. Caranya cukup sederhana yaitu dengan mengambil *leaf* dari *tree* tersebut kemudian disatukan di suatu pengubah dengan tipe data *string*.

Sebagai contoh, salah satu *tree* yang terdapat di Indonesian Treebank dapat



Gambar 3.1: Alur kerja pembuatan *dataset* untuk CCG *supertag*.

dilihat pada Gambar 3.2. Apa yang dimaksud dengan *leaf* pada *tree* tersebut adalah $(Kera)$, $(untuk)$, $(*)$, $(amankan)$, $(pesta\ olahraga)$. Terkhusus untuk *leaf* dengan bentuk spesial, seperti $(*)$, kita hilangkan sehingga teks yang diperoleh dari *tree* tersebut adalah "Kera untuk amankan pesta olahraga". Selanjutnya, contoh teks yang akan digunakan agar konsisten yaitu "Pamungkas dan Setyo menyukai rendang".

[1]github.com/famrashel/idn-treebank/blob/master/Indonesian_Treebank.bracket
[2]id.wikipedia.org

$$
\begin{array}{l}
(NP \\
\quad (NN\ (Kera)) \\
\quad (SBAR \\
\quad\quad (SC\ (untuk)) \\
\quad\quad (S\ (NP-SBJ\ (*)) \\
\quad\quad\quad (VP \\
\quad\quad\quad\quad (VB\ (amankan)) \\
\quad\quad\quad\quad (NP\ (NN\ (pesta\ olahraga)))))))
\end{array}
$$

Gambar 3.2: Salah satu contoh *tree* dalam Indonesian Treebank.

Pamungkas	dan	Setyo	menyukai	rendang
NNP	CC	NNP	VB	X

Gambar 3.3: Kalimat contoh dengan POS *tag*-nya.

3.1.2 POS Tagging

Berdasarkan teks bahasa Indonesia yang telah diambil, kita manfaatkan *tool* POS *tagger* bahasa Indonesia untuk mendapatkan *lexical category* atomik untuk masing-masing token. Dengan memanfaatkan POS *tag* kita dapat membuat *dataset* untuk CCG *supertag* lebih mudah dibandingkan dengan memberikan *tag* CCG secara manual. Ide dasarnya yaitu kita akan mentransformasikan POS *tag* yang didapatkan menjadi CCG *supertag* berdasarkan aturan-aturan khusus yang telah ditentukan. Sebagai contoh, kita dapat membuat aturan seperti mentransformasikan VB menjadi (S\NP)/NP. Dengan menggunakan contoh kalimat yang sama seperti di bagian sebelumnya, yaitu "Pamungkas dan Setyo menyukai rendang", setelah menggunakan POS *tagger* bahasa Indonesia kita dapatkan hasil sesuai dengan Gambar 3.3.

3.1.3 Transformasi POS Tag

Pada bagian proses transformasi POS *tag* ke bentuk CCG *supertag*-nya, kita buatkan aturan-aturan transformasinya. Aturan transformasi tersebut merupakan pemetaan berbasis aturan. Sebagai contoh, kata "menyukai" memiliki POS *tag* VB. Anggap saja dalam aturan transformasi terdapat pemetaan VB $\vdash$ (S\NP)/NP. Sehingga, kita dapatkan CCG *supertag* untuk "menyukai" yaitu (S\NP)/NP. Kendati demikian, masih ada bagian yang belum kita dapatkan yaitu *semantic representation*-nya. Kita dapat menggunakan *stemmer* bahasa Indonesia agar mendapatkan *root words* dari kata "menyukai" yaitu "suka". Langkah terakhirnya adalah membuatkan *semantic representation*-nya berdasarkan *root words* yang telah diperoleh sehingga kita

dapatkan fungsi lambdanya yaitu $\lambda x.\lambda y.$ suka(y, x).

Selain menggunakan *stemmer*, kita dapat menggunakan *morphological analyzer*. *Morphological analyzer* salah satu kegunaannya yaitu dapat menghasilkan *root words* sehingga dapat kita manfaatkan untuk membuat fungsi lambda. Untuk bahasa Indonesia, kita dapat menggunakan *tool* bernama MorphInd[3]. Selain MorphInd, alternatif *tool* yang dapat digunakan adalah IndMA (Indonesian morphological analyzer). Namun, MorphInd dipilih karena dapat memberikan keluaran berupa morfem tersegmentasi [4]. Kita dapat menggunakan MorphInd sebagai pelengkap POS *tagger* untuk bahasa Indonesia. Hal ini agar *dataset* yang dibuatkan secara semi-otomatis ini dapat memiliki kualitas yang baik.

