# Appendix A

# Generative Power and Parsing Efficiency of CCG-GTRC

In Section 4.1.4, we briefly touched on generative power and parsing efficiency for CCG involving Generalized Type-Raised Categories (CCG-GTRC). This Section explores these properties in detail based on two technical reports [Komagata, 1997d; Komagata, 1997b] (with minor revision on the notation). The main points are that a restricted version of CCG-GTRC is equivalent to the standard CCG, and that CCG-GTRC is polynomially parsable theoretically and practically. The results have also been presented as Komagata [1997c] and Komagata [1997a].

This section is organized as follows. Subsection A.1 motivates and introduces the formal framework of CCG-GTRC. Subsection A.2 proves the equivalence of CCG-GTRC and standard CCG under specific conditions. Subsections A.3 and A.4 discuss theoretical and practical results, respectively, on polynomial parsing for CCG-GTRC.

## A.1 CCG with Generalized Type-Raised Categories

**Motivation: Unbounded NP Sequence**

In languages including Japanese, a NP sequence can form a constituent for coordination and extraction as seen in Section 4.1.4. A similar type of constituent can also be formed of NPs extracted from different levels of embedding, as in the following example:

(1) 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."

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:[1]

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

As we have seen in (130) on page 102 (Subsection 4.1.4), NP sequence in Japanese can form a category of the form $S/((S\backslash NP)\backslash NP)$. 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. 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 $T/(T\backslash NP)$.

The formal properties of the standard CCGs not involving variable (CCG-Std) are relatively well-studied (see Section 4.1.5). But the use of variables can destroy these properties. For example, Hoffman [1993] showed that a grammar involving categories of the form $(T\backslash x)/(T\backslash y)$ can generate a language $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n\mathtt{e}^n$, which no mildly context-sensitive grammar can generate. The use of variables in the coordination schema "$x^+$ **conj** $x \Rightarrow x$" is also believed to generate a language $(\mathtt{wc})^n$ beyond LIG's power [Weir, 1988]. At a level higher in the scale, Becker et al. [1991], Rambow and Joshi [1994], and Hoffman [1995] propose formalisms that are more powerful than the standard CCG to account for 'doubly'-unbounded scrambling. 'Doubly'-unbounded scrambling has the following properties: (i) there is no bound on the distance of scrambling and (ii) there is no

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

bound on the number of unbounded dependencies in one sentence. As we know that full context-sensitive capacity is too powerful to be a formal model of natural language syntax [e.g., Savitch, 1987], it is essential to identify the generative power of the formalism that interests us.

**Component: 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**. We 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 call '$|a_i...|a_j$' a **sequence** (of arguments). The length of a sequence is defined as $\left||a_i...|a_1\right| = i$ while the null sequence is defined to have the length 0. Thus, an atomic constant category is considered a category with the target category with the null 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). A LTRC has the form $\overset{T}{\mathsf{T}}\left\langle \left(\mathsf{T}\right\rangle a\right)|b_i...|b_1$ 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 $T$. 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. '$\left\langle$' and '$\left\rangle$' indicate that either set of slashes in the upper or the lower tier can be chosen but a mixture such as '$/$' and '$/$' is prohibited [see Steedman, 1991b for a related discussion]. **GTRC** is defined as having the form of $\mathsf{T}\left\langle(\mathsf{T}|\underbrace{a_m...|a_2}\right\rangle a_1)\quad\underbrace{|b_n...|b_1}$

inner sequence outer sequence

resulting from compositions of LTRCs where $m \geq 1$, $n \geq 0$, and the directional constraint is carried over from the involved LTRCs. 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 $gtrc = \mathsf{T}|\left(\mathsf{T}|a_m...|a_1\right)|b_n...|b_1$, 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

by Carpenter [1991].

These categories can be combined by combinatory rule schemata. Rules of (forward) "generalized functional composition" have the following form:[2]

(3) $x/\underline{y} \qquad \underline{y}|z_k...|z_1 \qquad \Longrightarrow \qquad x|z_k...|z_1 \qquad (>B^k)$

functor category input category result category

The integer '$k$' in this schema is bounded by $k_{max}$ specific to the grammar, as in CCG-Std.[3] Rules of functional application, "$x/y \quad y \Rightarrow x$", can be considered a special case of (3) where the sequence $z_i$'s is null. We say "the combination of '$x/\underline{y}$' and '$\underline{y}|z_k...|z_1$' *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 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.

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

(4)

| | 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 as follows:

