October 1997

Efficient Parsing for CCGs with Generalized Type-Raised Categories

Nobo Komagata
University of Pennsylvania

University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-97-16.

Efficient Parsing for CCGs with Generalized Type-Raised Categories

Abstract

A type of 'non-traditional constituents' motivates an extended class of Combinatory Categorial Grammars, CCGs with Generalized Type-Raised Categories (CCG-GTRC) involving variables. Although the class of standard CCGs is known to be polynomially parsable, use of variables suggests more complexity for processing GTRCs. This paper argues that polynomial parsing is still possible for CCG-GTRC from practical and theoretical points of view. First, we show that an experimental parser runs polynomially in practice on a realistic fragment of Japanese by eliminating spurious ambiguity and excluding genuine ambiguities. Then, we present a worst-case polynomial recognition algorithm for CCG-GTRC by extending the polynomial algorithm for the standard CCGs.

Comments

University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-97-16.

Institute for Research in Cognitive Science

Efficient Parsing for CCGs with Generalized Type-Raised Categories

Nobo Komagata

University of Pennsylvania
3401 Walnut Street, Suite 400A
Philadelphia, PA 19104-6228

October 1997

Site of the NSF Science and Technology Center for
Research in Cognitive Science

IRCS Report 97--16

Efficient Parsing for CCGs with Generalized Type-Raised Categories

Nobo Komagata [*]

Department of Computer and Information Science
and
Institute for Research in Cognitive Science
University of Pennsylvania
komagata@linc.cis.upenn.edu

September 23, 1997

Abstract

A type of 'non-traditional constituents' motivates an extended class of Combinatory Categorial Grammars, CCGs with Generalized Type-Raised Categories (CCG-GTRC) involving variables. Although the class of standard CCGs is known to be polynomially parsable, use of variables suggests more complexity for processing GTRCs. This paper argues that polynomial parsing is still possible for CCG-GTRC from practical and theoretical points of view. First, we show that an experimental parser runs polynomially in practice on a realistic fragment of Japanese by eliminating spurious ambiguity and excluding genuine ambiguities. Then, we present a worst-case polynomial recognition algorithm for CCG-GTRC by extending the polynomial algorithm for the standard CCGs.

1 Introduction

In Japanese, as in other SOV languages, a sequence of NPs can form a conjunct as exemplified below.

(1)		John-ga	Mary-o		,		Ken-ga	Naomi-o		tazuneta.
	{	John-NOM	May-ACC	}	CONJ	{	Ken-NOM	Naomi-ACC	}	visited

"John visited Mary and Ken [visited] Naomi."

This type of 'non-traditional constituents' poses a problem to many grammar formalisms and parsers, including those specifically designed for Japanese, e.g., JPSG [Gunji, 1987] and JLE (based on finite-state syntax) [Kameyama, 1995]. Although these systems could be extended to cover the presented case, such extensions would not generalize to the wide range of non-traditional constituency.

Combinatory Categorial Grammar (CCG) has been proposed to account for non-traditional constituency in various areas of syntax [Ades and Steedman, 1982, Dowty, 1988, Steedman, 1985, Steedman, 1996] and also in the related areas of prosody, information structure, and quantifier scope [Steedman, 1991a, Prevost and Steedman, 1993, Prevost, 1995, Hoffman, 1995, Park, 1995, Park, 1996]. The mechanisms independently motivated to cover the wide range of non-traditional constituency can also provide an analysis for the NP-NP sequences in (1) as follows:

[*] I am grateful to Mark Steedman for his numerous suggestions and comments. I would also like to thank Jason Eisner, Anoop Sarkar, K. Vijay-Shanker, Matthew Stone, B. Srinivas, David Weir, and reviewers of IWPT97 for their comments. The research was supported in part by NSF Grant Nos. IRI95-04372, STC-SBR-8920230, ARPA Grant No. N66001-94-C6043, and ARO Grant No. DAAH04-94-G0426. This technical report (IRCS-97-16) is a long version of [Komagata, 1997a] presented at IWPT97.

(2)

$$\begin{array}{ccl} \text{John-ga} & \text{Mary-o} & \\ NP & NP & \\ \Downarrow & \Downarrow & \text{type raising} \\ S/\underline{(S\backslash NP)} & \underline{(S\backslash NP)}/((S\backslash NP)\backslash NP) & \text{functional composition (underlined portions are cancelled)} \\ \cline{1-2} \multicolumn{2}{c}{S/((S\backslash NP)\backslash NP)} & \end{array}$$

Informally, the NPs are assigned *higher-order function* categories associated with the basic category NP, and these functions can compose to derive another function category representing the NP-NP sequence.[1] The two instances of such a category can then be coordinated and take the transitive verb category, $(S\backslash NP)\backslash NP$, as the argument to derive the category S.

A similar type of constituents can also be formed of NPs extracted from different levels of embedding as in the following example:

(3) Japanese: Rinyouzai-wa natoriumu-ni, β syadanzai-wa koukan sinkei kei-ni,
kankei-no aru kouketuatu-no hito-ni kikimasu.
Gloss: {Diuretic-TOP sodium-DAT} & {β blocker-TOP sympathetic nervous system-DAT}
relevance-GEN exist hypertension-GEN person-DAT effective.
Translation: "Diuretic is effective for the person with hypertension related to sodium, and β blocker
[is for the person with hypertension related] to sympathetic nervous system."

This is a simplified version of the sentence taken from the text actually parsed in the experiment discussed below. The original sentence includes three conjuncts, a few adverbs, and another level of embedding. The underlined part is another instance of non-traditional constituent, which includes an extraction from the relative clause. Its structure is schematically shown as follows:

(4) [t_1 hypertension$_2$-GEN person-DAT effective.]
[t_2 t_3 relevance-GEN exist]

Due to functional composition, the category $(S\backslash NP)\backslash NP$, identical to the transitive verb category, can be derived for the above phrase with two extraction sites t_1 and t_3.[2] As in the first example, the NP-NP sequences at the sentence-initial position receive the category $S/((S\backslash NP)\backslash NP)$ and then the sentence category is derived after the combination with the category $(S\backslash NP)\backslash NP$.

Assuming that the competence grammar does not place a bound on the levels of embedding [Miller and Chomsky, 1963], we may have unboundedly-many extractions [Becker et al., 1991, Rambow and Joshi, 1994, Rambow, 1994]. Since no systematic constraint has been identified for the bound on the composition of such extracted constituents, we also assume that these constituents can compose without a limit, potentially resulting in an unboundedly-long NP sequence.[3] As in the case of embedding, the degraded acceptability of long sequences can be attributed to performance issues. These assumptions calls for an infinite set of type-raised categories such as $(S\backslash X_n ... \backslash X_1)/((S\backslash X_n ... \backslash X_1)\backslash NP)$ associated with NP. We capture this polymorphic situation by using variables as in $\mathsf{T}/(\mathsf{T}\backslash NP)$.[4] Although CCGs have also been motivated by psychological aspects, we will be concerned only with syntactic aspects in this paper.

Formal properties of the standard CCGs (CCG-Std) not involving variables are relatively well-studied. CCG-Std is proved weakly-equivalent to Head Grammars (HG), Linear Index Grammars (LIG), and Tree Adjoining Grammars (TAG), known as "mildly context-sensitive grammars" [Vijay-Shanker and Weir, 1994]. CCG-Std is also shown to

[1] The category $S/(S\backslash NP)$ is associated with a higher-order semantics $\lambda P.Pa$ where a is the semantics of NP. Composition of $\lambda P.Pa$ and $\lambda Q.Qb$ (left to right) is $\lambda Q.Qba$. In a sense, this semantics anticipates that of a two-argument predicate (simple or complex), Q.

[2] The use of trace t is for illustration purposes only. The current approach does not assume the notion of gap or movement as the theories which employ trace.

[3] These assumptions need to be examined more carefully and is left for future research [cf. Joshi et al., 1994]. Since this paper assumes that the scope of coordination and adverb modificatioin is bounded, the unbounded composition of type-raised category actually calls for a motivation involving information structure. For example, consider the following pseudo-Japanese:

(5) Q: A-TOP B-NOM C-DAT D-NOM E-FROM who-NOM who-ACC visited-COMP heard-COMP told-COMP thought-Q?

A: $\underbrace{\text{A-TOP B-NOM C-DAT D-NOM E-FROM}}_{theme}\underbrace{\{\text{F-NOM G-ACC}\}\ \&\ \{\text{H-NOM I-ACC}\}}_{rheme}$ visited-COMP heard-COMP told-COMP thought.

The 'theme' part can grow very long as can be seen. The information-structural status of the verb complex is not clear but should not affect the major division between the theme and rheme across where all the categories must be combined prior to the combination of the theme and the rheme.

[4] It is also possible to dynamically type-raise categories as needed. We take lexical approach to avoid the procedural aspects associated with dynamic type raising. Polymorphic forms also have advantage of underspecification which avoids unnecessary proliferation of categories.

be polynomially parsable [Vijay-Shanker and Weir, 1990, Vijay-Shanker and Weir, 1991, Vijay-Shanker and Weir, 1993].[5] A few extension of CCGs, Multiset-CCGs, are also shown to be polynomially parsable [Hoffman, 1995].[6] But, use of variables can change the situation. For example, Hoffman [1993] showed that a grammar involving categories of the form $(\mathsf{T}\backslash x)/(\mathsf{T}\backslash y)$ can generate a language $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n\mathtt{e}^n$, which is no longer equivalent to LIGs. The use of variables in the coordination schema "x^+ **conj** $x \to x$" is also believed to generate a language $(\mathtt{wc})^n$ beyond LIG's power [Weir, 1988]. These suggest that the use of variable introduces more processing for parsing as well. At the same time, the practical side of categorial grammar parsing is not well explored. Since Wittenburg [1986], few practical parsers for categorial grammar have been reported. This paper is concerned with polynomial parsability of the CCGs with a generalized class of type-raised categories (CCG-GTRC) involving variables, from both practical and theoretical points of view. Related questions about the generative power of CCG-GTRC are addressed in [Komagata, 1997b, Komagata, 1997c].

For the practical side, we demonstrate that our CKY-style parser for Japanese appears to run in polynomial time once *spurious ambiguities* (multiple derivations for the equivalent semantics) are eliminated by equivalence check and *genuine ambiguities* (e.g., attachment ambiguity) are excluded. For the theoretical side, we present a worst-case polynomial recognition algorithm, by extending the polynomial algorithm introduced for CCG-Std by Vijay-Shanker and Weir [1990] with one additional condition.

This paper is organized as follows: Section 2 introduces Generalized Type-Raised Categories and defines CCG-GTRC. Section 3 analyzes the performance of the practical parser through an experiment after a discussion about spurious ambiguity elimination. Section 4 presents the theoretical worst-case polynomial recognition algorithm for CCG-GTRC. Appendix A includes the details of the experiment. Appendix B includes an example of the central part of the worst-case polynomial algorithm for CCG-GTRC. A short version of this paper is presented as [Komagata, 1997a].

2 CCGs with Generalized Type-Raised Categories

CCG-GTRC involves the class of *constant categories* (Const) and the class of *Generalized Type-Raised Categories* (GTRC).

A constant (derivable) category c can always be represented as $F|a_n...|a_1$ where F is an atomic *target* category and a_i's with their directionality are *arguments*.[7] We will use "$A,...,Z$" for atomic, constant categories, "$a,...,z$" for possibly complex, constant categories, and '$|$' as a meta-variable for directional slashes $\{/,\backslash\}$. Categories are in the "result-leftmost" representation and associate left. Thus we usually write $F|a_n...|a_1$ for $(...(F|a_n)...|a_1)$. We will call "$|a_i...|a_j$" a *sequence* (of arguments). The length of a sequence is defined as $\Big||a_i...|a_1\Big| = i$ while the nil sequence is defined to have the length 0. Thus an atomic constant category is considered as a category with the target category with the nil sequence. We may also use the term 'sequence' to represent an ordered set of categories such as "$c_1,...,c_2$" but these two uses can be distinguished by the context. The standard CCGs (CCG-Std) solely utilize the class of Const.