3.1.4 Output CCG Supertag

Keluaran dari bagian pembuatan *dataset* ini adalah sebuah berkas JSON (JavaScript Object Notation) berisi CCG *supertag* lengkap dan beberapa berkas JSON dari CCG *supertag* yang lemanya dikelompokkan berasarkan alfabet. Adapun format JSON dari *dataset* yang disimpan dalam berkas tersebut dapat dilihat pada Gambar 3.4. Untuk setiap kalimat dalam berkas tersebut direpresentasikan oleh dua objek yaitu (1) "tokens" berupa daftar token, dan (2) "supertags" berupa daftar *supertag* untuk token ke-(i, j) dimana $1 \leq i \leq n$ dan $1 \leq j \leq m$ dalam sistem 1-*indexed array*. Sebagai contoh, kalimat "Pamungkas menyukai rendang" representasinya dapat dilihat pada Gambar 3.5. Demikian itu, kita dapat mengambil *dataset* secara lengkap dengan CCG *supertag*-nya.

3.2 Mempersiapkan CCG Supertagger

3.2.1 Melatih Model MaxEnt

Model yang akan digunakan oleh *supertagger* ini adalah Maximum Entropy (MaxEnt). Model ini dipilih karena ketersediaan *dataset* bahasa Indonesia yang masih sangat kurang. Selain itu, model MaxEnt digunakan di sebuah riset yang dilakukan oleh Stephen Clark dalam pembuatan *supertagger*-nya. Bahkan, performansi *supertagger* yang dikembangkan sangat baik. Riset tersebut pada intinya membuktikan bahwasannya MaxEnt dapat digunakan di *supertagger* juga meskipun pada peruntukannya MaxEnt dibuat untuk POS *tagger*. Tentunya terdapat beberapa penyesuaian yang harus dilakukan. Salah satunya adalah formula probabiltas yang digunakan. Formula yang telah disesuaikan tersebut dapat dilihat di persamaan 2.3.

Model yang telah dilatih akan disimpan di dalam sebuah berkas sebagai *snapshot* agar apabila proses *train* mengalami kegagalan kita tidak perlu mengulangi proses *train* dari awal lagi. Dalam hal ini, untuk setiap 10 *dataset* yang

[3]septinalarasati.com/morphind

[
  {
    “tokens”: [“$token_{1,1}$”, “$token_{1,2}$”, . . . , “$token_{1,m}$”],
    “supertags”: [“$supertag_{1,1}$”, “$supertag_{1,2}$”, . . . , “$supertag_{1,m}$”]
  },
  {
    “tokens”: [“$token_{2,1}$”, “$token_{2,2}$”, . . . , “$token_{2,m}$”],
    “supertags”: [“$supertag_{2,1}$”, “$supertag_{2,2}$”, . . . , “$supertag_{2,m}$”]
  },
  ⋮
  {
    “tokens”: [“$token_{n,1}$”, “$token_{n,2}$”, . . . , “$token_{n,m}$”],
    “supertags”: [“$supertag_{n,1}$”, “$supertag_{n,2}$”, . . . , “$supertag_{n,m}$”]
  }
]

Gambar 3.4: Format JSON *dataset* yang disimpan di dalam berkas.


```
[
  {
    “tokens”: [“Pamungkas”, “menyukai”, “rendang”, “.”],
    “supertags”: [ “NP: pamungkas′”, “(S\NP)/NP: λx.λy. suka(y, x)”, “NP: rendang′”, “Z” ]
  }
]
```


Gambar 3.5: Contoh isi dari berkas *dataset* dalam format JSON.

{
  “starting_point”: 70,
  “rows”: {“$token_1$”: 1, “$token_2$”: 2, . . . , “$token_n$”: N},
  “cols”: {“$token_1$”: 1, “$token_2$”: 2, . . . , “$token_m$”: M},
  “values”: [
    $[value_{1,1}, value_{1,2}, \ldots, value_{1,m}]$,
    $[value_{2,1}, value_{2,2}, \ldots, value_{2,m}]$,
    $\vdots$
    $[value_{n,1}, value_{n,2}, \ldots, value_{n,m}]$
  ]
}

Gambar 3.6: Format JSON untuk menyimpan *snapshot*.

telah diproses, proses penyimpanan *snapshot* akan dilakukan. Adapun format *snapshot* menggunakan JSON yang menyimpan informasi berupa jumlah kalimat terproses dan sebuah tabel yang menyimpan kalkulasi pada saat *snapshot* disimpan. Setelah proses *train* selesai, *snapshot* akan diperbarui kemudian digunakan untuk membuat sebuah berkas baru berupa JSON yang menyimpan *trained model* tersebut sehingga *supertagger* yang dibangun dapat memuat model tersebut tanpa perlu melakukan *train* kembali.