(5)
```
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
```

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

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

```
| 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 a greater 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.

**Composition Involving GTRCs**

Inclusion of GTRCs calls for a 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 A.1. In the following, a combination of two constant categories is written as Const+Const. Note that all of the following cases are written for '$>B^k$' and the other direction is analogous.

(6) Const+Const: $a/\underline{b} \quad \underline{c}|d_k...|d_1 \implies a|d_k...|d_1$

(7) GTRC+Const

*a*. Functor GTRC has an outer sequence:

$\mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_2/\underline{b_1} \quad \underline{c}|d_k...|d_1 \implies \mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_2|d_k...|d_1$

Example: $\mathsf{T}\backslash(\mathsf{T}/PP)/\underline{NP} \quad \underline{NP} \implies \mathsf{T}\backslash(\mathsf{T}/PP)$

*b*. Functor GTRC has no outer sequence:

$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)} \quad \overset{\overset{c_0|c_m...|c_1}{\underline{\quad\quad}}}{c} \quad |d_k...|d_1 \implies c_0|d_k...|d_1$

Example: $\mathsf{T}/\underline{(\mathsf{T}\backslash NP\backslash NP)} \quad \underline{S\backslash NP\backslash NP} \implies S$

(8) Const+GTRC

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

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

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

$\rightarrow (S/(S\backslash NP\backslash NP))\backslash(S/(S\backslash NP\backslash NP))$

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

$a/\underline{b} \quad \underline{\mathsf{T}}|(\mathsf{T}|c_m...|c_1)|d_{k-1}...|d_1 \implies a|(b|\overbrace{c_m...|c_1}^{\text{unbounded}})|d_{k-1}...|d_1$

Example: $S/\underline{S} \quad \underline{\mathsf{T}}/(\mathsf{T}\backslash NP\backslash NP) \implies S/(S\backslash NP\backslash NP)$

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

$a/\underline{b} \quad \underline{\mathsf{T}_0}|\mathsf{T}_1|(\mathsf{T}_0|\mathsf{T}_1|c_m...|c_1)|d_{k-2}...|d_1 \implies a|\underset{\text{residual}}{\mathsf{T}_1}|(b|\mathsf{T}_1|\overbrace{c_m...|c_1}^{\text{unbounded}})|d_{k-2}...|d_1$[4]

(9) GTRC+GTRC

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

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

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

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

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

$\implies \mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_2|(b_1|\overbrace{c_p...|c_1}^{\text{unbounded}})|d_{k-1}...|d_1$

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

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

$\implies \mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_2|\underset{\text{residual}}{\mathsf{U}_1}|(b_1|\mathsf{U}_1|\overbrace{c_p...|c_1}^{\text{unbounded}})|d_{k-2}...|d_1$

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

(i) $\mathsf{T}$ spans greater than $\mathsf{U}$ ($\mathsf{T} = \mathsf{U}|(\mathsf{U}|c_p...|c_1)|d_n...|d_{k+m+1}$):[5]

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

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

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

[4] $\mathsf{T}$ 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.

[5] Here, the most general unifier is considered.

| Case | | Functor cat | | Input cat | | Result cat | | |
|---|---|---|---|---|---|---|---|---|
| | | Class | Outer seq | Class | $\|\text{input}\| \lessgtr k$ | Class | Residual variable | Unbounded const argument |
| | 6 | Const | - | Const | $\geq$ | Const | no | no |
| | 7*a* | GTRC | yes | Const | $\geq$ | GTRC | no | no |
| | 7*b* | GTRC | no | Const | $\geq$ | Const | no | no |
| | 8*a* | Const | - | GTRC | $>$ | Const | no | no |
| | 8*b* | Const | - | GTRC | $=$ | Const | no | possible |
| * | 8*c* | Const | - | GTRC | $<$ | neither | yes | possible |
| | 9*a* | GTRC | yes | GTRC | $>$ | GTRC | no | no |
| | 9*b* | GTRC | yes | GTRC | $=$ | GTRC | no | possible |
| * | 9*c* | GTRC | yes | GTRC | $<$ | neither | yes | possible |
| | 9*d*i | GTRC | no | GTRC | $>$ | GTRC | no | no |
| | 9*d*ii | GTRC | no | GTRC | $>$ | Const | no | no |
| | 9*e* | GTRC | no | GTRC | $=$ | GTRC | no | no |
| * | 9*f* | GTRC | no | GTRC | $<$ | neither | yes | possible |

Table A.1: Combinatory Cases for CCG-GTRC

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

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

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

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

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

$$\mathsf{T}/\underline{(\mathsf{T}|a_m...|a_2\backslash a_1)}\quad \underline{\mathsf{U}}|(\mathsf{U}|c_p...|c_1)|d_{k-1}...|d_1$$

$$\Longrightarrow\ \mathsf{T}|(\mathsf{T}|\overbrace{a_m...|a_2\backslash a_1|c_p...|c_1}^{\text{inner seq of GTRC}})|d_{k-1}...|d_1$$

Example: $\mathsf{T}/\underline{(\mathsf{T}\backslash NP)}\quad \underline{\mathsf{U}}/(\mathsf{U}\backslash NP) \Longrightarrow \mathsf{T}/(\mathsf{T}\backslash NP\backslash NP)$

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

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

$$\Longrightarrow\ \mathsf{T}|\underbrace{\mathsf{U}_1}_{\text{residual}}|(\mathsf{T}|a_m...|a_2\backslash a_1|\mathsf{U}_1|\overbrace{c_p...|c_1}^{\text{unbounded}})|d_{k-2}...|d_1$$

The three cases indicated by '$*$' in Table A.1 introduce categories that 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:

(10) **Closure Condition**: The rule "$x/\underline{y} \quad \underline{y}|z_k...|z_1 \implies x/|z_k...|z_1$" (and analogously for the other direction) must satisfy $\left|y|z_k...|z_1\right| \geq k$.

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.

**Framework: CCG-GTRC**

We define the most general form of CCG-GTRC$_0$ as follows:

**Definition 1** A CCG-GTRC$_0$ is a five tuple $(V_N, V_T, S, f, R)$ where

- $V_N$ is a finite set of nonterminals (atomic categories)
- $V_T$ is a finite set of terminals (lexical items, written as $\mathtt{a}, ..., \mathtt{z}$)
- $S$ is a distinguished member of $V_N$
- $T$ is a countable set of variables[6]
- $f$ is a function that maps elements of $V_T$ to finite subsets of "Const $\cup$ LTRC"[7]
- $R$ is a finite set of rule instances of Generalized Functional Composition observing Closure Condition (i.e., those summarized in Table A.1 except for those with '$*$').

CCG-GTRC$_0$ differs from CCG-Std in some crucial respects.

(11) *a*. The set of arguments is not bounded. Not only the inner sequence of GTRC is unbounded, but also an argument of a constant category can be unboundedly long.

[6]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.

[7]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].

*b*. Combinatory rules cannot be specified in a 'finite' manner as described in [Vijay-Shanker and Weir, 1994].[8] The reason is that both functor and input categories can be unboundedly long unlike CCG-Std.

From both complexity and parsing points of view, this situation seems to require more 'power' to deal with. The conjecture is that this grammar is not equivalent to CCG-Std nor polynomially parsable. What I will do in the following is to find a subclass of CCG-GTRC$_0$ that still satisfies the original motivation and can be proved weakly-equivalent to CCG-Std. We discuss the following three problems in turn: (i) the bound of the arguments of constant categories, (ii) mixed directionality in GTRC inner sequence, and (iii) the behavior of GTRC outer sequences.

First, we want to apply the same techniques of CCG-Std to the "Const+Const" case. For this purpose, the set of arguments must be bounded [Vijay-Shanker and Weir, 1994; Vijay-Shanker and Weir, 1990]. Thus, we place a bound on the length of an argument.

(12) **Bounded Argument Condition**: Every argument except for the inner sequence of GTRC must be bounded by the grammar.

Then, the rules indicated as 'unbounded argument' in Table A.1 must be restricted to those satisfying the condition while the inner sequence of GTRCs can grow without limit. We now have the following property:

(13) The set of arguments of a constant category and the set of arguments of the inner and outer sequences of a GTRC are all finite. We denote the set of all these arguments as Args.

An alternative to the Bounded Argument Condition is to place a bound on the length of GTRC inner sequence. But, then, we need to re-evaluate our assumption about the unbounded NP sequence and the system degenerates to CCG-Std since every instance of GTRC can be represented as a constant.

The new subclass of CCG-GTRC$_0$ is defined as follows:

**Definition 2** CCG-GTRC$_{bound_arg}$ is a subclass of CCG-GTRC where the Bounded Argument Condition is observed.

The second problem is with the mixed directionality of the GTRC. For example, consider a GTRC $\mathsf{T}/(\mathsf{T}/a_m.../a_2\backslash a_1)$ derived from "$\mathsf{T}/(\mathsf{T}\backslash a_1) \quad \Big(\mathsf{T}\backslash(\mathsf{T}/a_2) \quad ... \quad \mathsf{T}\backslash(\mathsf{T}/a_m)\Big)$". This

[8] This 'finiteness' corresponds to the instantiation of the input categories. The functor category of a combinatory rule still needs a meta-variable since categories can grow without limit.

may proceed with the following derivation: "$\mathsf{T}/\underline{(\mathsf{T}/a_m.../a_2\backslash a_1)}$ $\underline{c/a_m.../a_2\backslash a_1}|d_k...|d_1$". Although the input category, $c/a_m.../a_2\backslash a_1|d_k...|d_1$, seeks the arguments $a_2,...,a_m$ to its right, the arguments are actually found on the left of the category. In addition, although the GTRC $\mathsf{T}\backslash(\mathsf{T}/a_m)$ stands adjacent to $c/a_m.../a_2\backslash a_1|d_k...|d_1$, $a_m$ is unboundedly-deep in the category $c/a_m.../a_2\backslash a_1|d_k...|d_1$. In a sense, this difficulty corresponds to the mixture of non-order preserving type raising and the unbounded version of generalized functional composition so that $\mathsf{T}/(\mathsf{T}/a_m)$ can combine with $s|d_k...|d_1/a_m.../a_2\backslash a_1$ (i.e., no limit on $k_{max}$). The current position is to stipulate the following condition:

(14) **Unidirectional GTRC Condition**: The inner sequence of a GTRC must have the uniform directionality as in: $\mathsf{T}\Big\langle\Big(\mathsf{T}\Big\rangle a_m...\Big\rangle a_1\Big)|b_n...|b_1$.

This condition is closely related to the linguistic aspect of long-distance 'movement' across the functor. Our motivation does not depends on these phenomena. For example, the gapping conjuncts of two underlined NPs in the English sentence, "John helped Mary, Bill, Rose." might involve
$S/(S\backslash NP/NP)$ from "$S/(S\backslash NP)$ $(S\backslash NP)\backslash((S\backslash NP)/NP)$". But I believe that such a case is inherently bounded and does not require a GTRC involving variables. We define the following subclass:

**Definition 3** CCG-GTRC$_{uni}$ is a subclass of CCG-GTRC$_{bound_arg}$ where the Unidirectional GTRC Condition is observed. The third problem is related to 'quasi-island' condition exemplified as follows:

(15) *a*. CCG-GTRC: $S/A \quad \Big(\mathsf{T}\backslash(\mathsf{T}/A)/B \quad B\Big) \Longrightarrow S$

*b*. CCG-GTRC: $B \quad S/A \quad \overset{S}{\mathsf{T}}\backslash(\mathsf{T}/A)\backslash B \Longrightarrow *$

*c*. CCG-Std: $S/A \quad \Big(A/B \quad B\Big) \Longrightarrow S$

*d*. CCG-Std: $B \quad (S/A \quad A\backslash B) \Longrightarrow S$

With respect to the interaction with input categories of constant class, GTRCs behave like an island. But we do not have a general way in CCG-Std proper to *exactly* capture the effect. Our next step is to exclude outer sequence from the GTRCs altogether.

**Definition 4** CCG-GTRC$_{no_outer}$ is a subclass of CCG-GTRC$_{uni}$ where no GTRC has outer sequence.

This limits the instances of GTRCs to a finite set since the inner argument of a GTRC is 'frozen'. It can only act as its own.[9] Although the expressiveness is greatly limited, it can still represent the example we started with in addition to the coverage of CCG-Std.

These conditions may appear unnatural. But note that they are applied when the grammar is constructed and do not change the way the grammar is used to recognize a string in CCG-GTRC. Thus, they are legitimate way to 'define' subclasses of grammar. In the rest of this paper, we focus on CCG-GTRC$_{no_outer}$ and prove its weak equivalence to CCG-Std. The only relevant cases are now (6), (7$b$), (8$a,b$), and (9$d$ii$,e$) where no outer sequence of GTRC is present.

## A.2 Weak Equivalence of CCG-GTRC and CCG-Std

This section presents the proof of the equivalence of CCG-Std and CCG-GTRC$_{no_outer}$ (CCG-GTRC hereafter). Let $G_{std}$ and $G_{gtrc}$ be the classes of CCG-Std and CCG-GTRC, respectively. A grammar is represented by $G_{index}$ where the subscript is optionally used to distinguish grammars. The proposition to prove is the following:

**Proposition 1** $G_{gtrc}$ is weakly equivalent to $G_{std}$.

Since any $G \in G_{std}$ is also $G \in G_{gtrc}$ by definition, we only need to show that for each $G_{gtrc} \in G_{gtrc}$, there is a $G_{std} \in G_{std}$ such that $G_{gtrc}$ and $G_{std}$ generate the same language, i.e., $L(G_{gtrc}) = L(G_{std})$. The proof is by the following lemma with the start category set to $S$.

**Lemma 1** (Main Lemma) For any $G_{gtrc} \in G_{gtrc}$, there is a $G_{std-sim} \in G_{std}$ such that a terminal string `w` is generated by a constant category $c$ in $G_{gtrc}$ iff `w` is generated by $c'$ in $G_{std-sim}$ where $c'$ is the category in $G_{std-sim}$ corresponding to $c$ in $G_{gtrc}$.

We construct $G_{std-sim}$ from $G_{gtrc}$ so that $G_{std-sim}$ *simulates* $G_{gtrc}$.[10] The process starts by translating $G_{gtrc}$ to the base CCG-Std, $G_{std-base}$ as follows:

[9] The case (9$d$ii) may result in decomposition of the inner argument in a restricted way. This will be treated as Bounded GTRC in a later section.

[10] The word 'simulation' is also used to describe operations involved in the process.

(16) *a*. Copy all the constant categories in $G_{gtrc}$ to $G_{std-base}$ assigned to the same terminal symbol.

*b*. For each LTRC $\mathsf{T}\left\langle \left(\mathsf{T}\right\rangle a\right)$ in $G_{gtrc}$, add an atomic category $\langle a \rangle$ to the lexicon of $G_{std-base}$ assigned to the same terminal symbol.[11] Note that the use of an atomic category is to avoid decomposition of the inner argument. This is possible since the inner arguments of GTRC never unifies with a target category and are never decomposed in the current formulation.[12]

Then, $G_{std-base}$ is extended to $G_{std-sim}$ to simulate $G_{gtrc}$. This situation is shown schematically as follows:

(17)

| | translation | | |
|---|---|---|---|
| $G_{gtrc}$ | $\longrightarrow$ | $G_{std-base}$ | |
| | $\nwarrow$ | $\downarrow$ | extension |
| simulation | | $G_{std-sim}$ | |

Since CCG-GTRC extends the way CCG captures phenomena including unbounded, but restricted 'permutation', it is crucial to identify the properties of GTRCs and provide appropriate methods for simulation. Once we have the right simulation, the equivalence can be shown by the set inclusion for both directions by invoking the simulation as needed. Two simulation techniques, 'wrapping' and 'bounded GTRC', and the proof of both directions will be described in the following.

**Wrapping**

CCG-GTRC allows permutation, as observed in the following example:

[11] The directionality of LTRC can be captured by features such as '$-left$' or '$-right$'. We will ignore this aspect for simplicity.

[12] This depends on the 'no-outer sequence' condition.

(18) *a.*

| a | b | c |
|---|---|---|
| $\mathsf{T}/(\mathsf{T}\backslash A)$ | $\underline{\mathsf{T}/(\mathsf{T}\backslash B)}$ | $\underline{S\backslash A\backslash B}$ |
| | $S\backslash A$ | |
| | $S$ | |

*b.*

| b | a | c | (permutation) |
|---|---|---|---|
| $\mathsf{T}/(\mathsf{T}\backslash B)$ | $\underline{\mathsf{T}/(\mathsf{T}\backslash A)}$ | $\underline{S\backslash A\backslash B}$ | |
| | $S\backslash B$ | | |
| | $S$ | | |