GTRC is a generalization of *Lexical Type-Raised Category* (LTRC) which has the form $\overset{T}{\mathsf{T}} \substack{/ \\ \backslash} \left(\mathsf{T} \substack{\backslash \\ /} a\right) |b_i...|b_1$ associated with a lexical category $a|b_i...|b_1$ where $\overset{T}{\mathsf{T}}$ is a variable over categories with the atomic target category T [cf. Dowty, 1988, Steedman, 1996]. The target indication may be dropped when it is not crucial or all the atomic categories are allowed for the target. We assume the order-preserving form of LTRC using the following notation. '$\substack{/ \\ \backslash}$' and '$\substack{\backslash \\ /}$' indicate that either set of slashes in the upper or the lower tier can be chosen but a mixture such as '$/$' and '$/$' is prohibited.[8] GTRC is defined as having the form of $\mathsf{T}\substack{/ \\ \backslash}\underset{\textit{inner sequence}}{(\mathsf{T}\;|a_m...|a_2 \substack{\backslash \\ /} a_1)}\;\underset{\textit{outer sequence}}{|b_n...|b_1}$ resulting from compositions of LTRCs

[5] This situation can be contrasted with Lambek calculus. For example, Hepple [1990] presented normal form parsing but did not analyze its computational complexity. König [1994] showed an exponential parsing algorithm for the calculus, which can be improved to a polynomial one by setting a bound on the number of extractions.

[6] In particular, Pure and Prioritized Multiset-CCGs are shown to be polynomially parsable and Curried Multiset-CCGs are conjectured to be so as well.

[7] Note that the representation $F|a_n...|a_1$ involves meta-variables for object-level constant categories but we will remain ambiguous about this object-meta distinction in this paper.

[8] For a related discussion, see [Steedman, 1991b].

where $m \geq 1$, $n \geq 0$, and the directional constraint is carried over from the involved LTRCs.[9] When the directionality is not critical, we may simply write a GTRC as $\mathsf{T}|(\mathsf{T}|a_m...|a_2|a_1)|b_n...|b_1$. For $gtrc = \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_1$, we define $|gtrc| = n+1$, ignoring the underspecified valency of the variable. Note that the introduction of LTRCs in the lexicon is non-recursive and thus does not suffer from the problem of the overgeneration discussed in [Carpenter, 1991].

These categories can be combined by combinatory rule schemata. Rules of (forward) "generalized functional composition" have the following form:[10]

$$(6)\qquad \begin{array}{ccccc} x/\underline{y} & \blacktriangleright^{k} & \underline{y}|z_k...|z_1 & \longrightarrow & x|z_k...|z_1 \\ \text{functor category} & & \text{input category} & & \text{result category} \end{array}$$

The integer 'k' in this schema is bounded by k_{max}, specific to the grammar, as in CCG-Std.[11] Rules of functional application, "$x/y \blacktriangleright^0 y \rightarrow x$", can be considered as a special case of (6) where the sequence z_i's is nil.[12] The index k may be dropped when no confusion occurs. We say " '$x/\underline{y} \blacktriangleright \underline{y}|z_k...|z_1$' *derives* $x|z_k...|z_1$", and "$x|z_k...|z_1$ *generates* the string of nonterminals '$x/y, y|z_k...|z_1$' or the string of terminals '`ab`' " where the terminals `a` and `b` are associated with x/y and $y|z_k...|z_1$, respectively. The case with backward rules involving '◀' is analogous.

The use of variable for polymorphic type drew attention of researchers working on Lambek calculus [Moortgat, 1988, Emms, 1993]. In particular, Emms showed decidability for an extension called Polymorphic Lambek Calculus. The use of variables in the current formulation is limited to type raising. This reflects the intuition about the choice of rules based on 'combinators' [Steedman, 1988]. But otherwise, we do not assume that categories are wildly polymorphic.[13]

One way to represent this situation is to use two distinct subclasses of the type 'category' constructed as follows:

(7)

	Type construction	Example	
a.	*const* (Target, Arguments)	$F\backslash a_n...\backslash a_1$	$\mapsto$ *const* $(F, \backslash a_n...\backslash a_1)$
b.	*gtrc* (Target, IDir, ISeq, OSeq)	$\overset{T}{\mathsf{T}}\,/(\mathsf{T}\backslash a_m...\backslash a_1)\backslash b_n...\backslash b_1$	$\mapsto$ *gtrc* $(T, /, \backslash a_m...\backslash a_1, \backslash b_n...\backslash b_1)$

Such type construction can be defined in ML style as follows:[14]

```
(8) datatype target A | B | C ...                     (* atomic categories *)
    datatype dir left | right
    datatype complex_cat = Complex of target * arg
         and arg = Arg of dir * complex_cat           (* mutually recursive *)
    datatype seq = Seq of arg list
    datatype cat = Const of target * seq
                 | GTRC of target * dir * seq * seq
```

Then we can define the combinatory rules on instantiated categories. Theoretically, no unification of variable is required although our implementation based on the proposed formalism uses variable unification for convenience. Although dealing with more number of cases is tedious, the technique is straightforward. This leads to a favorable result that CCG-GTRC is not only decidable but also polynomially recognizable.

Inclusion of GTRCs calls for thorough examination of each combinatory case depending on the involved category classes. All the possible combination of category classes are described below. Some cases are subdivided furthermore. Although the traditional categorial representation is used below, the complete description for the constructor format can be defined. A summary of the cases is given in Table 1.

(9) Const▶Const: $a/\underline{b} \blacktriangleright^{k} \underline{c}|d_k...|d_1 \longrightarrow a|d_k...|d_1$

(10) GTRC▶Const

[9] In a sense, GTRCs have two stacks. But these two sequences are not completely independent and does not behave like two stacks for a PDA, which is Turing-Machine equivalent.

[10] Vijay-Shanker and Weir [1994] call the functor and input categories as primary and secondary components, respectively.

[11] Weir [1988] comments that the categorial grammars defined by Friedman and Venkatesan [1986] is more powerful than CCGs due to no bound on k.

[12] Forward functional composition and application are labeled as '$>B^{k}_{(\times)}$' and '$>$', respectively, in [Steedman, 1996].

[13] Recall that we assume conjunctives (if treated categorially) and adverbs are mapped to finite sets of categories.

[14] Note that there is no constant constructor for atomic categories.

a. Functor GTRC has an outer sequence:

$$\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright^k \underline{c}|d_k...|d_1 \longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2|d_k...|d_1$$

Example: $\mathsf{T}\backslash\,(\mathsf{T}/PP)\,/\underline{NP} \blacktriangleright \underline{NP} \rightarrow \mathsf{T}\backslash\,(\mathsf{T}/PP)$

b. Functor GTRC has no outer sequence:

$$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underset{\underline{c_0|c_m...|c_1}}{\overset{c}{\shortparallel}} |d_k...|d_1 \longrightarrow c_0|d_k...|d_1$$

Example: $\mathsf{T}/\underline{(\mathsf{T}\backslash NP\backslash NP)} \blacktriangleright \underline{S\backslash NP\backslash NP} \rightarrow S$

(11) Const▶GTRC

a. $k < |\text{input}|$:

$$a/\underline{b} \blacktriangleright^k \underline{\mathsf{T}|(\mathsf{T}|c_m...|c_1)|d_n...|d_{k+1}}|d_k...|d_1 \longrightarrow a|d_k...|d_1$$

Example: $(S/\,(S\backslash NP\backslash NP))\,\backslash\,(S/\,(S\backslash NP\backslash NP))\,/\underline{(S/\,(S\backslash NP\backslash NP))} \blacktriangleright \underline{\mathsf{T}/\,(\mathsf{T}\backslash NP\backslash NP)}$
$\rightarrow (S/\,(S\backslash NP\backslash NP))\,\backslash\,(S/\,(S\backslash NP\backslash NP))$

b. $k = |\text{input}|$ (and $k \geq 1$):

$$a/\underline{b} \blacktriangleright^k \underline{\mathsf{T}}|(\mathsf{T}|c_m...|c_1)|d_{k-1}...|d_1 \longrightarrow a|(b\ \underset{\text{unbounded}}{\underleftrightarrow{|c_m...|c_1}})|d_{k-1}...|d_1$$

Example: $S/\underline{S} \blacktriangleright \underline{\mathsf{T}}/\,(\mathsf{T}\backslash NP\backslash NP) \rightarrow S/\,(S\backslash NP\backslash NP)$

c. $k > |\text{input}|$ (and $k \geq 2$):

$$a/\underline{b} \blacktriangleright^k \underline{\mathsf{T}_0}|\mathsf{T}_1|(\mathsf{T}_0|\mathsf{T}_1|c_m...|c_1)|d_{k-2}...|d_1 \longrightarrow a\ \underset{\substack{\uparrow\\ \text{residual}}}{|\mathsf{T}_1}\ |(b\ |\mathsf{T}_1\ \underset{\text{unbounded}}{\underleftrightarrow{|c_m...|c_1}})|d_{k-2}...|d_1{}^{15}$$

(12) GTRC▶GTRC

a. Functor GTRC has an outer sequence *and* $|\text{input}| > k$:

$$\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright^k \underline{\mathsf{U}|(\mathsf{U}|c_p...|c_1)|d_n...|d_{k+1}}|d_k...|d_1$$

$$\longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2|d_k...|d_1$$

b. Functor GTRC has an outer sequence *and* $|\text{input}| = k$ (and $k \geq 1$):

$$\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright^k \underline{\mathsf{U}}|(\mathsf{U}|c_p...|c_1)|d_{k-1}...|d_1$$

$$\longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2|(b_1\ \underset{\text{unbounded}}{\underleftrightarrow{|c_p...|c_1}})|d_{k-1}...|d_1$$

c. Functor GTRC has an outer sequence *and* $|\text{input}| < k$ (and $k \geq 2$):

$$\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright^k \underline{\mathsf{U}_0}|\mathsf{U}_1|(\mathsf{U}_0|\mathsf{U}_1|c_p...|c_1)|d_{k-2}...|d_1$$

$$\longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2\ \underset{\substack{\uparrow\\ \text{residual}}}{|\mathsf{U}_1}\ |(b_1\ |\mathsf{U}_1\ \underset{\text{unbounded}}{\underleftrightarrow{|c_p...|c_1}})|d_{k-2}...|d_1$$

d. The functor GTRC has no outer sequence *and* $|\text{input}| > k$:

(i) T spans greater than U ($\mathsf{T} = \mathsf{U}|(\mathsf{U}|c_p...|c_1)|d_n...|d_{k+m+1}$):[16]

$$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underline{\underbrace{\mathsf{U}|(\mathsf{U}|c_p...|c_1)|d_n...|d_{k+m+1}}_{\mathsf{T}}\underbrace{|d_{k+m}...|d_{k+1}}_{|a_m...\backslash a_1}}|d_k...|d_1$$

$$\longrightarrow \mathsf{U}\underset{\text{inner seq of GTRC}}{\underleftrightarrow{|(\mathsf{U}|c_p...|c_1)}}\ |d_n...|d_{k+m+1}|d_k...|d_1$$

Example: $\mathsf{T}/\underline{(\mathsf{T}\backslash NP)} \blacktriangleright \underline{\mathsf{U}/(\mathsf{U}\backslash PP)\backslash NP} \rightarrow \mathsf{U}/(\mathsf{U}\backslash PP)$

[15] T could also be decomposed into $\mathsf{T}_0|\mathsf{T}_k...|\mathsf{T}_1$ for a larger k but all of them share the same characteristics with the above scheme.

[16] We only consider the most general unifier.