Pada Gambar 3.6, *field* “*starting_point*” menyimpan informasi berupa bilangan cacah untuk memberikan tanda pada elemen ke-i sebaiknya proses *train* dimulai. Dalam hal ini, berdasarkan contoh, sebaiknya dimulai dari elemen ke-70. Selanjutnya, *field* “*rows*” dan “*cols*” masing-masing menyatakan daftar baris dan kolom beserta nomor *index*-nya untuk digunakan oleh *field* “*values*” nantinya. *Field* “*values*” merupakan representasi dari tabel yang memiliki N baris serta M kolom. Adapun “*value*” pada elemen ke-(i, j) menyimpan informasi berupa objek untuk model MaxEnt setelah proses *train* pada “*starting_point*” saat itu. Objek yang dimaksud berupa “konteks” yaitu berisikan daftar k token sebelum dan sesudah token ke-(i, j) juga daftar l *supertag* sebelum dan sesudah *supertag* ke-(i, j). Bahkan, “*value*” dapat pula meyimpan *prefix* dan *suffix* dari token ke- (i, j) tersebut.

3.2.2 Membangun Library Haskell

Implementasi *supertagger* bahasa Indonesia ini akan ditulis dalam bahasa pemrograman Haskell. Haskell dipilih karena kemampuannya dalam mengolah data teks terutama kegiatan *parsing* berkat desain bahasa serta dukungan *library* yang telah tersedia. Secara desain bahasa, Haskell sangat dekat dengan *category theory* sehingga diharapkan *supertag* yang disimpan di dalam program nantinya dapat berupa *algebraic data type* yang semirip mungkin dengan *category theory*. Hal ini menjadi pertimbangan karena tujuan akhir dari riset-riset

yang berhubungan dengan CCG adalah untuk membangun CCG *parser*.

Pembangunan *library* ini sangat penting karena nantinya *library* ini dapat dimanfaatkan untuk membangun program *command line interface* (CLI), *web application*, dan sebagainya. Dalam hal ini, *supertagger* yang dibangun dalam tugas akhir ini akan memiliki dukungan RESTful API agar tidak ada batasan dalam bahasa pemrograman yang harus digunakan untuk memanfaatkan *supertagger* ini. Sangat mungkin pula nantinya terdapat program yang memanfaatkan *library* Haskell ini secara langsung di programnya. Demikian itu, menyediakan *library* untuk suatu bahasa pemrograman (dalam hal ini, Haskell) dirasa perlu.

3.2.3 Menguji Model MaxEnt

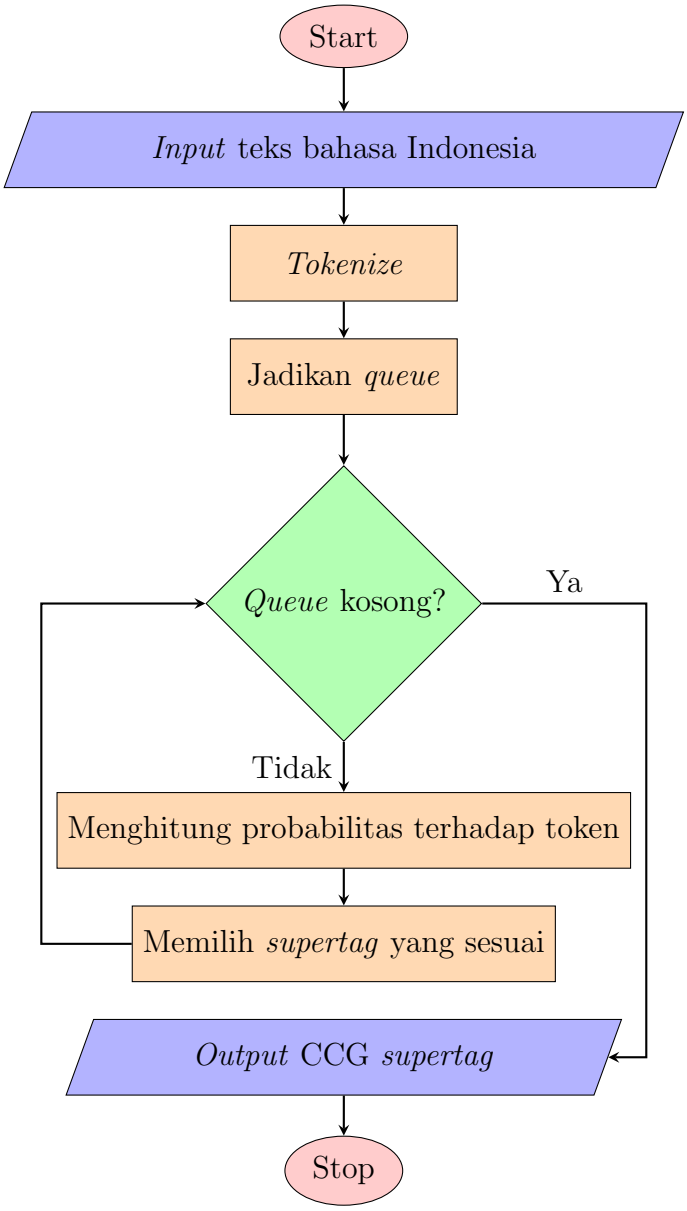
Pengujian model MaxEnt setelah proses *train* dilakukan dengan cara mencocokkan *supertag* yang dihasilkan oleh *supertagger* dengan *supertag* yang telah diberikan secara manual terhadap suatu kalimat yang diberikan. Dengan demikian, setidaknya akan ada dua *dataset* yaitu *data train* dan *data test*. *Data train* dibuat secara semi-otomatis sesuai dengan bagian "Pembuatan Dataset". Selanjutnya, *data test* dibuat secara manual yang mana berjumlah 20% hingga 30% dari jumlah kalimat pada *data train*. Sebagai catatan, untuk kata atau token yang tidak ada di *data train* tetapi ada di *data test* (*unseen words*) akan diberikan *supertag* berupa *category* N.