First, we attempt to simulate such a permutation by *wrapping* the arguments of a lexical category [the idea has been around for a while, e.g., Bach, 1979; Dowty, 1979]. For example, '$\backslash A$' in $S\backslash A\backslash B$ can wrap across '$\backslash B$' with the result of $S\backslash B\backslash\langle A\rangle$. We use '$\langle\rangle$' to represent the wrapped argument as an atomic category that will unify with the GTRC-translated category also represented in the same way. This corresponds to the permutation of (18*b*) as follows:

(19) *a.*

| b | a | c | (CCG-GTRC) |
|---|---|---|---|
| $\mathsf{T}/(\mathsf{T}\backslash B)$ | $\underline{\mathsf{T}/(\mathsf{T}\backslash A)}$ | $\underline{S\backslash A\backslash B}$ | |
| | $S\backslash B$ | | |
| | $S$ | | |

*b.*

| b | a | c | (CCG-Std) |
|---|---|---|---|
| $B$ | $\langle A\rangle$ | $S\backslash B\backslash\langle A\rangle$ | |
| | $S\backslash B$ | | |
| | $S$ | | |

The above-mentioned technique of wrapping arguments only applies to local permutation within a lexical category. But CCG-GTRC allows permutations across lexical categories, as seen below.

(20)

| | | |
|---|---|---|
| $\mathsf{T}/(\mathsf{T}\backslash A)$ | $\underline{T\backslash B}$ | $\underline{S\backslash A\backslash T}$ |
| | $S\backslash A\backslash B$ | |
| | $S\backslash B$ | |

Since we assume that GTRCs can compose without limit, there is no bound on the composition of the input to GTRCs.

(21) $$\dfrac{\dfrac{\mathsf{T}/\underline{(\mathsf{T}\backslash A_n...\backslash A_2)} \quad \underline{T}\backslash A_1 \quad ... \quad \underline{\dfrac{T\backslash A_{n-1}\backslash T \quad S\backslash A_n\backslash \underline{T}}{S\backslash A_n\backslash A_{n-1}\backslash \underline{T}}}}{\underline{S\backslash A_n...\backslash A_2}\backslash A_1}}{S\backslash A_1}$$

Then, we want to obtain a wrapped category like $S\backslash A_1\backslash\langle A_n\rangle...\backslash\langle A_2\rangle$. This situation can be captured by using the technique of *argument passing* as follows:[13]

(22) *a.* $S\backslash B\backslash\langle A\rangle \Longleftarrow \underline{T^{\{\backslash B\}}} \quad S\backslash B\backslash\langle A\rangle\backslash\underline{T^{\{\backslash B\}}}$

Note: Since subscripts are frequently used for indexing the categories in this paper, the features are placed as superscript.

*b.* $S\backslash A_1\backslash\langle A_n\rangle...\backslash\langle A_2\rangle \Longleftarrow$
$$\left(T^{\{\backslash A_1\}} \quad \left(T^{\{\backslash A_1\}}\backslash\langle A_2\rangle\backslash T^{\{\backslash A_1\}} \quad +s \quad \left(\underline{T^{\{\backslash A_1\}}}\backslash\langle A_{n-1}\rangle\backslash T^{\{\backslash A_1\}} \quad S\backslash A_1\backslash\langle A_n\rangle\backslash\underline{T^{\{\backslash A_1\}}}\right)\right)\right)$$

The arguments that are crossed by wrapping are placed as a feature on the target category and on the first argument. They are then passed on to the category corresponding to a deeper position of the composed category. As in the case of '$\langle\rangle$', we consider the category with passed arguments as an atomic category. This also applies for the case where the canceled category is complex such as: $(S/A)^{\{/C\}}/B$.

This simulation depends on the fact that the list of passed arguments is bounded. First, observe $(a)$ below. A particular argument can be crossed by any number of arguments by wrapping, which is the source of unbounded permutation. On the other hand, an argument can cross only a finite number of other arguments by wrapping, as seen in $(b)$. This latter case is bounded by $k_{max}$ of functional composition.

(23) *a.*
$$\begin{array}{ccccccccc} B & \mathsf{T}/(\mathsf{T}\backslash A_n) & +s & \mathsf{T}/(\mathsf{T}\backslash A_1) & S & \backslash A_n...\backslash A_1 & |B & \\ & & & & & & | & \\ B & \langle A_n\rangle & +s & \langle A_1\rangle & S & & |B & \backslash\langle A_n\rangle...\backslash\langle A_1\rangle \end{array}$$

[13] Argument passing is conceptually similar to the techniques found in grammar formalism and logic including: SLASH feature of GPSG/HPSG [Gazdar et al., 1985; Pollard and Sag, 1994] and assume/discharge of natural deduction [Hepple, 1990]. But it is finite and limited in its power.

$b.\quad \mathsf{T}/(\mathsf{T}\backslash A) \qquad S \quad \backslash A \quad |B_k...|B_1$

$\qquad\qquad\qquad\qquad\qquad\qquad\quad |$

$\qquad\quad \langle A\rangle \qquad\quad S \qquad\quad |B_k...|B_1 \quad \backslash\langle A\rangle$

Recall that the set of arguments Args is bounded. Thus, at any juncture of rule application, there are only finitely many possibility of argument passing. We add all these cases to the lexicon.

To describe wrapping concisely, we introduce the following notation: Depending on how we divide a category into the 'function' and the 'arguments', a category $c = F|a_m...|a_1$ can be viewed with different valencies, i.e., $c = \underset{F}{f_m}|a_m...|a_1$,..., $c = \underset{F|a_m...|a_{i+1}}{f_i}|a_i...|a_1, c = \underset{F|a_m...|a_2}{f_1}|a_1, c = \underset{F|a_m...|a_1}{f_0}$. Let us refer to $f_i$ as the *functional forms* of $c$. The functional forms with every valency can then be represented as follows: $c = F|a_m...|a_1 = f_i\mathbb{A}_i$ where $0 \leq i \leq m$ and $\mathbb{A}_i = |a_i...|a_1$.

The process of wrapping is now presented as follows:

(24) **Wrapping**: Consider functional forms of a lexical category $c = F|a_m...|a_1 = f_i[\mathbb{A}_i|a_1]$ where $\mathbb{A}_i = |a_i...|a_2$ and '[ ]' indicates the optionality. In case $a_1$ is not null, consider all the possible sequence of arguments $\mathbb{I} = |d_k...|d_1$ (as passed arguments) where $|\mathbb{I}| \leq k_{max}$. For a concatenation of $\mathbb{A}_i\mathbb{I}$ (including $|\mathbb{I}| = 0$), apply all the possible wrapping. Note the use of $\langle a_i\rangle$ to represent the wrapped argument $a_i$. Optionally, designate the last $j \leq k_{max}$ arguments as $\mathbb{O}$, and place them as the feature on $f_i$. The process can be abbreviated as follows: $f_i\mathbb{A}_i|a_1 \twoheadrightarrow f_i^{\{\mathbb{O}\}}\mathbb{A}_i'|a_1^{\{\mathbb{I}\}}$ where $A_i'$ is obtained by wrapping as described above and the both categories are assigned to the same terminal. Categories with passed arguments are considered atomic categories.

Categories including a wrapped argument and/or a passed argument, do not interact with constant category until these features are canceled. For example, the following unintended cases all fail.

(25) *a.* $\underline{D} \quad \underline{S\backslash B\backslash\langle D\rangle} \implies *$

*b.* $C/\underline{(S\backslash B\backslash A)} \quad \underline{S\backslash C\backslash\langle A\rangle} \implies *$

*c.* $\underline{S}\backslash B \quad \underline{S\backslash B\backslash A\backslash S^{\{\backslash B\}}} \implies *$

The use of '⟨⟩' avoids overgeneration of the following kind as well:

(26) *a.* $S/C/\underline{A} \quad \underline{\left(A/B \quad B\right)} \implies S/C$ (potential overgeneration)

*b.* $S/C/\underline{\langle A\rangle} \quad \underline{\left(\langle A/B\rangle \quad B\right)} \implies *$ (implemented)

**Bounded GTRC**

When GTRCs appear as input category, their instances are bounded, as shown in (13). Thus, we can replace the variables with constants. For example, suppose that coordination is lexical, defined for each instance of conjunct category, and the set of conjuncts is bounded. Coordination of non-traditional constituents might need the conjunctive category like $\left(S/\left(S\backslash NP\backslash NP\right)\right)/\left(S/\left(S\backslash NP\backslash NP\right)\right)\backslash\left(S/\left(S\backslash NP\backslash NP\right)\right)$. Then, we can derive $S/\left(S\backslash NP\backslash NP\right)$ as "$S/\left(S\backslash NP\right)\quad\left(S\backslash NP\right)/\left(S\backslash NP\backslash NP\right)$". Both of the instances must be added to the lexicon since $G_{std-sim}$ has no other way to represent this non-traditional constituency. Since we are motivated to deal with unboundedly-long inner sequence of GTRC, we cannot apply this technique to (9*e*). Wrapping has been introduced for this purpose. The procedure of adding GTRC instances is described as follows:

(27) **Bounded GTRC**:

(8*a*): Suppose that the whole GTRC $\mathsf{T}\langle\backslash\left(\mathsf{T}\rangle a_m\ldots\rangle a_1\right)$ unifies with some argument of a category, i.e., a member of the set of arguments Arg in $G_{gtrc}$. The GTRC must be derived uniquely from a sequence of LTRCs, $\mathsf{T}_m/\left(\mathsf{T}_m\backslash a_m\right),\ldots,\mathsf{T}_1/\left(\mathsf{T}_1\backslash a_1\right)$ or $\mathsf{T}_1\backslash\left(\mathsf{T}_1/a_1\right),\ldots,\mathsf{T}_m\backslash\left(\mathsf{T}_m/a_m\right)$, depending on the directionality (cf. **Lemma 3**).[14] Add the ground instances of the LTRCs to the lexicon of $G_{std-sim}$.

(8*b*): Since we have set a bound on the instances on $\left(b\rangle c_m\ldots\rangle c_1\right)$, add the LTRCs that derives $b\langle\left(b\rangle c_m\ldots\rangle c_1\right)$.

(9*d*ii): The only possibility is the following:

"$\mathsf{T}/\underline{\left(\mathsf{T}\backslash a\right)}\quad\underbrace{\underbrace{\mathsf{U}}_{\mathsf{T}}|\underbrace{\left(\mathsf{U}|c_p\ldots|c_1\right)}_{a=F|a_m\ldots|a_1}}\rightarrow F|a_m\ldots|a_{m-p}|d_k\ldots|d_1$". The functor category must be an LTRC and the instances of $a$ is bounded. We add those instances in the lexicon.

**Proof:** $L\left(G_{gtrc}\right)\subseteq L\left(G_{std-sim}\right)$

Now the simulation is established for the given CCG-GTRC. The proof of the direction from $G_{gtrc}$ to $G_{std-sim}$ is by induction on the height $h$ of a derivation in $G_{gtrc}$. The primary recursion (for both directions) deals only with constant categories (of CCG-GTRC) since we are concerned with

[14] This can be proved by induction on the length of the inner sequence.

derivations of a constant category, $S$ in particular. The current direction also involves GTRCs as the source derivation and these are handled by **Lemma 3** and wrapping handled by **Lemma 4** introduced below. The latter lemma sets a mutually-recursive situation with this direction of the main lemma (**Lemma 2**).

**Lemma 2** The direction $L(G_{gtrc}) \subseteq L(G_{std-sim})$ of the Main Lemma.

**Base case** ($h = 0$): $c$ is a lexical category. Then, $c$ is also in $G_{std-sim}$ assigned to the same terminal symbol.

**Induction hypothesis** (IH2): The lemma holds for all $h' \leq h-1$.

**Induction step** ($h \geq 1$): We only consider the following relevant cases where the result category is Const.

(28) *a*. (Const+Const, 6) The same derivation is available in $G_{std-sim}$. For the left and right categories, which are constant categories of smaller height, apply the induction hypothesis (IH2). The pair of strings obtained by the application of the induction hypothesis in the same order can be concatenated to provide the desired string in $G_{std-sim}$.

*b*. (GTRC+Const, 7*b*) $\mathsf{T}/\underline{(\mathsf{T}\backslash a_m ... \backslash a_1)} \quad \underbrace{c}_{\underline{c_0|c_m...|c_1}} \quad |d_k...|d_1 \implies c_0|d_k...|d_1$

This case requires the simulation. Note that $c$ is unbounded. **Lemma 4** provides us the wrapped form $c_0|d_k...|d_1\backslash a_m...\backslash a_1$ from $c_0\backslash a_m...\backslash a_1|d_k...|d_1$. **Lemma 3** shows that there is a sequence of categories with the corresponding string that can combine with $c_0|d_k...|d_1\backslash a_m...\backslash a_1$ in the same order with the same string. Thus, after applying each category of the sequence to the wrapped category, we have the desired result $c_0|d_k...|d_1$ with the same string.

*c*. (Const+GTRC, 8*a*) $a/\underline{b} \quad \underline{\mathsf{T}|(\mathsf{T}|c_m...|c_1)} \implies a$

Since the GTRC is bounded, we have the corresponding category in $G_{std-sim}$ by (27). The rest is similar to the previous case.

*d*. (Const+GTRC, 8*b*) $a/\underline{b} \quad \underline{\mathsf{T}}|(\mathsf{T}|c_m...|c_1) \implies a|(b|c_m...|c_1)$

Since the GTRC is bounded by the stipulated Bounded Argument Condition, we have the corresponding category in $G_{std-sim}$ by (27). The rest is similar to $(a)$.

*e*. (GTRC+GTRC, 9*d*ii) $\mathsf{T}/(\mathsf{T}\backslash \underbrace{a}_{\underbrace{a_0|a_p...|a_1}}) \quad \underline{\mathsf{U}|(\mathsf{U}|c_p...|c_1)} \implies a_0$

Since $a$ is bounded, the process is similar to the previous case.

■

**Lemma 3** If the derivation of $\mathsf{T}/(\mathsf{T}\backslash a_m...\backslash a_1)$ from the string $\mathtt{w}$ can combine with $x\backslash a_m...\backslash a_1$ in $G_{gtrc}$, $\langle a_m\rangle,...,\langle a_1\rangle$ that is associated with the same terminal string can combines with $x\backslash y_m...\backslash y_1$ for some $x$ in $G_{std-sim}$ where $y_i$ may be $a_i$ or $\langle a_i\rangle$.

Proof: By induction on the height $h$ of derivation.[15]

**Base case** ($h=0$): The category must be an LTRC, $\mathsf{T}/(\mathsf{T}\backslash a)$. Thus, there is $\langle a\rangle$ and $a$ assigned to the same terminal in $G_{std-sim}$ by the simulation. Then, either $\langle a\rangle$ or $a$ can combine with $x\backslash a$ or $x\backslash\langle a\rangle$ as desired.

**Induction hypothesis**: The lemma holds for $h' \leq h-1$.

**Induction step** ($h \geq 1$): The GTRC $\mathsf{T}/(\mathsf{T}\backslash a_m...\backslash a_1)$ must be derived as "$\mathsf{T}/\underline{(\mathsf{T}\backslash a_m...\backslash a_{i+1})} \quad \underline{\mathsf{U}}/(\mathsf{U}\backslash a_i...\backslash a_1) \rightarrow \mathsf{T}/(\mathsf{T}\backslash a_m...\backslash a_1)$" for some $i$ (9*e*). Apply the induction hypothesis to the input category. Then, we have a sequence of $\langle a_i\rangle,...,\langle a_1\rangle$, which generates the same string as $\mathsf{U}/(\mathsf{U}\backslash a_i...\backslash a_1)$. Since each of $\langle a_i\rangle$ has the corresponding $a_i$, the sequence can apply to $x'\backslash y_i...\backslash y_1$ in series to derive some $x' = x\backslash y_m...\backslash y_{i+1}$ in $G_{std-sim}$. Next, apply the induction hypothesis to the functor category and $x\backslash y_m...\backslash y_{i+1}$ to obtain $x$ in $G_{std-sim}$ from the same string as desired.

■

**Lemma 4** Consider a category $c|a_m...|a_1|d_k...|d_1$ derivable in $G_{gtrc}$ where $k \leq k_{max}$ and $m \geq 0$. If this category combines with a GTRC $\mathsf{T}/(\mathsf{T}\backslash a_m...\backslash a_1)$ to reduce to "$\mathsf{T}/(\mathsf{T}\backslash a_m...\backslash a_1) \quad c\backslash a_m...\backslash a_1|d_k...|d_1 \implies c|d_k...|d_1$", then there is a category $c|d_k...|d_1\backslash y_m...\backslash y_1$ in $G_{std-sim}$ where $y_i$ is either $a_i$ or $\langle a_i\rangle$, which generates the same terminal string as $c\backslash a_m...\backslash a_1|d_k...|d_1$ in $G_{gtrc}$.

The proof is by the following lemma that is a more general version.

[15] Induction on the length of the inner sequence also works for this case.

**Lemma 5** Consider a category $x := c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e$ in $G_{gtrc}$ where $k \geq 1$, $k \leq k_{max}$, $m \geq 0$, '$|e$' may be nil. $c$ and $e$ may be associated with passed arguments as a feature. If "$\mathsf{T}/(\mathsf{T}\backslash a_m ... \backslash a_1)$ $\quad c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e \implies c | d_{k-1} ... | d_1 | e$", there is a category $y := c^{\{\mathbb{O}\}} | b_j ... | b_1 \backslash \langle a_m \rangle ... \backslash \langle a_1 \rangle | e^{\{\mathbb{I}\}}$ in $G_{std-sim}$ where $j \leq k_{max}$, $m \geq 0$, $\mathbb{O}$ and $\mathbb{I}$ are sequences of arguments shorter than $k_{max}$ (possibly nil) such that $y$ derives the same terminal string as $x$.

Proof: By induction on the height $h$ of derivation.

**Base case** ($h = 0$): $c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e$ is a lexical category. By (24), there is $c | d_{k-1} ... | d_1 \backslash \langle a_m \rangle ... \backslash \langle a_1 \rangle | e$ in $G_{std-sim}$ which is associated with the same terminal.

**Induction hypothesis**: The lemma holds for $h' \leq h - 1$.

**Induction step** ($h \geq 1$): Consider the following cases:

(29) *a*. Reduction: $c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e \Longleftarrow c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_{p+1} / f \quad f | d_p ... | d_1 | e$

By induction hypothesis, $c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_{p+1} / f$ has the corresponding $c | d_{k-1} ... | d_{p+1} | d_p ... | d_1 \backslash \langle a_m \rangle ... \backslash \langle a_1 \rangle / f^{\{|d_p ... |d_1\}}$ which generates the same string, and $f | d_p ... | d_1 / e$ has the corresponding $f^{\{|d_p ... |d_1\}} | e$ which generates the same string. This case may involve a GTRC as the input category ($p = 0$). But such a case is limited to a bounded form. We can thus consider the bounded instances as if they are constants.