Case	Functor cat		Input cat		Result cat		
	Class	Outer	Class	$\lvert\text{input}\rvert \overset{<}{\underset{>}{=}} k$	Class	Residual	Unbounded
		seq				variable	const argument
9	Const	-	Const	$\geq$	Const	no	no
10a	GTRC	yes	Const	$\geq$	GTRC	no	no
10b	GTRC	no	Const	$\geq$	Const	no	no
11a	Const	-	GTRC	$>$	Const	no	no
11b	Const	-	GTRC	$=$	Const	no	possible
$*$ 11c	Const	-	GTRC	$<$	neither	yes	possible
12a	GTRC	yes	GTRC	$>$	GTRC	no	no
12b	GTRC	yes	GTRC	$=$	GTRC	no	possible
$*$ 12c	GTRC	yes	GTRC	$<$	neither	yes	possible
12di	GTRC	no	GTRC	$>$	GTRC	no	no
12$d$ii	GTRC	no	GTRC	$>$	Const	no	no
12e	GTRC	no	GTRC	$=$	GTRC	no	no
$*$ 12f	GTRC	no	GTRC	$<$	neither	yes	possible

Table 1: **Combinatory Cases for CCG-GTRC**

(ii) T spans no greater than U ($\mathsf{T}|a_m...|a_{m-j+1} = \mathsf{U}$):

$$\underline{\mathsf{T}/(\mathsf{T}|a_m...|\underset{\underset{a_{m-j,0}|a_{m-j,p}\cdots|a_{m-j,1}}{\shortparallel}}{a_{m-j}}...|a_2\backslash a_1)} \blacktriangleright^k \underline{\underbrace{\mathsf{U}_0}_{\mathsf{T}}\underbrace{|\mathsf{U}_j...|\mathsf{U}_1}_{|a_m\cdots}\underbrace{|(\mathsf{U}_0|\mathsf{U}_j...|\mathsf{U}_1|c_p...|c_1)}_{a_{m-j}=F|a_{(m-j,q)}\cdots|a_{(m-j,1)}}\underbrace{|d_n...|d_{k+1}}_{\cdots\backslash a_1}}|d_k...|d_1$$

$$\longrightarrow F|a_{(m-j,q)}\cdots|a_{(m-j,q-j-p)}|d_k...|d_1 \qquad \text{where } q \geq j+p$$

Example: $\mathsf{T}/\underline{(\mathsf{T}\backslash(S/NP))} \blacktriangleright \underline{\mathsf{U}\backslash(\mathsf{U}/NP)} \rightarrow S$

e. The functor GTRC has no outer sequence *and* $|\text{input}| = k$ (and $k \geq 1$):

$$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underline{\mathsf{U}}|(\mathsf{U}|c_p...|c_1)|d_{k-1}...|d_1 \longrightarrow \mathsf{T}|(\mathsf{T}\underset{\text{inner seq of GTRC}}{\underleftrightarrow{|a_m...|a_2\backslash a_1|c_p...|c_1}})|d_{k-1}...|d_1$$

Example: $\mathsf{T}/\underline{(\mathsf{T}\backslash NP)} \blacktriangleright \underline{\mathsf{U}}/(\mathsf{U}\backslash NP) \rightarrow \mathsf{T}/(\mathsf{T}\backslash NP\backslash NP)$

f. The functor GTRC has no outer sequence *and* $|\text{input}| < k$ (and $k \geq 2$):

$$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underline{\mathsf{U}_0}|\mathsf{U}_1|(\mathsf{U}_0|\mathsf{U}_1|c_p...|c_1)|d_{k-2}...|d_1$$

$$\longrightarrow \mathsf{T}\,|\underset{\substack{\uparrow\\ \text{residual}}}{\mathsf{U}_1}|(\mathsf{T}\,|a_m...|a_2\backslash a_1|\mathsf{U}_1\,\underset{\text{unbounded}}{\underleftrightarrow{|c_p...|c_1}})|d_{k-2}...|d_1$$

The three cases indicated by '$*$' in Table 1 introduce categories which are neither Const nor GTRC due to the residual variables. This is an unintended, accidental use of functional composition. The closure of the system must be maintained by excluding these cases by the following condition:

(13) **Closure Condition**: The rule "$x \blacktriangleright^k y$" must satisfy $|y| \geq k$.

Note that the distinction between constant categories and GTRCs must be made. This condition is particularly important for implementation since the residual variables can behave beyond our imagination and the parser must be able to compute the length of a category distinctively for constant categories and GTRCs.

We are now in the position to define the class of CCG-GTRC:[17]

Definition 1 A CCG-GTRC is a six tuple $(V_N, V_T, S, \mathcal{T}, f, R)$ where

- V_N is a finite set of nonterminals (atomic categories)

[17] We may use the term 'CCG-GTRC' ambiguously between for the class of grammars and for a grammar instance.

- V_T is a finite set of terminals (lexical items, written as $\mathtt{a}, ..., \mathtt{z}$)
- S is a distinguished member of V_N
- $\mathcal{T}$ is a countable set of variables[18]
- f is a function that maps elements of V_T to finite subsets of "Const ∪ LTRC"[19]
- R is a finite set of rule instances of Generalized Functional Composition observing Closure Condition (i.e., those summarized in Table 1 except for those with '$*$').[20]

3 Progress Towards a Practical Parser for CCG-GTRC

This section investigates the performance of the experimental parser and demonstrates that it runs polynomially in practice. For both the practical parser and the theoretical algorithm, we use CKY-style parsing scheme [Aho and Ullman, 1972]. In addition to the use of CKY-table for recognition of the start category, we associate semantic representation for each category and derive the semantics in a single pass. We will focus on 'category-only' case for purely syntactic analyses but it should be noted that the parser can derive semantics and is not just a recognizer. We start with a discussion about spurious ambiguity elimination techniques.

3.1 Spurious Ambiguity Elimination

We first define the types of ambiguities referred to in this paper:

(14) *a*. Categorial ambiguity: Availability of multiple categories (lexical/derivational), e.g., $\{S, NP\}$.

b. Spurious ambiguity: Multiple derivations of the same category which are semantically *equivalent* [Wittenburg, 1986], e.g., "John visited Bill." has two derivations of S (left and right branching) in CCG with the identical semantics '$(visited\ bill\ john)$'.

c. Genuine ambiguity:

(i) Lexico-semantic ambiguity: Multiple semantic assignments to a single lexical category:

(ii) Attachment ambiguity: Multiple derivations of the same category with *distinct* semantics. E.g., PP attachment.

Since spurious ambiguity is often accused of as the source of inefficient parsing, CCG parsers must implement some means of spurious ambiguity elimination. We review three classes of approaches: (i) syntactic, (ii) semantic, and (iii) those which do not belong to the previous two.

The syntactic approaches eliminate 'spurious derivations' which are not 'the normal form'. This approach blends naturally with Poly-Std. Vijay-Shanker and Weir [1990] includes a mechanism of spurious ambiguity check during a stage after recognition. Hendriks [1993] and König [1994] worked on Lambek calculus which does not have functional composition as a primitive rule. Hepple and Morrill [1989] cover a subset of the current formalism but do not have the mixed instances of function composition nor type raising. Eisner [1996] covers a wider range of CCGs but the case including type raising remains to be shown correct. Labeled deduction [Morrill, 1994] has a means to interpret semantics syntactically but normal form deduction must be adjusted.

Karttunen [1986] proposed the following semantic method. For each new derivation, discard it if its semantics is *equivalent to* (or mutually subsumes) that of some entry with the same category already in the cell.[21] This directly enforces the definition of spurious ambiguity and does not depend on the syntax. Note that 'equivalence' depends on

[18] Each instance of GTRC must be assigned a new variable when the GTRC is instantiated at a particular string position in order to avoid unintended variable binding.

[19] Our definition does not include the empty string in the domain of f as in [Vijay-Shanker and Weir, 1993] but unlike [Vijay-Shanker and Weir, 1994].

[20] Due to the introduction of GTRC, the rule instances may involve variables even at the first argument of the functor category and at the input category.

[21] Shieber [1986] contains a detailed discussion of subsumption.

the semantic representation [Thompson, 1991]. For the case where the semantics is represented in λ-calculus, equivalence is not computable [Paulson, 1991]. For the case of feature structure, equivalence is defined as alphabetic variants and characterized by the isomorphism between the structures [Carpenter, 1992]. Our case corresponds to the latter although the equivalence check on predicate-argument structures are term unification rather than graph unification for the feature structure.

Among the third type, Pareschi and Steedman [1987] is a hybrid but the published algorithm has been shown to be incomplete [Hepple, 1987]. Wittenburg and Wall [1991] use a compilation scheme but the crucial notion of flexible constituency is compromised.

To be precise, the definition of spurious ambiguity for syntactic methods is not the same as our definition given above. Since syntactic methods are insensitive to semantics, they cannot distinguish genuine (attachment) and spurious ambiguities and eliminate both of these. With semantic equivalence check, we can adjust the way the genuine ambiguity is handled. While equivalence check in its original form is sensitive to genuine ambiguity, equivalence check applied only to category (ignoring semantics) is insensitive to genuine ambiguity much like syntactic methods. We refer to these two cases as (i) *category+semantics* and (ii) *category-only*. Both cases are explored in the experiment in the following subsection.

Among the presented methods, the only proposal immediately compatible with the present formalism is Karttunen's semantic equivalence check. Let us review some arguments against this approach. Eisner [1996] argued that a sequence of categories exemplified by "X/X ... X ... $X\backslash X$" can slow down the parser exponentially. Note that we assume that X/X and $X\backslash X$ are 'modifiers' of X with distinct semantics. But this is an instance of genuine ambiguity, a problem to any parser. Note that for syntactic methods, this appears as spurious ambiguity and the semantic processing is simply left for a later stage. Wittenburg [1987] objected to the cost of equivalence check. But an equivalence check (for our semantics) is inherently easier than the general case of subsumption check, the latter requiring the *occurs check* for soundness [Pereira and Shieber, 1987]. Hepple and Morrill [1989] have raised another objection. While syntactic methods detect spurious ambiguities before deriving a result, equivalence checking needs to compare the derived result with every entry in the current table cell. However, the cost associated with the semantic method depends on how many genuinely ambiguous entries are in the cell but not on the number of spuriously-ambiguous entries (they are eliminated as soon as they are derived and do not accumulate). This does not introduce additional complexity specific to spurious ambiguity check. Further, once semantics is involved, it is not possible to distinguish between spurious and genuine ambiguities unless we actually check the involved semantics.

3.2 Experiment

We now look at the results of a pilot experiment done on Sun Ultra E4000 2x167MHz Ultraspacs with 320MB memory running SunOS 5.5.1. The program (100KB approx., about a half is the grammar) was written in Sicstus Prolog Ver. 3 and CPU time was measured by Sicstus' built-in predicate `statistics`. The details of the experiment are included as Appendix A. We parsed 22 contiguous sentences (6 paragraphs) in Japanese in a section arbitrarily chosen from "Anata-no Byouin-no Kusuri (Your Hospital Drugs) 1996" by Tadami Kumazawa and Ko-ichi Ushiro. The romanized sentences are partially-segmented to the word level but the verb inflection and suffixes are considered as a part of the word. The average number of words in a sentence is 20 and the longest sentence contains 41 words. The sentences are realistically difficult, and include complex clauses (relative and complement), coordination (up to 4 conjuncts), nominal/verbal modifications (adjectives/adverbs), scrambling, and verb argument dropping.

The parser is based on a CKY algorithm equipped with Karttunen's equivalence check for spurious ambiguity elimination but without the worst-case polynomial algorithm introduced in the next section.[22] LTRCs are assigned to words by lexical rules and GTRCs are restricted to unidirectional forms. Coordination is handled by special trinomial rules [Steedman, 1996] with a few categorial features added to limit the coordination involving multiple constituents only to the left-branching structure. Verb argument dropping is handled by lexical rules which change the verb valency. Morphological analysis is a complete substring match and the results are dynamically 'asserted' among the code. About 200 lexical entries are asserted after parsing the 22 sentences. Currently, morphological analysis takes about 0.2 seconds per word on average and needs improvement.[23] The output of a parse is an enumeration of the final result

[22] Earlier applications of a CKY-style algorithm to CCG parsing include [Pareschi and Steedman, 1987].

[23] Whitelock [1988] has worked on morpho-syntax of Japanese in categorial framework. Some recent work on morphology includes [Hisamitsu and Nitta, 1994], [Tanaka et al., 1993].