3.3 Membangun CCG Supertagger Versi CLI

Sebagai langkah awal, program CLI akan dibangun lebih dahulu sebelum versi *web application*-nya rilis. Hal ini agar validasi dapat dilakukan yaitu untuk memeriksa apakah keluaran dari *supertagger* sudah sesuai. Adapun *flowchart* dari program CLI *supertagger* ini terdapat pada Gambar 3.7.

3.3.1 Tokenize

Pertama-tama, *toknize* akan memisahkan antar kalimat yang dipisahkan oleh karakter titik (“.” tanpa tanda kutip) yang diikuti oleh spasi. Selanjutnya, untuk setiap kalimat, *tokenize* akan memisahkan beberapa bagian yang terindikasi sebagai token. Token umumnya berupa kata yang dipisahkan oleh spasi. Namun, token juga dapat berupa karakter khusus seperti karakter tanda koma (“,” tanpa tanda kutip). Dengan demikian, *supertagger* dapat dengan mudah melakukan kalkulasi probabilitas untuk masing-masing token. Proses *tokenize* pada awalnya terlihat mudah. Sayangnya, proses ini cukup *tricky* karena ada beberapa hal tidak umum yang seringkali muncul. Sebagai contoh, “Rp5.000,00” memiliki dua token yaitu “Rp” sebagai *currency* dan “5.000,00” sebagai nominalnya.



Gambar 3.7: Alur kerja proses *tagging*.

3.3.2 Menghitung Probabilitas

Probabilitas untuk sebuah token ke-(i, j) dapat dihitung dengan menggunakan *beam search*. Persamaan yang digunakan adalah persamaan 3.1. Persamaan tersebut memiliki sebuah *category sequence* C serta kalimat S sebagai syarat dari probabilitas tersebut.

$$p(C|S) = \prod_i p(c_i, h_i) \tag{3.1}$$

dimana c_i adalah *category* ke-i dalam *sequence* tersebut dan h_i berupa konteks dari token ke-i. *Beam search* digunakan untuk mendapatkan N *sequence* teratas saat proses *tagging* dilakukan. Dalamn hal ini, $N = 10$ digunakan.

3.3.3 Memilih Supertag

Setelah proses penghitungan probabilitas selesai dilakukan, proses pemilihan *supertag* dilakukan berdasarkan N *sequence* teratas yang disimpan. Berhubung MaxEnt yang digunakan adalah *conditional maximum entorpy*, maka selanjutnya akan dilakukan proses pemilihan *supertag* berdasarkan kondisi-kondisi yang terpenuhi.

Daftar Pustaka

[1] Stephen Clark. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 19–24, Universitá di Venezia, May 2002. Association for Computational Linguistics.

[2] Julia Hockenmaier and Mark Steedman. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn treebank. *Computational Linguistics*, 33(3):355–396, 2007.

[3] Alexander Kurz. Introduction to category theory.

[4] S.D. Larasati, V. Kuboň, and D. Zeman. Indonesian morphology tool (morphind): Towards an indonesian corpus. *Systems and Frameworks for Computational Morphology*, pages 119–129, 2011.

[5] Kiet Van Nguyen and Ngan Luu-Thuy Nguyen. Vietnamese transition-based dependency parsing with supertag features, 2019.

[6] Taraka Rama. Supertagging: Introduction, learning, and application. *CoRR*, abs/1412.6264, 2014.

[7] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging, 1996.

[8] Raul Rojas. A tutorial introduction to the lambda calculus. *CoRR*, abs/1503.09060, 2015.

[9] Mark Steedman. Categorial grammar. Technical report, 1992.

[10] Mark Steedman. A very short introduction to ccg. Technical report, 1996.

Lampiran