*b*. Reduction: $c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e \Longleftarrow c\backslash a_m ... \backslash a_{i+1} / f \quad f \backslash a_i ... \backslash a_1 | d_{k-1} ... | d_1 | e$

By induction hypothesis, $c\backslash a_m ... \backslash a_{i+1} / f$ has $c | d_{k-1} ... | d_1 \backslash \langle a_m \rangle ... \backslash \langle a_{i+1} \rangle / f^{\{|d_{k-1} ... |d_1\}}$, and $f \backslash \langle a_i \rangle ... \backslash \langle a_1 \rangle | d_{k-1} ... | d_1 / e$ has $f^{\{|d_{k-1} ... |d_1\}} \backslash a_i ... \backslash a_1 | e$.

*c*. Reduction: $c\backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e \Longleftarrow c / f \quad f \backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 | e$

By induction hypothesis, $f \backslash a_m ... \backslash a_1 | d_{k-1} ... | d_1 / e$ has $f | d_{k-1} ... | d_1 \backslash \langle a_m \rangle ... \backslash \langle a_1 \rangle | e$. By the induction hypothesis of the main lemma (IH2) there is a constant category that generates the same string as $c / f$.

■

**Proof:** $L(G_{gtrc}) \supseteq L(G_{std-sim})$

We will use the following classification for the categories in $G_{std-sim}$.

(30) *a*. $\text{Const}_2$: Categories translated from Const of $G_{gtrc}$. Exclusive of the following.

*b*. GTRC: Categories translated from GTRC of $G_{gtrc}$. Represented as $\langle x \rangle$.

*c*. Wrap: Categories obtained by Wrapping. They may include wrapped argument represented as $\langle x \rangle$ and/or passed argument $\{\mathbb{P}\}$.

*d*. BGTRC: Categories obtained by Bounded GTRC.

Note that 'Const$_2$' in this classification stands in relation to 'Const' in $G_{gtrc}$ and that all the categories in $G_{std-sim}$ are *constant*. We will drop the subscript on Const$_2$ where no confusion arises.

The proof is by induction on the height $h$ of a derivation in $G_{std-sim}$. The primary recursion is on Const and we introduce **Lemma 7** to have a mutually-recursive situation on wrapped categories.

**Lemma 6** The direction $L(G_{gtrc}) \supseteq L(G_{std-sim})$ of the Main Lemma.

**Base case** ($h = 0$): By the definition of Const$_2$ above, there is a corresponding constant lexical category with the same terminal string in $G_{gtrc}$.

**Induction hypothesis** (IH6): The lemma holds for $h' \leq h - 1$.

**Induction step** ($h \geq 1$): We consider the following cases that result in Const.

(31) *a*. Const+Const: Apply the induction hypothesis (IH6) to the functor and input categories. Then, the same strings can be generated from the corresponding categories in $G_{gtrc}$. Since we can apply the same rule in $G_{gtrc}$, we generates the same string from the same category.

*b*. Const+BGTRC, BGTRC+Const, and BGTRC+BGTRC: By the simulation, any bounded instance of GTRC in $G_{std-sim}$ has the corresponding GTRC in $G_{gtrc}$. Apply IH6 to the Const. Then, this case has the corresponding derivation. Note that there is no formal distinction between BGTRC and Const. Thus, there may be ambiguous case where a single derivation may need be considered for both cases, where only one of them may apply.

*c*. Const+Wrap, Wrap+Const, Wrap+BGTRC, BGTRC+Wrap: These cases do not apply. Regardless of the position of the indication of wrapping (either $\langle x \rangle$ or passed argument), either they fail to unify with the other category or would remain in the result category.

*d*. GTRC+<any class>, BGTRC+GTRC, Const+GTRC: Not applicable. GTRC-translated category $\langle x \rangle$ can only combine with the identical argument of a wrapped category.

*e*. Wrap+Wrap: The only applicable case is the following: "$a/b^{\{\mathbb{P}\}} \quad b^{\{\mathbb{P}\}}\mathbb{C} \implies a\mathbb{C}$" (other instances of wrapping are not applicable for the same reason as (3)). Apply **Lemma 7** to both categories.

*f*. Wrap+GTRC: The rule application takes the form: "$a/\langle b \rangle \quad \langle b \rangle \implies a$". By the simulation, the same string can be generated by the corresponding categories in $G_{gtrc}$.

■

For the case where the result category is Wrap, consider the following lemma.

**Lemma 7** For a wrapped category $c$ in $G_{std-sim}$, there is a constant category $c'$ in $G_{gtrc}$ that generates the same terminal string.

Proof: By induction on the height $h$ of derivation.

**Base case** ($h = 0$): $c$ is a lexical category in $G_{std-sim}$. There must be a category $c'$ in $G_{gtrc}$ by wrapping (24).

**Induction hypothesis**: The lemma holds for $h' \leq h - 1$.

**Induction step** ($h \geq 1$):

(32) *a*. Wrap+Wrap: The derivation takes the form: "$f^{\{\mathbb{O}\}}\mathbb{A}/b^{\{\mathbb{P}\}} \quad b^{\{\mathbb{P}\}}\mathbb{C}|d^{\{\mathbb{I}\}} \implies f^{\{\mathbb{O}\}}\mathbb{A}\mathbb{C}|d^{\{\mathbb{I}\}}$". Either $\mathbb{O}$ or $\mathbb{I}$ is non-nil. Apply the induction hypothesis to both categories. We have the corresponding $f\mathbb{O}\mathbb{A}'/b$ and $b\mathbb{P}\mathbb{C}'|d$ where $\mathbb{A}'$ and $\mathbb{C}'$ are the result of removing $\mathbb{P}$ and $\mathbb{I}$ from $\mathbb{A}$ and $\mathbb{C}$, respectively. They can derive: "$f\mathbb{O}\mathbb{A}'/b \quad b\mathbb{P}\mathbb{C}'|d \implies f\mathbb{O}\mathbb{A}'\mathbb{P}\mathbb{C}'|d = f\mathbb{O}\mathbb{A}\mathbb{C}'|d$".

*b*. Const+Wrap, Wrap+Const: Apply IH6 to Const and the induction hypothesis to Wrap. The rest is similar to the above.

*c*. No other case can result in Wrap.

■

**Example of Simulation**

**Example 1** English heavy NP-shift

"John gave the book to Mary."

"[John gave to Mary] **the book which ....**"

$$f_{gtrc} = \left\{ \begin{array}{l} (\texttt{john}, NP), \left(\texttt{john}, \mathsf{T}\left\langle \left(\mathsf{T}\right\rangle NP\right)\right), (\texttt{the book}, NP), \left(\texttt{the book}, \mathsf{T}\left\langle \left(\mathsf{T}\right\rangle NP\right)\right), \\ (\texttt{to mary}, PP), (\texttt{to mary}, \mathsf{T}\backslash(\mathsf{T}/PP)), (\texttt{gave}, S\backslash NP/PP/NP), \ldots \end{array} \right\}$$

$$f_{std}^{base} = \left\{ \begin{array}{l} \textit{replace the GTRCs with } (\texttt{john}, \langle NP\rangle), (\texttt{the book}, \langle NP\rangle), (\texttt{to mary}, \langle PP\rangle), \\ \textit{the rest is the same} \end{array} \right\}$$

$$f_{std}^{sim} = \left\{ \begin{array}{l} \textit{add the following to the above} \\ (\texttt{gave}, S\backslash NP/NP/\langle PP\rangle), (\texttt{gave}, S/NP\backslash\langle NP\rangle/\langle PP\rangle), \ldots \end{array} \right\}$$

$$\dfrac{\dfrac{\text{John} \quad \text{gave} \quad \text{to Mary} \quad \text{the book which ....}}{\langle NP\rangle \quad S/NP\backslash\langle NP\rangle/\langle PP\rangle \quad \langle PP\rangle \quad NP}}{S/NP\backslash\langle NP\rangle}$$

$$S/NP$$

**Example 2** Japanese long-distance extraction

"Mary-nom John-nom Mary-acc helped-comp thought."

"**Mary-acc** [Mary-nom John-nom helped-comp thought]."

$$f_{gtrc} = \left\{ \begin{array}{l} (\texttt{john-nom}, NP_{nom}), (\texttt{mary-nom}, NP_{nom}), (\texttt{john-acc}, NP_{acc}), (\texttt{mary-acc}, NP_{acc}), \\ (\texttt{john-nom}, \mathsf{T}/(\mathsf{T}\backslash NP_{nom})), (\texttt{mary-nom}, \mathsf{T}/(\mathsf{T}\backslash NP_{nom})), \\ (\texttt{john-acc}, \mathsf{T}/(\mathsf{T}\backslash NP_{acc})), (\texttt{mary-acc}, \mathsf{T}/(\mathsf{T}\backslash NP_{acc})), \\ (\texttt{helped}, S\backslash NP_{nom}\backslash NP_{acc}), (\texttt{comp}, S'\backslash S), (\texttt{thought}, S\backslash NP_{nom}\backslash S'), \ldots \end{array} \right\}$$

$$f_{std}^{base} = \left\{ \begin{array}{l} \textit{replace the GTRCs with } (\texttt{john-nom}, \langle NP_{nom}\rangle), (\texttt{mary-nom}, \langle NP_{nom}\rangle), \\ (\texttt{john-acc}, \langle NP_{nom}\rangle), (\texttt{mary-acc}, \langle NP_{nom}\rangle), \\ \textit{the rest is the same} \end{array} \right\}$$