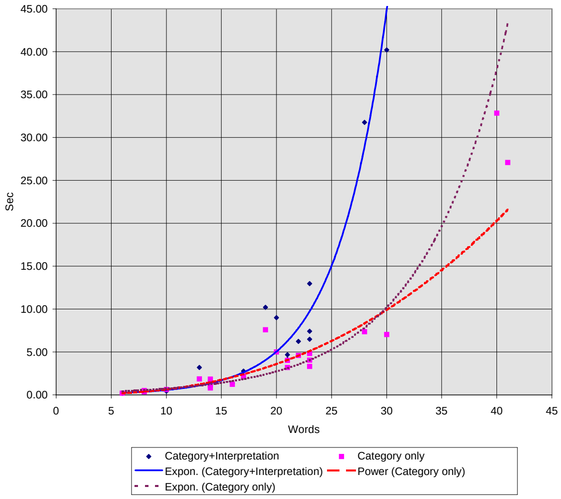

Figure 1: Linear scale

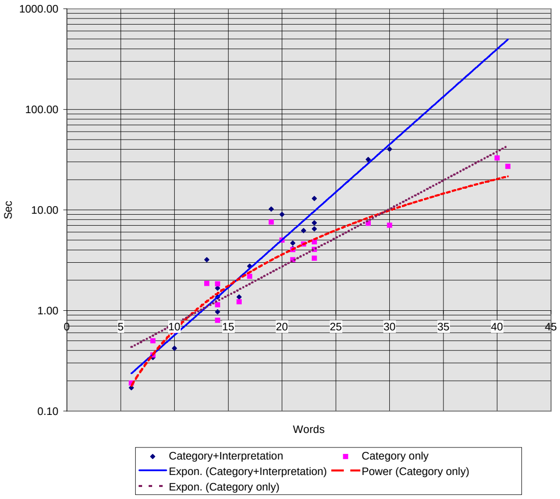

Figure 2: Log scale

categories associated with the features and the semantics as seen below (Sentence 7).

```
(15) itumo 95mmHg o koeru baai_wa tiryou ga hituyoudesu.
     always num(95mmHg) -ACC exceed in_case treatment [-NOM,-CONJv] necessary
       Cat:
         SS: s
         Fs: []
         PA: (in_case always((exceed num(95mmHg) $1)) (necessary treatment))
       Cat:
         SS: s
         Fs: []
         PA: always((in_case (exceed num(95mmHg) $2) (necessary treatment)))
     CPU time: 280 ms  Elapsed: 320 ms  Words: 8  Solutions: 2
```

The unresolved pronoun is shown as '$\$n$' where $n \in \mathbb{N}$. The ambiguity regarding adverbial modification is left unresolved. The implementation has a simplified treatment of quantifiers and scope ambiguity too is left unresolved.

We consider the following two cases: (i) category-only and (ii) category+semantics. As we have discussed in the previous subsection, the application of equivalence check to the category-only case not only eliminates spurious ambiguities but also provide a result without genuine ambiguities.

Let us start with the analysis of the category-only case.[24] This case corresponds to the situation involving syntactic methods and also the polynomial algorithms introduced in the next section. The results are shown in Figure 1 (linear scale) and 2 (log scale). Both exponential ($y = 0.1963 \times 1.14^n$) and polynomial ($y = 0.002 \times n^{2.496}$) regression lines calculated by Microsoft Excel are provided. Although it is often easier to fit either an exponential or a polynomial curve on a log-scale graph, the data do not seem to be enough for such a conclusion. To see how the experiment might extend to the case with words longer than 45 words, we parsed pseudo-long sentences. That is, some of the test sentences are conjoined to form long sentences. Although these are semi-fabricated data, most long sentences are in fact the results of coordination. Thus natural data are expected to behave similarly rather than differently from our pseudo-long sentences. The results are shown in Figures 3 and 4. The polynomial curves ($y = 0.0017 \times n^{2.5616}$) seem to represent the data better than the exponential curve ($y = 0.4375 \times 1.09^n$), especially on the log-scale graph. Since the data is sparse, we do not attempt to obtain a significant statistic analysis for these and simply eye-fit the data. With these qualifications, we conclude that the performance appears no worse than n^3. The result also shows that categorial ambiguities still present in the parses are in practice within this bound.

A few remarks are in order. We compared our results with the following experiment to see how the figures

[24] Category-only is the case also corresponding to the spurious ambiguity check of syntactic methods.

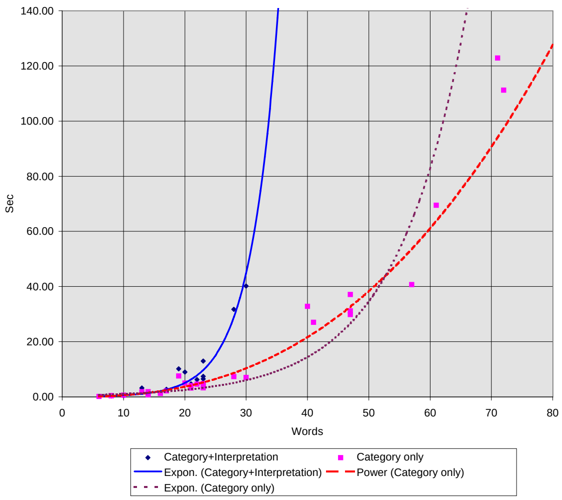

Figure 3: Linear scale (extended)

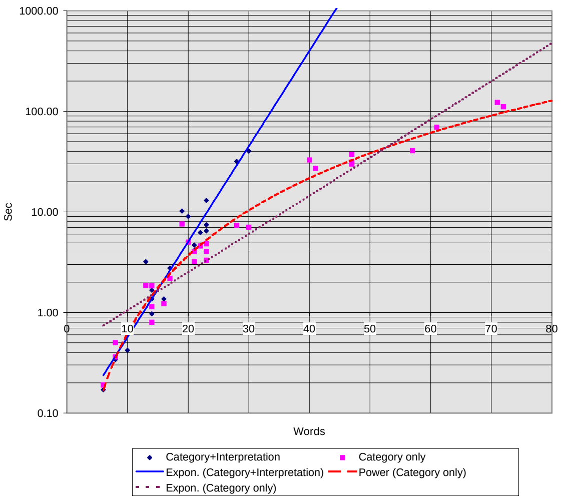

Figure 4: Log scale (extended)

stand. Tanaka and Ueki [1995] report that the LR-based syntactic analysis of a 19-word sentence in Japanese took 3.240sec.[25] The range of CPU times for our sentences with 19-23 words is between 3 to 8sec (category-only case). The performance of our parser seems to be within a comparable range.

Another point is that the effect of spurious ambiguity check is immediate. Without the check, only the sentences with 10 or fewer words were parsed. Under this condition, the maximum number of cell entries easily exceeds 300 for longer sentences, which resulted in out-of-space errors. We thus confirmed that the exponential effect of spurious ambiguity is well controlled by semantic equivalence check.

The above conclusion naturally remains qualified by the small scale of the experiment reported here. But, the test sentences are reasonably representative and relatively challenging. They vary in sentence length and complexity and span the space we may typically encounter. With additional data, it is reasonable to expect that the missing points will be filled and statistic significance will be obtained. It is also reasonable to believe that the experiment with pseudo-long sentences characterizes the kind of complexity that will be found in natural data.

Since one of the advantages of CCG parsers is the ability to derive semantics along with syntactic structure, the results of the category+semantics case is of special interest.[26] The situation naturally looks quite different. The exponential regression line $t = 0.0638 \times 1.24^n$ (Figures 1 and 2) seems to fit the data closely. In fact, the two longest sentences with 40 and 41 words results in out-of-space errors. Since spurious ambiguities are eliminated by equivalence check and categorial ambiguity is only polynomial as shown in the category-only experiment, the exponential slow down is due exclusively to genuine ambiguity. Genuine ambiguity is a major problem for our parser as it is for any parser. We noticed that the longest two sentences become parsable if modifications across the top-level coordination are prohibited by assigning a special category to the sentential conjunctive. This kind of technique improves the performance, usually without affecting the completeness. But of course, we need a more principled idea of how to deal with such a case. The following example (adopted from Sentence 8) shows how easily genuine ambiguities can grow:

(16) *a*. Modification ambiguities:

n-TOP $\underset{\rightarrow}{\textbf{adj}}$ $\underline{\underline{\underline{\textbf{n}_1}\text{-GEN } \textbf{n}_2}}$-NOM v-coord, $\underset{\rightarrow}{\textbf{adv}_1}$ $\underset{\rightarrow}{\textbf{adv}_2}$ $\underline{\underline{\textbf{v}_1}\text{-COMP-TOO v}_2}$

2-way ambiguous 3-way ambiguous

b. Coordination ambiguities:

$\underline{\text{n-TOP } \underline{\underline{\text{adj n-GEN n-NOM v-}\textbf{coord}\text{, adv adv v}}}\text{-COMP-TOO v}}$

Each case involves adjective modification ambiguity (2 cases each).

[25] The word count is based on our criteria. Other details are ignored for now.

[26] The current implementation *enumerates* the derivations.

$\underline{\text{n-TOP } \underline{\text{adj n-GEN n-NOM v-}\textbf{coord}\text{, adv adv v-COMP-TOO v}}}$

Each case involves both adjective and adverb modification ambiguities (2×3 cases each).

c. Total ambiguities: $2 \times 2 + (2 \times 3) \times 2 = 16$

The worst case (Sentence 4) resulted in 96 parses. Since semantics is not considered by Poly-GTRC, Poly-GTRC does not affect the above situation.

4 Worst-Case Polynomial Recognition Algorithm

Although the implemented parser runs efficiently (the category-only case), its worst-case performance is still exponential due to categorial ambiguity. This section presents a worst-case polynomial recognition algorithm for a subclass of CCG-GTRC (Poly-GTRC) by extending the polynomial algorithm of Vijay-Shanker and Weir [1990] for CCG-Std (Poly-Std). We will observe below that the crucial property of CCG-Std employed by Poly-Std can be extended to the subclass of CCG-GTRC with an additional condition. Let us start with a brief review of the intuition behind Poly-Std and then move on to Poly-GTRC. Note that Poly-Std has the second stage of structure building but we concentrate on the more critical part of recognition.

4.1 Polynomial Algorithm for CCG-Std

First, observe the following properties of CCG categories:

(17) *a*. The length of a category in a cell can grow proportionally to the input size.

b. The number of categories in a cell may grow exponentially to the input size.

For example, consider a lexicon $f = \{(\mathtt{a}, S/NP/S),\ (\mathtt{a}, S/PP/S)\}$. Then, for the input "$\underset{n}{\underleftrightarrow{\mathtt{a}....\mathtt{a}}}$", the top CKY-cell includes 2^n combinations of categories like $S \underset{n}{\underleftrightarrow{\begin{Bmatrix} /NP \\ /PP \end{Bmatrix} \ldots \begin{Bmatrix} /NP \\ /PP \end{Bmatrix}}} /S$ derived by functional composition. Thus, we have exponential worst-case performance with respect to the input size.

The idea behind Poly-Std is to store categories as if they were some kind of linked list. Informally, a long category $F|a_n...|a_2|a_1$ is stored as '$F\boxed{\text{this portion is } \textit{linked} \text{ in a cell } [p,q] \text{ with } \textit{index } |a_2}\,|a_1$'. A crucial point here is that the instances of target category F and arguments a_2 and a_1 are *bounded*. We will come back to this point in the next subsection. The pair $[p,q]$ can be represented as a n^2 matrix. Thus by setting up n^2 subcells in each CKY-table cell, we can represent a category in a finite manner.

The effectiveness of such a representation comes from the fact that CCG rule application does not depend on the entire category. Namely, in order to verify "$F \underbrace{|a_n...|a_2}_{\star} |\underline{a_1} \blacktriangleright^k \underline{b_0}|b_k...|b_1 \to F \underbrace{|a_n...|a_2}_{\star} |b_k...|b_1$", the sequence marked by '★' does not need to be examined. Thus, for the functor category, we only need to check F and a_1 available in the current cell. In addition, since b_0 must be unified with a bounded a_1, and k is also bounded by k_{max}, the entire input category is bounded and thus can be stored in the current cell. Therefore, the proposed representation does not slow down this type of process. When the result category exceeds a certain limit, we leave the excessive portion right in the original cell and set up a link to it.

