IEICE TRANS. INF. & SYST., VOL.E94–D, NO.3 MARCH 2011

PAPER Special Section on Knowledge Discovery, Data Mining and Creativity Support System

Probabilistic Treatment for Syntactic Gaps in Analytic Language Parsing

Prachya BOONKWAN[†,††], *Nonmember and* **Thepchai SUPNITHI**[†a)], *Member*

SUMMARY This paper presents a syntax-based framework for gap resolution in analytic languages. CCG, reputable for dealing with deletion under coordination, is extended with a memory mechanism similar to the slot-and-filler mechanism, resulting in a wider coverage of syntactic gaps patterns. Though our grammar formalism is more expressive than the canonical CCG, its generative power is bounded by Partially Linear Indexed Grammar. Despite the spurious ambiguity originated from the memory mechanism, we also show that its probabilistic parsing is feasible by using the dual decomposition algorithm.

key words: *combinatory categorial grammar, mildly context-sensitive grammar, syntactic gap resolution, analytic language, probabilistic parsing, dual decomposition*

1. Introduction

Syntactic gap has been a nontrivial issue in natural language parsing. It manifests a challenge of dealing with nonstandard constituent coordination. Combinatory Categorial Grammar [1], [2], a near context-free grammar formalism, accounts for this problem as a major concern. Its combinatory operations resolve deletion under coordination, such as right-node raising (e.g. "Mary likes but John dislikes potatoes.") and gapping (e.g. "Mary bought potatoes, and John, tomatoes."). However, the issues of zero pronouns and gapping in coordination become more challenging when considering languages with a higher degree of analyticity.

In this paper, we explain how we can deal with zero pronouns and gapping in coordinate construction by incorporating a memory mechanism and how we restrict the generative power of the resulted hybrid. The integrated memory mechanism is motivated by anaphoric resolution mechanism in modern thoerem proving [3]–[8] and categorial grammar [9], [10]. These mechanisms are designed for associating fillers and gaps found in an input sentence.

Theoretically, we discuss how this hybrid efficiently helps us deal with zero pronouns and gapping and how far the generative power grows from CCG after incorporating the memory mechanism. We will also introduce a probabilistic parsing model which separates the memory mechanism from the syntactic derivation and explain how its inference can be done in polynomial time using dual decomposition [11].

The rest of the paper is organized as follows. We will explain CCG in Sect. 2 and then motivate the need of the memory mechanism in dealing with zero pronouns and gapping in Sect. 3. We will describe the hybrid model of CCG and the filler-gap memory in Sect. 4. In Sect. 5, we will elaborate a probabilistic parsing model for the hybrid and explain the inference. Finally, we will conclude the paper in Sect. 6.

2. CCG

CCG is a lexicalized grammar; i.e. a grammar is encoded in terms of lexicons assigned with one or more syntactic categories. Each syntactic category may be an atomic element or curried functions specifying linear directions in which they seek for their arguments. A word is assigned with a syntactic category by the turnstile operator $\vdash$. For example, a simplified English CCG is given in Eq. (1).

$$\begin{aligned} \text{John} &\vdash \texttt{np} \\ \text{sandwiches} &\vdash \texttt{np} \\ \text{eats} &\vdash \texttt{s\textbackslash np/np} \end{aligned} \tag{1}$$

The categories `X\Y` (and `X/Y`) denotes that `X` seeks for the argument `Y` from the left (right) side.

Combinatory rules are used to combine words into a phrase, forming a derivation of a sentence. For basic combination, forward (>) and backward (<) functional applications, defined in Eq. (2), are used.

$$\begin{aligned} \texttt{X/Y}\ \ \texttt{Y} &\Rightarrow \texttt{X} \quad [>] \\ \texttt{Y}\ \ \texttt{X\textbackslash Y} &\Rightarrow \texttt{X} \quad [<] \end{aligned} \tag{2}$$

We can derive the sentence "John eats sandwiches." by the rules and the grammar in Eq. (1) as illustrated in Fig. 1. For ease of understanding, a type of combination is also attached to each syntactic derivation.