$$f_{std}^{sim} = \left\{ \begin{array}{l} \textit{add the following to the above} \\ (\texttt{helped}, S\backslash NP_{acc}\backslash\langle NP_{nom}\rangle), \left(\texttt{helped}, S^{\{\backslash NP_{acc}\}}\backslash\langle NP_{nom}\rangle\right), \\ \left(\texttt{comp}, S'^{\{\backslash NP_{nom}\}}\backslash S^{\{\backslash NP_{nom}\}}\right), \\ \left(\texttt{thought}, S\backslash NP_{nom}\backslash NP_{acc}\backslash S'^{\{NP_{acc}\}}\right), \\ \left(\texttt{thought}, S\backslash NP_{acc}\backslash\langle NP_{nom}\rangle\backslash S'^{\{NP_{acc}\}}\right), \ldots \end{array} \right\}$$

| Mary-acc | Mary-nom | John-nom | helped | -comp | thought |
|---|---|---|---|---|---|
| $NP_{acc}$ | $\langle NP_{nom}\rangle$ | $\langle NP_{nom}\rangle$ | $S^{\{\backslash NP_{acc}\}}\backslash\langle NP_{nom}\rangle$ | $S^{\prime\{NP_{acc}\}}\backslash S^{\{NP_{acc}\}}$ | $S\backslash NP_{acc}\backslash\langle NP_{nom}\rangle\backslash S^{\prime\{NP_{acc}\}}$ |
| | | | | $S\backslash NP_{nom}\backslash\langle NP_{nom}\rangle\backslash S^{\{NP_{acc}\}}$ | |
| | | | $S\backslash NP_{acc}\backslash\langle NP_{nom}\rangle\backslash\langle NP_{nom}\rangle$ | | |

## A.3 Worst-Case Polynomial Recognition Algorithm

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.

**Polynomial Algorithm for CCG-Std**

First, observe the following properties of CCG categories:

(33) *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 "$\underbrace{\mathtt{a}....\mathtt{a}}_{n}$", the top CKY-cell includes $2^n$ combinations of categories like $S\underbrace{\left\{\begin{smallmatrix}/NP\\/PP\end{smallmatrix}\right\}\ldots\left\{\begin{smallmatrix}/NP\\/PP\end{smallmatrix}\right\}}_{n}/S$ 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 '$F$ this portion is *linked* in a cell $[p,q]$ with *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 "$F\underbrace{|a_n...|a_2}_{\bigstar}|\underline{a_1} \quad \underline{b_0}|b_k...|b_1 \Longrightarrow F\underbrace{|a_n...|a_2}_{\bigstar}|b_k...|b_1$", 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:

(34) **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 |
|---|---|---|---|
| $\lvert result \rvert$ | Link info | | |
| $< limit$ | none | No link | $F\lvert a_n...\lvert a_1$ |
| $< limit$ | available | Pass link info | $F\boxed{\ }\lvert a_i...\lvert a_1 \rightarrow F\boxed{\ }\lvert a_i...\lvert a_1$ |
| $\geq limit$ | either | Set up a new link | $F\boxed{\lvert a_n...\lvert a_i+1}\lvert a_i...\lvert a_1$ ↓ $\boxed{\ }$ |
| $= 0$ | available | Restore the linked info | $\boxed{\lvert a_i...\lvert a_1}$ ↓ $F\boxed{\ }$ |

**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 that does not depend on this property, we pursue a straightforward extension of Poly-Std with an additional condition on the rules.[16] In the rest of this section, we will concentrate on the subclass of CCG-GTRC constrained by the Bounded Argument Condition.

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 (9*e*) , "GTRC+GTRC", repeated below:

(35) $\mathsf{T}/\underset{\substack{\uparrow \\ (k \geq 1)}}{(\mathsf{T}|a_m...|a_2\backslash a_1)}$ $\quad \underline{\mathsf{U}}|(\mathsf{U}|c_p...|c_1)|d_{k-1}...|d_1 \implies \mathsf{T}|(\mathsf{T}|a_m...|a_1|c_p...|c_1)|d_{k-1}...|d_1$

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 encodes a kind of grammar for deriving the inner sequence and is thus considered 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 (7*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

[16]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.

process as an iteration of (backward) functional application of the form "$\underline{a_i}$ $\quad c_0|c_m...|\underline{c_i}$" 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 (7) 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.

(36)
$$\begin{array}{ccccc} C_m & +s & C_2 & & \{c_1\} \\ \updownarrow & & \updownarrow & \longleftarrow & \updownarrow \\ A_m & +s & A_2 & & \{a_1\} \end{array}$$

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) (7*b*): GTRCs can be ignored since they do not affect the recovery process, (ii) (8*a*), (9*a*): GTRCs are bounded, and (iii) (9*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.

(37) $\mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_2/\underline{b_1} \quad \underline{c}|d_k...|d_1 \implies \mathsf{T}|(\mathsf{T}|a_m...|a_1)|b_n...|b_2|d_k...|d_1$

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.

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

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

We need to recover the contents of the GTRC in both cases. The GTRC in (38*a*) is bounded since category *b* in the functor category is bounded. The one in (38*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":

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

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

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

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

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

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

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

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

The cases $(a)$ and $(b)$ are analogous to (38*a*) and (38*b*). 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 $\mathsf{T}$ spans greater than $\mathsf{U}$, 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

**Initialization**:

- Create an $n^2$ GTRC recovery table, $R$

**Table setup** (Stage 1):

- For each cell (top-down) $O(n^2)$
  - For each entry (depending on the midpoint) $O(n)$
    - Restore the derivation info from CKY table and store the children in the appropriate cells $O(n^2)$

**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) $O(n^2)$
  - For each entry $O(n)$
    - For each parent $O(n^2)$
      - 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 A.1: GTRC Recovery Algorithm

from the remaining part of the inner sequence of the input category with the remaining sequence "$|d_k \ldots |d_1$".

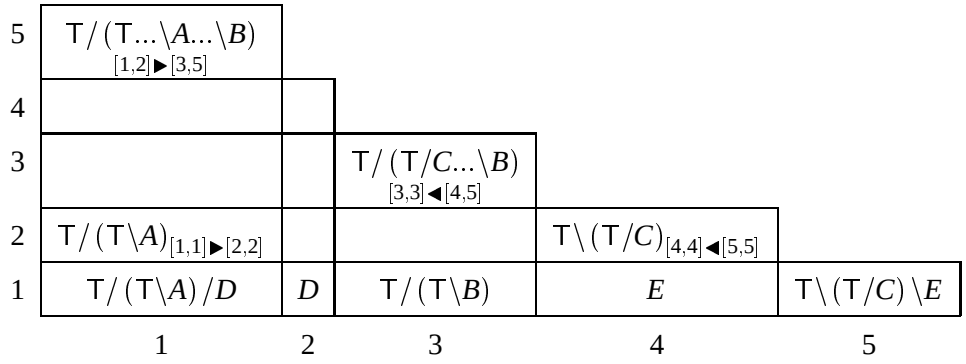
The case $(d\text{ii})$ is slightly different from the previous one in that the inner sequence of the functor category is excessive. We need a process of comparing the inner sequences of the functor and the input categories. Two GTRC recovery processes must be run in parallel.

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 A.1. An example of GTRC recovery is given in Appendix B of Komagata [1997b].

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

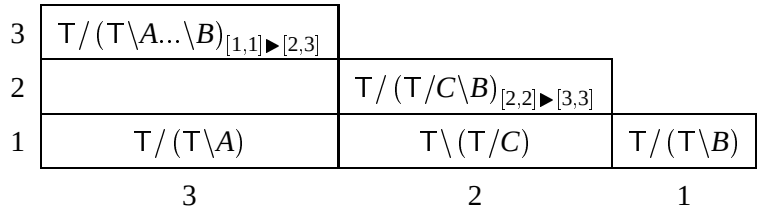
time. The first stage (*table setup*) is to represent the derivational structure available in the CKY table in a slightly different way. Suppose the following partial CKY table starting from a GTRC in question (here '▶' and '◀' indicate the direction of combination):

(40) Partial CKY table:




$T/(T...\backslash A...\backslash B)_{[1,2]\blacktriangleright[3,5]}$ represents the derivation "$T/(T\backslash A)_{[1,1]\blacktriangleright[2,2]}$ $T/(T/C...\backslash B)_{[3,3]\blacktriangleleft[4,5]} \Longrightarrow T/(T...\backslash A...\backslash B)$" 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 (36). Since GTRC recovery process only concerns 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:

(41) GTRC recovery table:




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 $O\left(n^{10}\right) = O(\underset{i,j,k}{n^3} \times \underset{\text{recovery}}{n^2} \times \underset{\text{percolation}}{n^5})$.

## A.4 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. Discussion of spurious ambiguity is included in Subsection 6.2.2.

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`. 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 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 previous section.[17] 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 that 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. At the time of Komagata [1997a], morphological analysis takes about 0.2 seconds per word on average and needs improvement.[18] The output of a parse is an enumeration of the final result categories associated with the features and the semantics, as seen below (Sentence 7).

(42)
```
itumo 95mmHg o koeru baai_wa tiryou ga hituyoudesu.
always num(95mmHg) -ACC exceed in_case treatment [-NOM,-CONJv] necessary
  Cat:
    SS: s
    PA: (in_case always((exceed num(95mmHg) $1)) (necessary treatment))
  Cat:
    SS: s
    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 provides a result without genuine ambiguities.