One complication is that when an argument (e.g., a_1 in the above example) is canceled, we may have to restore a portion of the category from the linked cell (as the 'index' for the cell is required). We need to scan the linked cells and find the categories with the same index from n^2 subcells. Even though there may be multiple such categories, all of them can be restored in one of n^2 subcells associated with the result category. This case dominates the computational complexity but can be done in $O\left(n^4\right)$. Since this is inside i, j, k of CKY-style loop, the overall complexity is $O\left(n^7\right)$, which can be improved to $O\left(n^6\right)$ by rearranging the loops. The following is an informal description of the algorithm:

(18) **Poly-Std algorithm**:

a. Initialize: set up lexical categories

b. Main loop: for $1 \leq i < j \leq n$,
for $i \leq k < j$, apply rule schemata as follows:

Conditions		Case	Intuition
$\|result\|$	Link info		
$< limit$	none	No link	$F\|a_n\ldots\|a_1$
$< limit$	available	Pass link info	$F\boxed{}\|a_i\ldots\|a_1 \rightarrow F\boxed{}\|a_i\ldots\|a_1$
$\geq limit$	either	Set up a new link	$F\ \boxed{\|a_n\ldots\|a_{i+1}}\ \|a_i\ldots\|a_1$ $\downarrow$ $\boxed{}$
$= 0$	available	Restore the linked info	$\boxed{\|a_i\ldots\|a_1}$ $\downarrow$ $F\ \boxed{}$

4.2 Polynomial Algorithm for CCG-GTRC

We first note that there are cases where a crucial property of CCG-Std cannot be maintained in CCG-GTRC. The property is that arguments of derived categories are bounded. Although there might be a polynomial algorithm for CCG-GTRC which does not depend on this property, we pursue a straightforward extension of Poly-Std with an additional condition on the rules.[27]

The problematic cases are (11*b*) and (12*b*) in Table 1. As we have motivated in Introduction, the inner sequence of GTRC must be allowed to grow unboundedly long (otherwise, the resulting grammar becomes equivalent to CCG-Std). Then an argument can grow unboundedly long. But we do not want to exclude Cases (11*b*) and (12*b*) entirely since a case involving a sentential adverb may be possible as follows: "$S/S \blacktriangleright \mathsf{T}|(\mathsf{T}|A|B)$". The present approach is to place the following condition on the rule application:

(19) **Bounded Argument Condition**: For the rule "$x \blacktriangleright^k y$" where (i) x is not a GTRC with $|x| = 1$ and (ii) $|y| = k$, y must be instantiated.

This includes the cases (11*b*) and (12*b*). By this condition, we have the following property:

(20) The set of arguments of constant categories and the set of arguments of the inner and outer sequences of GTRC are all finite.

That is, by considering the inner sequence at the same level as the outer sequence of a GTRC or the arguments of a constant category, we can maintain the property analogous to the one found for CCG-Std.[28]

Now, consider the subclass of CCG-GTRC constrained by the Bounded Argument Condition. In the rest of this section, we will concentrate on this subclass.

Poly-GTRC is an extension to Poly-Std. The basic organization of the algorithm is analogous to Poly-Std. We use the same $n^2 \times n^2$ CKY-style table and a similar representation for constant categories. But we need to deal with GTRCs in polynomial time as well. First, let us examine two representative cases of rule applications since this reveals the necessary conditions for polynomial parsing.

The inner sequence of GTRC can grow as a result of Case (12*e*) , "GTRC▶GTRC", repeated below:

(21) $\mathsf{T} \underset{\uparrow}{/} \underline{(\mathsf{T}|a_m\ldots|a_2\backslash a_1)} \blacktriangleright^k \underline{\mathsf{U}}|(\mathsf{U}|c_p\ldots|c_1)|d_{k-1}\ldots|d_1 \longrightarrow \mathsf{T}|(\mathsf{T}|a_m\ldots|a_1|c_p\ldots|c_1)|d_{k-1}\ldots|d_1 \quad (k \geq 1)$

The only information needed to determine if the rule is applicable is the directionality of the slash indicated by '↑'. Thus we do not actually need to know the inner sequence of the functor or input categories. The inner sequence of the result GTRC can thus be represented as two links to the functor and input categories. This link information virtually

[27] The length of the argument is still bounded by $O(n)$ in CCG-GTRC since the only source of unboundedness is GTRC inner sequences. If every argument can be represented in some finite manner with link information similar to the one used for the Poly-Std, polynomial recognition might be possible.

[28] Note that the Bounded Argument Condition is not implemented in our parser. The inner sequence of GTRC does not grow unboundedly in practice and thus the arguments of categories do not grow unboundedly either.

encodes a kind of grammar for deriving the inner sequence and is thus considered as an application of structure sharing [Billot and Lang, 1989, Dymetman, 1997]. The outer sequence can be represented in a way similar to the argument of constant category. Although there may be exponentially-many GTRCs associated with each CKY cell, the number of cell entries is bounded by the link destinations of the inner sequence and the finite representation for the outer sequence.

Next, consider Case (10*b*), "GTRC▶Const". We need to show that the unification process of the underlined portions can be done in polynomial time. As the first approximation, consider this process as an iteration of (backward) functional application of the form "$\underline{a_i} \blacktriangleleft c_0|c_m...|\underline{c_i}$" for $i = 1$ to m where a_i and c_i are canceled. But recall that in general, we only store a finite portion of both the functor and the input categories in the current cell and the remaining information must be restored through the links. The restoration of the information could cost exponential time since there may be multiple links to lower locations at any point. Therefore it is crucial that we proceed from $i = 1$ to m so that no enumeration of all the instances of $c_i, ..., c_1$ and $a_i, ..., a_1$ in (10) is actually generated. The traversal of the link from c_1 and a_1 may introduce sets of categories C_i and A_i for each position of $i \geq 2$, as schematically shown below.

(22)
$$\begin{tikzcd}
C_m \arrow[dd, leftrightarrow] & \cdots & C_2 \arrow[dd, leftrightarrow] & & \{c_1\} \arrow[dd, leftrightarrow] \\
& & & {} \arrow[l] & \\
A_m & \cdots & A_2 & & \{a_1\}
\end{tikzcd}$$

Note that each set C_i and A_i are bounded. This is the crucial point we needed the Bounded Argument Condition. Now, suppose that an element in C_i is canceled with some elements in A_i. We can proceed to the next set C_{i+1} where the elements in C_{i+1} are obtained by traversing the links from the canceled elements in C_i. Notice that the recovery process may encounter GTRCs as a part of derivation. There are three such cases: (i) (10*b*): GTRCs can be ignored since they do not affect the recovery process, (ii) (11*a*), (12*a*): GTRCs are bounded, and (iii) (12*d*ii): process shifts to GTRC recovery shown below.

Once we move from C_i to C_{i+1}, the history of cancellation can be forgotten, as in the case of iterative functional application in Poly-Std. Thus even though we have potentially exponential instances of $c_i, ..., c_1$, the traversal of this side can be done step-by-step without suffering the exponential effect.

The traversal of A_i's is more challenging. The availability of a_i for cancellation with some c_i depends on the history of the cancellation of $a_{i-1}, ..., a_1$. Actually, it depends on the *tree structure* exactly encoded by the structure sharing technique. The 'GTRC recovery algorithm' will be introduced below to handle this situation in polynomial time.

The other cases are variation of the previous one. The "Const▶Const" case can be processed as in Poly-Std. Next, consider the "GTRC▶Const" case.

(23) $\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright \underline{c}|d_k...|d_1 \longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2|d_k...|d_1$

This case can actually be handled in a way similar to the "Const▶Const" case. The only point is that the representation of GTRC must be bounded to avoid exponential combination of a_i's. We will come back to the representation of GTRC below.

The "Const▶GTRC" cases are simpler.

(24) *a.* $a/\underline{b} \blacktriangleright \underline{\mathsf{T}|(\mathsf{T}|c_m...|c_1)|d_n...|d_{k+1}}|d_k...|d_1 \longrightarrow a|d_k...|d_1$

b. $a/\underline{b} \blacktriangleright \underline{\mathsf{T}}|(\mathsf{T}|c_m...|c_1)|d_{k-1}...|d_1 \longrightarrow a|(b|c_m...|c_1)|d_{k-1}...|d_1 \quad (k \geq 1)$

We need to recover the contents of the GTRC in both cases. The GTRC in (24*a*) is bounded since category b in the functor category is bounded. The one in (24*b*) is bounded by k_{max} (for $|d_{k-1}...|d_1$) and the Bounded Argument Condition (for $b|c_m...|c_1$). Thus the recovery process for both cases are bounded.

Recall the following cases for "GTRC▶GTRC":

(25) *a.* $\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright \underline{\mathsf{U}|(\mathsf{U}|c_p...|c_1)|d_n...|d_{k+1}}|d_k...|d_1$

$\longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2|d_k...|d_1$

b. $\mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2/\underline{b_1} \blacktriangleright \underline{\mathsf{U}}|(\mathsf{U}|c_p...|c_1)|d_{k-1}...|d_1$

$\longrightarrow \mathsf{T}|\,(\mathsf{T}|a_m...|a_1)\,|b_n...|b_2|(b_1|c_p...|c_1)|d_{k-1}...|d_1 \quad (k \geq 1)$

```
Initialization:
  • Create an $n^2$ GTRC recovery table, $R$
Table setup (Stage 1):
  • For each cell (top-down)                                    $O\left(n^2\right)$
      • For each entry (depending on the midpoint)              $O(n)$
          • Restore the derivation info from CKY table
            and store the children in the appropriate cells     $O\left(n^2\right)$
Recovery (Stage 2):
  • For each cell in the bottom row (right-to-left)                       $O(n)$
      • For each entry                                                    $O(n)$
          • If there is a matching category in the target category set
              • Mark the current entry as ‘success’
            Otherwise
              • Mark the current entry as ‘fail’
          • Do status percolation
Status percolation (subprocedure):
  • For each cell (bottom-up)                                                        $O\left(n^2\right)$
      • For each entry                                                               $O(n)$
          • For each parent                                                          $O\left(n^2\right)$
              • If the parent is marked as ‘fail’
                  • Mark the current entry as ‘fail’
                Otherwise
                  • If the current entry is the right branch and marked as ‘fail’
                    and all the right branch siblings are marked as ‘fail,
                      • Mark the parent as ‘fail’
                  • If the current entry is the left branch and marked as ‘success’
                      • Mark the parent as ‘success’
```

Figure 5: **GTRC Recovery Algorithm**

$$d\text{i.}\quad \mathsf{T}/\underline{(\mathsf{T}|a_m\ldots|a_2\backslash a_1)} \blacktriangleright \underline{\underbrace{\mathsf{U}|(\mathsf{U}|c_p\ldots|c_1)|d_n\ldots|d_{k+m+1}}_{\mathsf{T}}|\underbrace{d_{k+m}\ldots|d_{k+1}}_{|a_m\ldots\backslash a_1}}|d_k\ldots|d_1$$

$$\longrightarrow \mathsf{U}|(\mathsf{U}|c_p\ldots|c_1)|d_n\ldots|d_{k+m+1}|d_k\ldots|d_1$$

$$d\text{ii.}\quad \mathsf{T}/\underline{(\mathsf{T}|a_m\ldots|\underset{\substack{\shortparallel\\ a_{m-j,0}|a_{m-j,p}\ldots|a_{m-j,1}}}{a_{m-j}}\ldots|a_2\backslash a_1)} \blacktriangleright \underline{\underbrace{\mathsf{U}_0}_{\mathsf{T}}\underbrace{|\mathsf{U}_j\ldots|\mathsf{U}_1}_{|a_m\ldots}\underbrace{|(\mathsf{U}_0|\mathsf{U}_j\ldots|\mathsf{U}_1|c_p\ldots|c_1)}_{a_{m-j}=F|a_{(m-j,q)}\ldots|a_{(m-j,1)}}|\underbrace{d_n\ldots|d_{k+1}}_{\ldots\backslash a_1}}|d_k\ldots|d_1$$

$$\longrightarrow F|a_{(m-j,q)}\ldots|a_{(m-j,q-j-p)}|d_k\ldots|d_1 \qquad \text{where } q \geq j+p$$

The cases (a) and (b) are analogous to (24a) and (24b). For the case $(d\text{i})$, we start comparing the outer sequence of the input category and the inner sequence of the functor category. The outer sequence of the input category can be treated as if it were the arguments of a constant category. Since T spans greater than U, the entire inner sequence of the functor category must be exhausted by comparing with the outer sequence of the input category. The result category can be obtained from the remaining part of the inner sequence of the input category with the remaining sequence “$|d_k\ldots|d_1$”.

The case $(d\text{ii})$ is slightly different from the previous one in that the inner sequence of the functor category is excessive. We need a process of comparing the inner sequences of the functor and the input categories. Two GTRC recovery processes must be run in parallel.

Through the examination, we conclude that the polynomial parsability of CCG-GTRC depends on recovery of GTRCs. We present the polynomial GTRC recovery algorithm in Figure 5. An example of GTRC recovery is given in Appendix B.