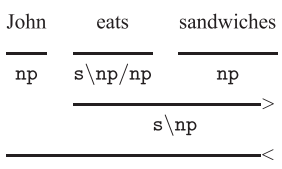

Fig. 1 A CCG derivation of "John eats sandwiches".

Manuscript received June 1, 2010.
Manuscript revised September 30, 2010.

[†]The authors are with Human Language Technology Laboratory, National Electronics and Computer Technology Center (NECTEC), 112 Thailand Science Park, Phaholyothin Road, Khlong 1, Pathumthani 12120, Thailand.
[††]The author is with School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK.
a) E-mail: thepchai.supnithi@nectec.or.th

DOI: 10.1587/transinf.E94.D.440

For coordination of two constituents, the coordination rules are used. There are two types of coordination rules regarding their directions: forward coordination (> &) and backward coordination (< &), defined in Eq. (3).

$$\begin{array}{rcll} \& & \mathtt{X} & \Rightarrow & \mathtt{X} \quad [>\&] \\ \mathtt{X} & [\mathtt{X}]_{\&} & \Rightarrow & \mathtt{X} \quad [<\&] \end{array} \tag{3}$$

Beyond functional application and coordination, CCG also makes use of rules motivated by combinators in combinatory logics: functional composition (**B**) and type raising (**T**). Classified by directions, the functional composition and type raising rules are defined in Eqs. (4) and (5), respectively.

$$\begin{array}{llll} \mathtt{X/Y} & \mathtt{Y/Z} & \Rightarrow & \mathtt{X/Z} \quad [>\mathbf{B}] \\ \mathtt{Y\backslash Z} & \mathtt{X\backslash Y} & \Rightarrow & \mathtt{X\backslash Z} \quad [<\mathbf{B}] \end{array} \tag{4}$$

$$\begin{array}{lll} \mathtt{X} & \Rightarrow & \mathtt{Y/(Y\backslash X)} \quad [>\mathbf{T}] \\ \mathtt{X} & \Rightarrow & \mathtt{Y\backslash(Y/X)} \quad [<\mathbf{T}] \end{array} \tag{5}$$

These rules permit associativity in derivation resulting in that coordination of non-standard constituents with similar types is possible.

CCG also allows functional composition with permutation called *disharmonic functional composition* to handle constituent movement such as heavy NP shift and dative shift in English. These rules are defined in Eq. (6).

$$\begin{array}{llll} \mathtt{X/Y} & \mathtt{Y\backslash Z} & \Rightarrow & \mathtt{X\backslash Z} \quad [>\mathbf{B}_{\times}] \\ \mathtt{Y/Z} & \mathtt{X\backslash Y} & \Rightarrow & \mathtt{X/Z} \quad [<\mathbf{B}_{\times}] \end{array} \tag{6}$$

To cope with gapping in coordination SVO&SO, the decomposition rule was proposed as a separate mechanism from CCG and is therefore not taken into account when considering CCG's generative power [2]. It decomposes a complete constituent into two parts for being coordinated with another incomplete constituent. The decomposition rule is defined in Eq. (7).

$$\mathtt{X} \quad \Rightarrow \quad \mathtt{Y} \quad \mathtt{X\backslash Y} \qquad [\mathbf{D}] \tag{7}$$

where `Y` and `X\Y` must be seen earlier in the derivation; that is, this rule yields a derivation only if it has access to the gap interpretation via discourse anaphor. The decomposition rule, together with syntax-external/anaphoric process, allows us to derive the sentence "John eats sandwiches, and Mary, noodles" as in Fig. 2. [2] stated that English is a forward gapping language because gapping always takes place at the right conjunct.

3. Zero Pronouns

CCG deals with deletion under coordination by the combinatory rules: functional composition, type raising, disharmonic functional composition, and decomposition. This enables CCG to handle a number of coordination patterns such as SVO&VO, SV&SVO, and SVO&SO.