Let us start with the analysis of the category-only case.[19] This case corresponds to the situation involving syntactic methods and also the polynomial algorithms introduced in the previous section. The results are shown in Figures A.2 (linear scale) and A.3 (log scale). Both exponential ($y =$

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

[18] 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].

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

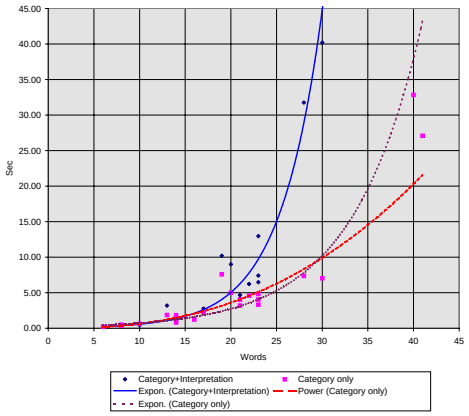



Figure A.2: Basic Data Set (linear scale)

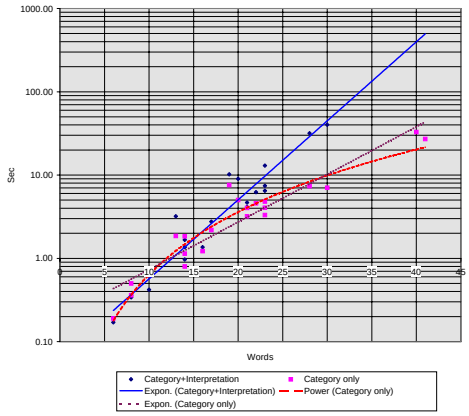



Figure A.3: Basic Data Set (log scale)

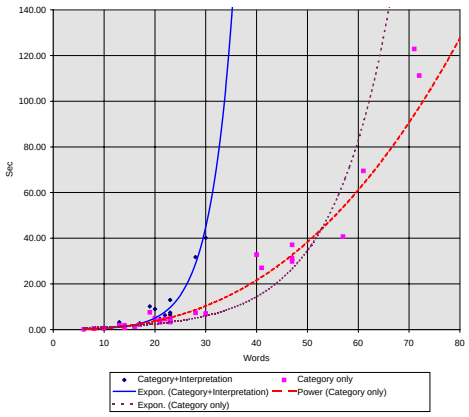



Figure A.4: Extended Data Set (linear scale)

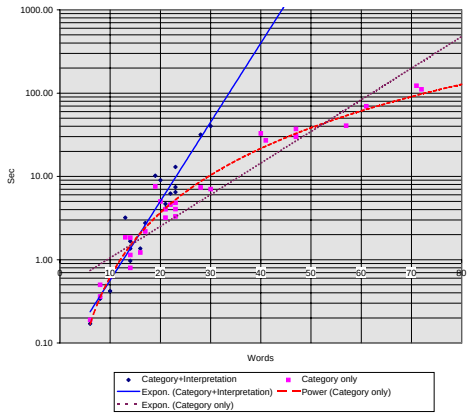



Figure A.5: Extended Data Set (log scale)

$0.1963 \times 1.14^n$) 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 A.4 and A.5. 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 stand. Tanaka and Ueki [1995] report that the LR-based syntactic analysis of a 19-word sentence in Japanese took 3.240sec.[20] 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.[21] The situation naturally looks quite different. The exponential regression line $t = 0.0638 \times 1.24^n$ (Figures A.2 and A.3) 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

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

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

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:

(43) *a*. Modification ambiguities:

n-TOP $\underbrace{\mathbf{adjn}_1\text{-GEN}\ \mathbf{n}_2\text{-NOM}}_{\overrightarrow{\ }\text{2-way ambiguous}}$ v-coord, $\mathbf{adv}_1$ $\underbrace{\mathbf{adv}_2\mathbf{v}_1\text{-COMP-TOO}}_{\overrightarrow{\ }\text{3-way ambiguous}}$ $\text{v}_2$

*b*. Coordination ambiguities:

n-TOP adj n-GEN n-NOM v-**coord**, adv adv v-COMP-TOO v

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

n-TOP adj n-GEN n-NOM v-**coord**, adv adv v-COMP-TOO v

Each case involves both adjective and adverb modification ambiguities ($2 \times 3$ 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.

## A.5 Conclusion

Through the investigation of CCG-GTRC in detail, a subclass of CCG-GTRC is shown to be equivalent to CCG-Std. This is done by way of simulating unbounded, but restricted 'permutations' of CCG-GTRC by lexical wrapping and argument-passing across categories. This contrasts with the formalisms involving 'doubly'-unbounded scrambling, which are strictly more powerful than CCG-Std. Thus, CCG-GTRC can be used in place of CCG-Std to account for non-traditional constituents including the ones shown in the introduction without proliferation of type-raised categories with the same computational properties.

The most restrictive condition for the choice of the studied subclass seems to be 'no outer sequence'. This is also associated with the limitation that the instances of GTRCs are finite. Naturally, we want $\mathsf{T}\big/\!\!\backslash\left(\mathsf{T}\big\backslash\!\!/ PP\right)/NP$ for English prepositions and $\mathsf{T}/(\mathsf{T}\backslash NP)\backslash NP$ for Japanese particles and to derive categories freely. Inclusion of outer sequence seems to increase the power since

that class cannot be simulated by CCG-Std due to the fact that CCG-Std cannot simulate certain 'island'-like behavior of GTRCs. But, in practice, CCG-Std calls for additional mechanism such as conditions on rule application for various linguistic reasons. These conditions cannot be in general expressed in CCG-Std proper either. We are thus at the borderline of LTAG-equivalence. To find out where exactly we are is another question we want to ask.

We have also shown that the extension of CCGs including GTRCs can be parsed polynomially in theory and in practice, with some qualifications and conditions. These polynomial results support the proposed grammar that can describe non-traditional 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.

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 eliminate a possibility that rarely occurs in practice. 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 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.

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].[22] 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

[22]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].

improve the performance to a really practical level.