The GTRC recovery algorithm takes advantage of the encoded shared structure, and utilizes an additional n^2 GTRC recovery table to restore possibly ambiguous GTRC derivations in polynomial time. The first stage (*table setup*) is to represent the derivational structure available in the CKY table in a slightly different way. Suppose the following partial CKY table starting from a GTRC in question:

(26) Partial CKY table:

5	$\mathsf{T}/(\mathsf{T}\ldots\backslash A\ldots\backslash B)_{[1,2]\blacktriangleright[3,5]}$				
4					
3			$\mathsf{T}/(\mathsf{T}/C\ldots\backslash B)_{[3,3]\blacktriangleleft[4,5]}$		
2	$\mathsf{T}/(\mathsf{T}\backslash A)_{[1,1]\blacktriangleright[2,2]}$			$\mathsf{T}\backslash(\mathsf{T}/C)_{[4,4]\blacktriangleleft[5,5]}$	
1	$\mathsf{T}/(\mathsf{T}\backslash A)/D$	D	$\mathsf{T}/(\mathsf{T}\backslash B)$	E	$\mathsf{T}\backslash(\mathsf{T}/C)\backslash E$
	1	2	3	4	5

$\mathsf{T}/(\mathsf{T}\ldots\backslash A\ldots\backslash B)_{[1,2]\blacktriangleright[3,5]}$ represents the derivation "$\mathsf{T}/(\mathsf{T}\backslash A)_{[1,1]\blacktriangleright[2,2]} \blacktriangleright \mathsf{T}/(\mathsf{T}/C\ldots\backslash B)_{[3,3]\blacktriangleleft[4,5]} \rightarrow \mathsf{T}/(\mathsf{T}\ldots\backslash A\ldots\backslash B)$" at the designated string positions. Only the last argument of the link is stored in the current cell to avoid exponential number of entries. A GTRC recovery table can be used to store the same derivational structure with the bottom row corresponding to *the order of the inner sequence* of the GTRC rather than the string position. This is the order to process the inner sequence for C_i-A_i comparison shown in (22). Since GTRC recovery process only concerns with the inner sequence of the GTRCs, the recovery table may have a dimension smaller than the corresponding portion of the CKY table as seen in the following example:

(27) GTRC recovery table:

3	$\mathsf{T}/(\mathsf{T}\backslash A\ldots\backslash B)_{[1,1]\blacktriangleright[2,3]}$		
2		$\mathsf{T}/(\mathsf{T}/C\backslash B)_{[2,2]\blacktriangleright[3,3]}$	
1	$\mathsf{T}/(\mathsf{T}\backslash A)$	$\mathsf{T}\backslash(\mathsf{T}/C)$	$\mathsf{T}/(\mathsf{T}\backslash B)$
	3	2	1

The categorial ambiguities originally aligned at string positions are now aligned in the order of processing.

In the second stage (*recovery stage*), the comparison with the target categories is done while the above-mentioned dependency among LTRCs in the bottom row is checked. The comparison proceeds from right to left in the bottom row. The decision on the cancellation of the argument under consideration, a_i, depends on (i) if it is unifiable with some target category (in C_i) *and* (ii) if the corresponding sequence to the right of a_i was successfully canceled. This latter condition can be checked by observing the status of the first right branch from the current position since all the processes up to that point must have been completed. For the later processing (for the positions to the left), the success/failure status of the current category must also be percolated to the relevant higher nodes (*status percolation*). Note that even though the algorithm needs to check *all* the right branch siblings, the number of the siblings is bounded by the number of categories and directionalities. The total complexity turns out to be a rather daunting $O\left(n^{10}\right) = O(\underset{i,j,k}{n^3} \times \underset{\text{recovery}}{n^2} \times \underset{\text{percolation}}{n^5})$.

The extremely-high cost of GTRC recovery process seems to be related to the following conjecture: the generative power of this subclass of CCG-GTRC is greater than CCG-Std. A more restricted version of CCG-GTRC has been shown to be weakly equivalent to CCG-Std [Komagata, 1997b, Komagata, 1997c]. This subclass restricts the directionality of GTRC to the unidirectional case and only deals with GTRCs without outer sequence, thus allowing only the GTRCs of the form $\mathsf{T}\langle(\mathsf{T}\rangle a_m\ldots\rangle a_1)$. In this case, the inner sequence and the string positions agree on the number and the order of the arguments and no GTRC recovery algorithm is needed. Realistically, most languages seem to restrict the use of GTRCs in some sense. For example, the Japanese grammar used in our experiment restricts GTRCs to the unidirectional variety but still with outer sequence.

In summary, the practical and the theoretical polynomial results are due to distinct factors. The former comes from a practical bound on the number of cell entries and spurious ambiguity elimination. The latter (for both Poly-Std and Poly-GTRC) is achieved by efficiently representing and processing the potentially exponentially-many entries in a cell. This is possible even with the presence of spurious/genuine ambiguities. But what the polynomial algorithms do is to eliminate a possibility which in practice rarely occurs. The additional cost for Poly-Std/GTRC of managing n^2 subcells and links to cover all the cases of exponential factors including spurious/genuine ambiguities is thus considered as overkill for the practical case. Although it may be possible to add spurious ambiguity check to Poly-Std/GTRC, we are better off with a simple CKY-style parser with equivalence check, without the overhead of the

Poly-Std/GTRC.

5 Conclusion

We have shown that the extension of CCGs including GTRCs can be parsed polynomially in practice and in theory, with some qualifications and conditions. These polynomial results support the proposed grammar which can describe non-traditinoal constituency widely observed across languages, without resorting to a special mechanism for each case. We expect that the grammar is also useful for practical applications.

For practical applications of the parser, though, we have an agendum for future research. A larger-scale experiment is necessary to obtain statistical significance for varying domains. We may want to consider potentially faster algorithms. For example, GLR-style algorithm may be extended to the proposed case. The most critical problem remains to be that of genuine ambiguity. We may explore a more compact representation of the derived semantics, e.g. polynomial shared structure algorithm of Dörre [1997].[29] Alternatively, we may try to disambiguate early during the recognition stage by a probabilistic method or contextual information (e.g., use of information structure). We expect that these techniques will be applicable to the presented parser and will improve the performance to a really practical level.

A Experiment

A.1 List of Parsed Sentences

A few commas ‘,’ are inserted manually as $\boxed{,}$ to maintain grammaticality within the current framework.

1. kouketuatu-wa, kourei-ni naru-to ookuno hito-ni araware, nousottyuu+sinsikkan-nado-no ookina byouki-no gen’in-ni narimasu.

2. sunawati, kouatu-no saidaino mokuhyou-wa nousottyuu-to kyoketusei sinsikkan-no yobou-desu.

3. genzai, nousottyuu-no yobou_kouka-wa mitomeraretekiteˆimasu-ga, kyoketusei sinsikkan-niˆkansiteˆwa mada huzyuubunna zyoutai-desu.

4. kore-wa kono sinsikkan-o warukusuru youso-tosite, ketuatu-igaini doumyakukouka-no seiin-to_naru kousiketusyou, toutaisya no izyou, himan, kituen-nado-mo kankeisiteiru-koto-ni_mo-yorimasu.

5. ketuatu ni_wa sinzou-ga syuusyukusite, ketueki-o okuridasu-toki-no syuusyukuki_ketuatu-to sinzou-ga kakutyousi, sinzou_nai-ni ketuatu-ga modottekuru-toki-no kakutyouki_ketuatu-ga ari, sinzou-ga syuusyukusita-toki_ni ketuatu-wa mottomo takakunari, kakutyousita-toki_ni mottomo hikukunarimasu.

6. kouketuatu-no teido-wa kakutyouki_ketuatu-de handansare, 90_105mmHg-nara keido kouketuatu_syou, 105_115mmHg-de tyuutoudo kouketuatu_syou, 115mmHg-izyou_deˆwa zyuusyou kouketuatu_syou-to_narimasu.

7. itumo 95mmHg-o koeru-baai_wa tiryou-ga hituyoudesu.

8. kouatu-no mokuhyou-tosite_wa, seizyou ketuatu-no 140/90mmHg-ga ippantekidesu-ga, zin_kinnou-nado-no zyouken-ni_yori takameni kontoroorusareru-koto-mo arimasu.

9. ketuatu-ga takai-toki_ni_wa, kekkan-no naka-o tooru ketueki_ryou-ga oosugiru-baai-to, ketueki_ryou-wa seizyoudemo kekkan-no kabe-ga semakunatteiru-baai-to_ga kangaerare, korera-no zyoutai-o kaizensuru-no-ga kouketuatu-no tiryou-desu.

[29] Applicability of this technique to our parser needs to be carefully examined because semantic equivalence check will be required to traverse the shared structures. It is not clear if the traversal can be done in polynomial time. This concern is shared by the situation of applying structure sharing technique to conceptual dependency [Bröker et al., 1994]. Other reports on shared structure on semantic representation include [Nagao, 1994] and [Schiehlen, 1996].

10. desukara yoku sirareteiru-koto-desu-ga, enbun_seigen-ya himan-no kaizen-nado-ga tiryou_zyou totemo taisetudesu.

11. mata kituen-ya insyu-mo sakenakerebanarimasen.

12. korera-ni tyuuisite-mo ketuatu-no kanri-ga muzukasii-baai_ni hazimete kusuri-o tukaimasu.

13. siyousuru kusuri-wa sayou-ga odayakade$\boxed{,}$tyoukikan hukuyousite-mo hukusayou-ga sukunaku, hukuyou_kaisuu-ga sukunakutesumu-mono-ga risou-desu.

14. kouatu_syou-no seiin-tosite tugino mittu-no youso-ga kankeisiteimasu.

15. natoriumu_sessyu, koukan_sinkei_kei, renin_angiotensin_kei-no mittu-desu.

16. rinyouzai-wa natoriumu-ni, beeta_arufa_syadanzai-wa koukan_sinkei_kei-ni, sosite ace_sogaizai-wa renin_angiotensin_kei-ni sorezore kankei_no_tuyoi taipu-no kouketuatu-no hito-ni yoku kiku-koto-ga kitaisaremasu.

17. mazu saisyo_ni motiirareru-no-wa rinyouzai, beeta_syadanzai, karusiumu_kikkouzai, ace_sogaizai-desu.

18. kouatuzai-no naka-ni_wa beeta_syadanzai-ya rinyouzai-ga sisitu-no taisya-ni warui eikyou-o oyobosu kanousei-ga aru-to iwareteimasu.

19. mata saikin-de_wa sisitu-ya tousitu-o hukumeta mondai-tosite, insurin_kanzyusei-ni korera kouatuzai-ga henka-o oyobosu-koto-ga tyuumokusareteimasu.

20. insurin_kanzyusei-o yokusuru kouatuzai-to warukusuru yakuzai-to-ga ari, warukusuru daihyou-ga beeta_syadanzai-to rinyouzai-to iwareteimasu.

21. korera-no ten-mo kangaete$\boxed{,}$syuzyuno kouatuzai-ga sono hito-goto-ni motiirareteimasu.

22. rinyouzai-to karusiumu_kikkouzai-to-wa maeni nobemasita-node, kokode_wa, karusiumu_kikkouzai-no itibu-to beeta_syadanzai$\boxed{,}$ ace_sogaizai-o toriagemasu.

A.2 Features of the Parser

- Environment: Sun Ultra E4000 2x167MHz Ultraspacs with 320MB memory running SunOS 5.5.1

- Program: The program of about 100KB is written in Sicstus Prolog Ver. 3. About a half of the program is the grammar. CPU time was measured by Sicstus built-in predicate `statistics`.

- Data: The data set consists of 22 contiguous sentences (6 paragraphs) from p. 31-32 of “Anata-no Byouin-no Kusuri (Your Hospital Drugs) 1996” by Tadami Kumazawa and Ko-ichi Ushiro. There are simple and complex clauses (relative and complement), coordination (2-4 conjuncts), nominal/verbal modifications (adjectives/adverbs), scrambling, and verb argument dropping.

- Input format: The input is romanized and partially segmented sentences in Japanese. Verb inflection and suffixes are considered as a part of the word. As a consequence, certain verbs are fairly long and this contributes the long average time for morphological analysis.

- Language-specific constraints: GTRCs are restricted to the unidirectional form by restricting the directionality of rule application Case (12*e*), “GTRC▶GTRC”, to the form “$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \blacktriangleright^k \underline{\mathsf{U}} \underset{\uparrow}{/} (\mathsf{U}|c_p...\backslash c_1)|d_{k-1}...|d_1$”.

  The bound on functional composition k_{max} is 2.

- Parsing algorithm: CKY-style with semantic spurious ambiguity check (mutual subsumption check) but without the presented worst-case polynomial recognition algorithm

- Morphological analysis: Every possible complete substring match is sought. Successful results are dynamically asserted in the lexicon. The complete match is probably the source of slowness in morphological analysis.