However, CCG is challenged by some patterns of coordinate structures in analytic languages such as Chinese and Thai in which the use of zero pronouns in a coordinate structure is prevalent. A zero pronoun may appear in the positions of the subject and the object. This computationally complicates the parsing of coordinate structures containing zero pronouns as it possibly leads us to the use of zero cataphora in the object position. Let us consider the following Thai sentence.

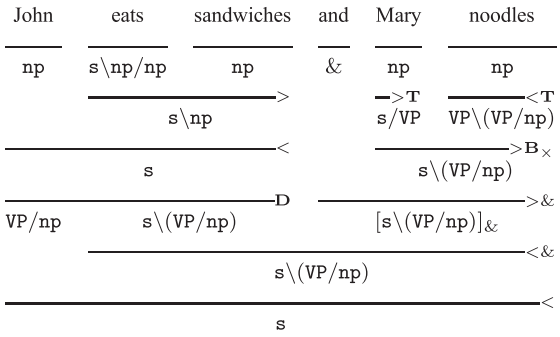

Fig. 2 A CCG derivation of "John eats sandwiches, and Mary, noodles". `VP` is short for `s\np`.

$$\begin{array}{lllllll} \mathrm{c^hāːvbâːn} & \mathrm{ʔɔ̌ːk} & \mathrm{tāːmhãː} & e_i & \mathrm{tɛː} \\ \text{villager} & \text{go-out} & \text{seek} & & \text{but} \\ \mathrm{mɛ̂ː} & \mathrm{p^hóp} & \mathrm{[lûːkc^hāːj]}_i & \mathrm{nɔ̄ːnlàp} \\ \text{mother} & \text{find} & \text{son} & & \text{sleep} \end{array} \tag{8}$$

'Villagers seek for a boy, but his mother finds him sleeping.'

The sentence in Eq. (8) exhibits the use of zero cataphora in the object position. Two conjuncts are juxtaposed with the connective tɛ̀ː 'but'. In the first conjunct, a zero pronoun is used to refer to the word lûːkc$^{\mathrm{h}}$āːj 'son' in the second conjunct. This example is a challenge for CCG because CCG fails to parse it as illustrated in Fig. 3.

The issue of zero pronouns in coordinate structures can be resolved by incorporating a memory mechanism that associates fillers to their gaps. We will show how the memory mechanism improves CCG's capability in dealing with of zero pronouns and gapping in the next section.

4. Memory Mechanism

The notion of memory mechanism in natural language parsing can be traced back to HOLD registers in ATN [12] in which fillers (antecedents) are held in registers for being filled to gaps found in the rest of the input sentence. These registers makes parsing equivalent to a Turing machine which recognizes the entire class of context-sensitive grammars and is too powerful for any natural languages.

This attempt was drawn into modern theorem proving. In Type Logical Grammar (TLG) [5]–[8], Gentzen's sequent calculus was incorporated with variable quantification to resolve pro-forms and VP ellipses to their antecedents. The variable quantification in TLG is comparable to the use of memory in storing antecedents and anaphora. In Categorial Type Logic (CTL) [3], [4], gap induction was incorporated. Syntactic categories were modified with modalities

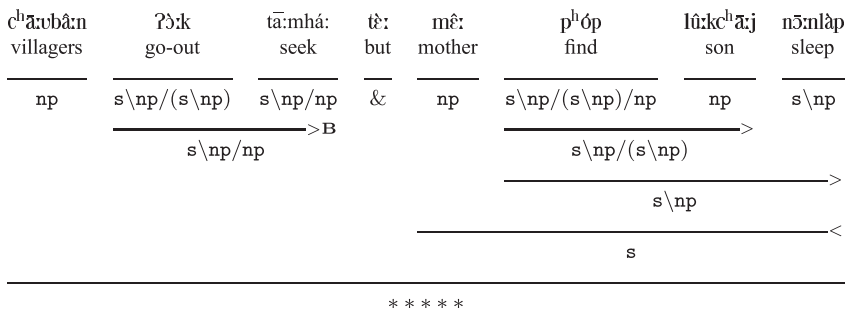

Fig. 3 An unsuccessful derivation of the sentence "Villagers seek for a boy, but his mother finds him sleeping".

which permit or prohibit gap induction in derivation. However, logical reasoning obtained from TLG and CTL is an NP-complete problem.