- Coordination: A set of special trinomial rules for specific categories are implemented [Steedman, 1996]. By implementing coordination as rule has advantage of avoiding compositions such as “$S/NP \blacktriangleleft S/S\backslash S$”. This is equivalent to restricting composition with conjunctives. Although bounded coordination is covered by CCGs, this restriction changes the behavior of the formal system. Only the left-branching conjunction is considered (a kind of normal form condition). The special punctuation in Japanese, ‘+’, is implemented as coordination only between two uncombined Ns. Some ‘,’ are inserted (this may be done automatically). The semantics of coordination is preliminary and requires more research [cf. Park, 1992, Bayer and Johnson, 1995].

- Verb-argument dropping: First, we call verb arguments NP after noun-particle combination of “N+NP\N” but not PP after “NP+PP\NP” [cf. Gunji, 1987]. ‘Dropping’ is implemented as lexical rule changing verb valency and make the semantic argument a free variable. This treatment supports lexical analysis of binding as follows. Since reflexive pronouns cannot be dropped, reflexivization must precede dropping. Then it must be lexical.

- Passive: This is treated as lexical rule changing verb subcategorization and semantics. A few additional cases of pseudo-passive are also implemented.

- Scrambling: Both local and long-distance are currently handled by crossing composition $x/y \blacktriangleright y\backslash z \rightarrow x\backslash z$. Currently, derivation such as “$\{B\ A\}$ & $\{B\ A\}$ $S\backslash A\backslash B$” is not accepted. Since this type of coordination can appear within an embedded clause where no theme-rheme distinction is presumed to be made, we may need to allow such a case. The current implementation has capability to handle this by lexical rule to derive $S\backslash B\backslash A$ from $S\backslash A\backslash B$. Alternatively, ‘bag’ notation of Hoffman [1995] may be used. For Japanese, the scope of a bag seems to be limited to local, unlike Turkish.

- Examples of lexical entry/lexical rules:

```
"ketuatu" := [lang=j2,class=n(com-1),int=blood_pressure,
        features=[ana:(-),agr:(3,_N,n,_C)]].
"ga" := [lang=j2,class=case(nom),int=nom,gloss='-NOM',
        features=[top:(-)],is=_].
"mitomeru" := [lang=j2,class=v(2-np(acc)),int=accept,features=[],
        infl=((ru,itidan,reg),(ta,ta,reg))].
lex_stages(j2,parsing,[init,gloss,
        category,
        (infl_no|[infl_yes,
        rare,must,{scrambling},{dropping},polite,{attributive},{adverbial}]),
        sent_final,{type_raising}]).
lex_rule(j2,category,[if(class=n(com-1)),
        cat=n(q),
        int=(Int=>q(?,X,Int-X)),
        is=_]).
lex_rule(j2,category,[if(class=case(Case)),
        cat=np(Case)\n(_),
        features=(F=>[comb:(+)|F]\F),
        int=I^I]).
lex_rule(j2,category,[if(class=v(2-np(dat))),
        cat=s\np(nom)\np(dat),
        features=(F=>F\[]\[]),
        int=(Int=>A^B^(Int-A-B)),
        is=_]).
lex_rule(j2,dropping,[if(class=v(2-np(_))),
        class=v(1-v),
        ifnot(drop_nom=no),
        cat=(s\np(_C2)\np(C3)=>s\np(C3)),
        features=(F1\_F2\F3=>F1\F3),
        int=(A^B^(V-A-B)=>A^(V-A-B))]).
lex_rule(j2,type_raising,[if(cat=np(C)\n(ind)),
        cat=((np(C))^^)\n(ind),
        int=X^(X^P)^P]).
```

- User programmability: The grammar file is independent of the system program. A linguist may specify lexical assignment, segmentation setting, rule specification, inflection entry, and various lexical rules. Run-time switch can also be changed. Thus it is straightforward to create grammars for other languages.
- Output: The output is a (possibly empty) set of logical form in predicate-argument structure (no scope ambiguity, no pronoun resolution, no lexical/syntactic ambiguity resolution).
- Other: IS capability (e.g., wa/ga distinction) has been experimented but not activated for the current experiment.

A.3 Test Results

Legend:

- Sent: sentence ID
- #words: number of words in a sentence
- 1st: first run after resetting the asserted lexicon
- 2nd: second run of the same sentence (no morphological analysis)
- #parse: number of distinct parses
- #entry$_{max}$: maximum number of entries in a cell

(28) Basic Data Set:

Sent	#words	Category+Semantics				Category only		
		1st [s]	2nd [s]	#parse	#entry$_{max}$	2nd [s]	#parse	#entry$_{max}$
1	23	15.60	12.96	42	69	4.82	2	11
2	14	3.50	1.36	6	9	0.80	1	7
3	16	9.80	1.36	4	5	1.22	2	5
4	30	46.54	40.20	96	144	7.04	3	7
5	41	–	–	–	–	27.08	2	8
6	23	16.01	7.42	24	48	3.31	4	8
7	8	4.86	0.36	2	8	0.36	1	8
8	23	12.11	6.48	16	20	4.05	2	8
9	40	–	–	–	–	32.84	3	29
10	17	7.05	2.76	18	18	2.19	1	8
11	6	1.32	0.17	1	7	0.19	1	7
12	14	4.05	0.97	3	7	1.14	1	7
13	21	10.26	4.68	28	28	3.19	2	8
14	10	2.45	0.42	2	5	0.63	1	5
15	8	1.30	0.34	1	5	0.50	1	9
16	28	35.57	31.77	16	82	7.36	2	10
17	13	7.48	3.20	24	40	1.86	2	11
18	21	9.69	5.63	2	15	4.03	1	8
19	22	9.93	6.22	23	23	4.59	4	8
20	19	15.60	10.20	3	17	7.60	1	7
21	14	7.06	1.67	1	7	1.84	1	7
22	20	11.76	9.00	8	16	5.00	1	7
Total	431	231.90	147.20	320	573	121.64		
Avg	19.6	11.60	7.36	16.0	28.7	5.53		

In the following, sentences are fabricated by conjoining two sentences from the test data. '$i \times 2$' means Sentence i is repeated twice and '$i + j$' means Sentence i and Sentence j are conjoined. The conjunction is obtained by the use of either a conjunctive or the adverbial verb ending (on the first sentence) followed by comma.

(29) Pseudo-Long Sentences for Extended Data Set:

Sent	#words	Category only		
		2nd [s]	#parse	#entry$_{max}$
1×2	47	31.22	2	11
6×2	47	37.16	5	12
8×2	47	29.88	4	14
16×2	57	40.70	1	9
4×2	61	69.47	6	12
9+4	71	122.90	6	
5+4	72	111.20	4	

- Average lexical lookup: 4.24 sec/sentence, 0.2 sec/word
- Without spurious ambiguity check, only s11, s14, and s15 terminated (s7 seems to have problem with the numeral). Other sentences ended up with 300-700 maximum entries in a cell after 3 mins.
- Number of asserted lexical entry (forms): 200 approx.

B An Example of Poly-GTRC Process

In this appendix, we explore a process of the GTRC recover algorithm with example. We introduce the following representation for constant category and GTRC:

(30) *a*. Const: $\mathrm{const}(\mathrm{target}, -, \{\mathrm{link}, \mathrm{index}\}, \mathrm{tail})$

b. GTRC: $\mathrm{gtrc}(\mathrm{target}, (\{\mathrm{link}_{inner1}, \mathrm{index}_{inner1}\}, \{\mathrm{link}_{inner2}, \mathrm{index}_{inner2}\}), \{\mathrm{link}_{outer}, \mathrm{index}_{outer}\}, \mathrm{tail})$

The representation for Const is basically the same as Poly-Std except for the arrangement of the elements. *Link* refers to the cell location where further information is stored. *Index* contains a sequence of arguments (bounded) to identify the appropriate linked representation which would match the tail of the linked cell. For the case of GTRC, index$_{inner1/2}$ only contains a single argument and two links are needed since neither of the component GTRCs is in general bounded. A few examples are shown below.

(31) *a*. Const: $\mathrm{const}(S, -, -, \backslash A) = S\backslash A$, $\quad (S, -, \{[1,3], \backslash B\}, \backslash A) = S \underset{\text{link to [1,3]}}{...\backslash B} \backslash A$

b. GTRC: $\mathrm{gtrc}(T, (-, \{[4,4], \backslash A\}), -, -) = \overset{T}{\mathsf{T}} / (\mathsf{T}\backslash A)$,

$\mathrm{gtrc}(T, (\{[5,5], \backslash B\}, \{[4,4], /A\}), \{[6,8], /C\}, /D) = \overset{T}{\mathsf{T}} \backslash (\mathsf{T} \underset{[5,5]}{\backslash B} \underset{[4,4]}{/A}) \underset{[6,8]}{.../C} /D$

As noted earlier, the component ordering in this representation does not necessarily correspond to the string positions. For example, “$\mathsf{T}\backslash(\mathsf{T}/A) \blacktriangleleft \mathsf{T}\backslash(\mathsf{T}/B) \rightarrow \mathsf{T}\backslash(\mathsf{T}/B/A)$” is represented as “$(\mathsf{T}, (-, \{[i,i], /A\}), -, -) \blacktriangleleft (\mathsf{T}, (-, \{[i+1, i+1], /B\}), -, -) \longrightarrow (\mathsf{T}, (\{[i+1, i+1], /B\}, \{[i,i], /A\}), -, -)$” with appropriate *i*. In any case, the functor always comes to the left of the input.

The much harder part is to restore GTRCs from a representation like $(\mathsf{T}, (\{[1,3], \backslash A\}, \{[4,5], \backslash B\}), -, -)$. What are the corresponding set of GTRCs? First, observe the following derivation of GTRCs (in CKY-style table) as background:

(32) GTRC derivation example:

3	$a.\ \mathsf{T}/(\mathsf{T}\backslash A\backslash B\backslash C)_{[1,2]\blacktriangleright[3,3]\text{ or }[1,1]\blacktriangleright[2,3]}$ $b.\ \mathsf{T}\backslash(\mathsf{T}\backslash A\backslash B/C)_{[1,2]\blacktriangleright[3,3]\text{ or }[1,1]\blacktriangleright[2,3]}$ $c.\ \mathsf{T}/(\mathsf{T}/C\backslash A\backslash B)_{[1,2]\blacktriangleleft[3,3]}$ $d.\ \mathsf{T}/(\mathsf{T}\backslash B/C\backslash A)_{[1,1]\blacktriangleleft[2,3]}$ $e.\ \mathsf{T}/(\mathsf{T}\backslash A/C\backslash B)_{[1,1]\blacktriangleright[2,3]}$		
2	$\mathsf{T}/(\mathsf{T}\backslash A\backslash B)_{[1,1]\blacktriangleright[2,2]}$	$\mathsf{T}/(\mathsf{T}\backslash B\backslash C)_{[2,2]\blacktriangleright[3,3]}$ $\mathsf{T}\backslash(\mathsf{T}\backslash B/C)_{[2,2]\blacktriangleright[3,3]}$ $\mathsf{T}/(\mathsf{T}/C\backslash B)_{[2,2]\blacktriangleleft[3,3]}$	
1	$\mathsf{T}/(\mathsf{T}\backslash A)$	$\mathsf{T}/(\mathsf{T}\backslash B)$	$\mathsf{T}/(\mathsf{T}\backslash C)$ $\mathsf{T}\backslash(\mathsf{T}/C)$
	1	2	3

Suppose we recover the LTRCs to derive the GTRCs in the above example. For simplicity, let us focus on the entries *a*, *b*, and *e*. We start from the top cell with two entries corresponding to the two distinct derivations. Note that the entries are copied from the CKY table and that the positional information such as $[1,2]$ represents 'the position in the CKY table'.

(33) GTRC Recovery Table (setup stage):

3	$(\{[1,2],\backslash B\},\{[3,3],\backslash C\})$ $(\{[1,1],\backslash A\},\{[2,3],\backslash C\})$ $\vdots$ $(\{[1,1],\backslash A\},\{[2,3],\backslash B\})$		
2	$(\{[1,1],\backslash A\},\{[2,2],\backslash B\})$	$(\{[2,2],\backslash B\},\{[3,3],\backslash C\})$ $(\{[3,3],/C\},\{[2,2],\backslash B\})$	
1	$(-,\{[1,1],\backslash A\})$	$(-,\{[2,2],\backslash B\})$ $(-,\{[3,3],/C\})$	$(-,\{[3,3],\backslash C\})$ $(-,\{[2,2],\backslash B\})$
	3	2	1

The setup procedure is as follows: for each cell from top-down, find the child GTRC representations from CKY table and store them in the appropriate cell. For example, $(\{[1,1],\backslash A\},\{[2,3],\backslash C\})$ has the right child in [2,3] of CKY table. This is $\mathsf{T}/(\mathsf{T}\backslash B\backslash C)$ and the corresponding representation is stored in the $[2,3]$ cell of the recovery table. $\mathsf{T}/(\mathsf{T}/C\backslash B)$ is also stored in the same cell since it matches $(\{[1,1],\backslash A\},\{[2,3],\backslash B\})$. In addition, allocate a small number of status bits for each entry.

The recovery stage processes the bottom row from right to left by checking the categorial consistency with the target and maintaining status bits.

(34) GTRC Recovery Table (recovery stage):

3	$status{:}?\!\rightarrow\! succ\,\boxed{5_c}$ $(\{[1,2],\backslash B\},\{[3,3],\backslash C\})$ $status{:}?\!\rightarrow\! succ\,\boxed{5_b}$ $(\{[1,1],\backslash A\},\{[2,3],\backslash C\})$ $\vdots$ $status{:}?\!\rightarrow\! fail\,\boxed{2_b}$ $(\{[1,1],\backslash A\},\{[2,3],\backslash B\})$		
2	$status{:}?\!\rightarrow\! succ\,\boxed{5_a}$ $(\{[1,1],\backslash A\},\{[2,2],\backslash B\})$	$status{:}?\!\rightarrow\! succ\,\boxed{3_a}$ $(\{[2,2],\backslash B\},\{[3,3],\backslash C\})$ $status{:}?\!\rightarrow\! fail\,\boxed{2_a}$ $(\{[3,3],/C\},\{[2,2],\backslash B\})$	
1	$status{:}?\!\rightarrow\! succ\,\boxed{5}$ $(-,\{[1,1],\backslash A\})$	$status{:}?\!\rightarrow\! succ\,\boxed{3}$ $(-,\{[2,2],\backslash B\})$ $status{:}?\!\rightarrow\! fail\,\boxed{4}$ $(-,\{[3,3],/C\})$	$status{:}?\!\rightarrow\! succ\,\boxed{1}$ $(-,\{[3,3],\backslash C\})$ $status{:}?\!\rightarrow\! fail\,\boxed{2}$ $(-,\{[2,2],\backslash B\})$
	3 $\updownarrow$ $\{\backslash A\}$	2 $\updownarrow$ $\{\backslash B,\backslash C\}$	1 $\updownarrow$ $\{\backslash C\}$

Note that there is a dependency between the categories in the bottom row positions 1 and 2 of the recovery table. Although $\mathsf{T}/(\mathsf{T}\backslash A\backslash B\backslash C)$ and $\mathsf{T}/(\mathsf{T}\backslash A/C\backslash B)$ are valid entries in (32), $\mathsf{T}/(\mathsf{T}\backslash A\backslash B\backslash B)$ and $\mathsf{T}/(\mathsf{T}\backslash A/C\backslash C)$ are not. In general, this dependency can span the order of *n*. Checking the entire history of the previous positions will cost exponential time. But we only need the information if the right branch of the current position has failed or not (depending on tree structure with directionality and success/failure for each leaf). Thus by marking the success status at the root of each substructure, we can check the status of the right branch in polynomial time.

Let us start from the bottom row position 1. Compare $\backslash C$ with the target set (suppose that the target is $S\backslash A \left\{\begin{matrix}\backslash B\\ \backslash C\end{matrix}\right\} \backslash C$). Since the categories are compatible, we set the status '*succ*' as indicated by $\boxed{1}$. On the other hand, $\backslash B$ fails, getting '*fail*' indicated by $\boxed{2}$. For an entry which is the right branch of a parent, if all the right branch sibling of the current node failed, we percolate up the failure status to the parent. This will block any sequence involving $\backslash B$ at this position. The failed parent is labelled with $\boxed{2_a}$ and $\boxed{2_b}$. Move to the second position. $\backslash B$ succeeds in comparison. For the left branch, we percolate up the success status to the parents. This is shown by $\boxed{3a}$. $/C$ can be failed without comparing with the target since the parent already indicates the failure. The recovery process succeeds when one of the top entries gets '*succ*' as in $\boxed{5_b}$ and $\boxed{5_c}$.

References

Anthony Ades and Mark J. Steedman. 1982. On the Order of Words. *Linguistics and Philosophy*, 4.

Alfred V. Aho and J. D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling. Vol. 1: Parsing*. Prentice-Hall.

Sam Bayer and Mark Johnson. 1995. Features and Agreement. In *ACL33*.

Tilman Becker, Aravind Joshi, and Owen Rambow. 1991. Long-Distance Scrambling and Tree Adjoining Grammars. In *EACL5*.

Sylvie Billot and Bernard Lang. 1989. The Structure of Shared Forests in Ambiguous Parsing. In *ACL27*.

Norbert Bröker, Udo Hahn, and Susanne Schacht. 1994. Concurrent Lexicalized Dependency Parsing: The ParseTalk Model. In *COLING-94*.

Bob Carpenter. 1991. The Generative Power of Categorial Grammars and Head-Driven Phrase Structure Grammars with Lexical Rules. *Computational Linguistics*, 17.

Robert L. Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press.

Jochen Dörre. 1997. Efficient Construction of Underspecified Semantics under Massive Ambiguity. In *ACL35/EACL8*.

David Dowty. 1988. Type Raising, Functional Composition, and Non-Constituent Conjunction. In Richard Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Reidel.

Marc Dymetman. 1997. Charts, Interaction-Free Grammars, and the Compact Representation of Ambiguity. In *IJCAI-97*.

Jason Eisner. 1996. Efficient Normal-Form Parsing for Combinatory Categorial Grammar. In *ACL 34*.

Martin Emms. 1993. Parsing with polymorphism. In *EACL6*.

Joyce Friedman and Ramarathnam Venkatesan. 1986. Categorial and Non-Categorial Languages. In *ACL24*.

Takao Gunji. 1987. *Japanese Phrase Structure Grammar: A Unification-Based Approach*. D. Reidel.

Herman Hendriks. 1993. *Studied Flexibility: Categories and Types in Syntax and Semantics*. ILLC Dissertation Series.

Mark Hepple and Glyn Morrill. 1989. Parsing and Derivational Equivalence. In *EACL 4*.

Mark Hepple. 1987. Methods for Parsing Combinatory Grammars and the Spurious Ambiguity Problem.

Mark Hepple. 1990. *The Grammar and Processing of Order and Dependency: a Categorial Approach*. PhD thesis, University of Edinburgh.

Toru Hisamitsu and Yoshihiko Nitta. 1994. An Efficient Treatment of Japanese Verb Inflection for Morphological Analysis. In *COLING-94*.

Beryl Hoffman. 1993. The Formal Consequences of Using Variables in CCG Categories. In *ACL31*.

Beryl Hoffman. 1995. *The Computational Analysis of the Syntax and Interpretation of "Free" Word Order in Turkish*. PhD thesis, University of Pennsylvania.

Aravind K. Joshi, Tilman Becker, and Owen Rambow. 1994. Complexity of Scrambling: A New Twist to the Competence - Performance Distinction. In *3e Colloque International sur les grammaires d'Arbres Adjoints*.

Megumi Kameyama. 1995. The Syntax and Semantics of the Japanese Language Engine. In Reiko Mazuka and Noriko Nagai, editors, *Japanese Sentence Processing*. Lawrence Erlbaum.

Lauri Karttunen. 1986. Radical Lexicalism. Technical report, CSLI.

Nobo Komagata. 1997a. Efficient Parsing for CCGs with Generalized Type-Raised Categories. In *IWPT97*.

Nobo Komagata. 1997b. Generative Power of CCGs with Generalized Type-Raised Categories. In *ACL35/EACL8 (Student Session)*.

Nobo Komagata. 1997c. Generative Power of CCGs with Generalized Type-Raised Categories. Technical report (IRCS-97-15), University of Pennsylvania.

Esther König. 1994. A Hypothetical Reasoning Algorithm for Linguistic Analysis. *J. Logic and Computation*, 4(1):1–19.

George Miller and Noam Chomsky. 1963. Finitary Models of Language Users. In R.R. Luce et al., editors, *Handbook of Mathematical Psychology, Vol. II*. John Wiley and Sons.

Michael Moortgat. 1988. *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*. Foris.

Glyn V. Morrill. 1994. *Type logical grammar: categorial logic of signs*. Kluwer.

Katashi Nagao. 1994. A Preferential Constraint Satisfaction Technique for Natural Language Analysis. *IEICE Trans. Inf. & Syst*, E77-D(2).

Remo Pareschi and Mark Steedman. 1987. A Lazy Way to Chart-Parse with Categorial Grammars. In *ACL25*.

Jong C. Park. 1992. A Unification-Based Semantic Interpretation for Coordinate Constructs. In *ACL30*, pages 209–215.

Jong C. Park. 1995. Quantifier Scope and Constituency. In *ACL33*.

Jong Cheol Park. 1996. *A Lexical Theory of Quantification in Ambiguous Query Interpretations*. PhD thesis, University of Pennsylvania.

Lawrence C. Paulson. 1991. *ML for the Working Programmer*. Cambridge University Press.

Fernando C.N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural-Language Analysis*. CSLI.

Scott Prevost and Mark Steedman. 1993. Generating Contextually Appropriate Intonation. In *EACL6*.

Scott Prevost. 1995. *A Semantics of Contrast and Information Structure for Specifying Intonation in Spoken Language Generation*. PhD thesis, University of Pennsylvania.

Owen Rambow and Aravind K. Joshi. 1994. A Processing Model for Free Word Order Languages. In Jr. C. Clifton, L. Frazier, and K. Rayner, editors, *Perspectives on Sentence Processing*. Lawrence Erlbaum.

Owen Rambow. 1994. *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, University of Pennsylvania.

Michael Schiehlen. 1996. Semantic Construction from Parse Forests. In *COLING-96*.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI.

Mark J. Steedman. 1985. Dependency and Coordination in the Grammar of Dutch and English. *Language*, 61:523–

56.

Mark J. Steedman. 1988. Combinators and Grammars. In Richard Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Reidel.

Mark Steedman. 1991a. Structure and Intonation. *Language*, 67.

Mark Steedman. 1991b. Type-Raising and Directionality in Combinatory Grammar. In *ACL29*.

Mark Steedman. 1996. *Surface Structure and Interpretation*. MIT Press.

Hozumi Tanaka and Masahiro Ueki. 1995. Japanese Parsing System using EDR Dictionary. `http://www.icot.or.jp/AITEC/PUBLICATIONS/Itaku/95/catalogue18-E.html`.

Hozumi Tanaka, Takenobu Tokunaga, and Michio Aizawa. 1993. Integration of Morphological and Syntactic Analysis Based on the LR Parsing Algorithm. In *IWPT93*.

Simon Thompson. 1991. *Type Theory and Functional Programming*. Addison-Wesley.

K. Vijay-Shanker and David J. Weir. 1990. Polynomial Time Parsing of Combinatory Categorial Grammars. In *ACL28*.

K. Vijay-Shanker and David J. Weir. 1991. Polynomial Parsing of Extentions of Context-Free Grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer.

K. Vijay-Shanker and David J. Weir. 1993. Parsing Constrained Grammar Formalisms. *Computational Linguistics*, 19(4).

K. Vijay-Shanker and D. J. Weir. 1994. The Equivalence of Four Extensions of Context-Free Grammars. *Mathematical Systems Theory*, 27:511–546.

David Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania.

Pete J. Whitelock. 1988. A Feature-Based Categorial Morpho-Syntax for Japanese. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*. Kluwer.

Kent Wittenburg and Robert E. Wall. 1991. Parsing with Categorial Grammar in Predictive Normal Form. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer.

Kent Barrows Wittenburg. 1986. *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification-Based Formalism*. PhD thesis, University of Texas, at Austin.

Kent Wittenburg. 1987. Predictive Combinators: A Method for Efficient Processing of Combinatory Categorial Grammars. In *ACL25*.