In CCG, [9] attempted to explicitly denote non-local anaphoric requirement whereby she introduced the anaphoric slash (|) and the anaphoric connective (**Z**) to connect anaphors to their antecedents. However, this framework does not support anaphora whose argument is not its antecedent, such as possessive adjectives. Recently, a filler-gap memory mechanism was again introduced to Categorial Grammar, called Memory-Inductive Categorial Grammar (MICG) [10]. Fillers and gaps, encoded as memory modalities, are attached to syntactic categories and they are associated by the gap-resolution connective when coordination and serialization take place. Though their framework is successful in resolving a wide variety of gapping, its generative power falls between LIG and Indexed Grammar, theoretically too powerful for natural languages.

4.1 Filler-Gap Association

The memory mechanism introduced in this paper deals with fillers and gaps in coordinate structures. It is similar to anaphoric resolution in ATN, Jacobson's model, TLG and CTL. However, it also has prominent distinction from them. The anaphoric mechanisms aforementioned are dealing with unbounded dependency or even inter-sentential ellipses, while the memory mechanism is dealing with only intra-sentential bounded dependency in coordinate structures. Moreover, choices of filler-gap association can be pruned out by the use of combinatory directionality because the word orders of analytic languages are fixed. It is noticeable that we can simply determine the grammatical function (subject or object) of arbitrary noun phrases from the directionality (e.g. the subject is always on the left and the object is always on the right of the verb). With these reasons, the notion of MICG's memory modalities and gap-resolution connective [10] are adapted for the backbone of the memory mechanism.

In CCG with memory mechanism extension (CCG+MM), syntactic categories are modalized with memory modalities. For each functional application, a syntactic category can be stored, or *memorized*, into the filler storage and the result category is modalized with the modality $\Box$ (read 'square'). A syntactic category can also be induced as a gap in a unary derivation called *induction* and the result category is modalized with the modality $\Diamond$ (read 'diamond').

There are two constraint parameters for each modality: a combinatory directionality $d \in \{<, >\}$ and the syntactic category of a filler/gap c, resulting in the filler and the gap denoted in the forms $\Box_c^d$ and $\Diamond_c^d$, respectively. For example, the syntactic category $\Box_{\mathtt{np}}^{<}\Diamond_{\mathtt{np}}^{>}\mathtt{s}$ has a filler of type $\mathtt{np}$ on the left side and a gap of type $\mathtt{np}$ on the right side.

The filler $\Box_c^d$ and the gap $\Diamond_c^d$ of the same directionalities d and syntactic categories c are said to be symmetric under the gap-resolution connective $\oplus$; i.e. they are matched and canceled in the gap resolution process. Unlike MICG, the associative power of $\oplus$ of CCG+MM is restricted to match only a filler and a gap, not between two gaps, so that the generative power can be preserved linear. Given two strings of modalities m_1 and m_2, the gap resolution connective $\oplus$ is recursively defined in Eq. (9).

$$\begin{array}{lrcl} \text{Recurrence:} & \Box_c^d m_1 \oplus \Diamond_c^d m_2 & = & m_1 \oplus m_2 \\ & \Diamond_c^d m_1 \oplus \Box_c^d m_2 & = & m_1 \oplus m_2 \\ \text{Base case:} & \epsilon \oplus \epsilon & = & \epsilon \end{array} \tag{9}$$

The notation ϵ denotes an empty string. It means that a syntactic category modalized with an empty modality string is simply *unmodalized*; that is, any modalized syntactic categories $\epsilon\mathtt{X}$ are equivalent to the unmodalized ones $\mathtt{X}$.

Since the syntactic categories are modalized by a modality string, all combinatory operations in the canonical CCG must preserve the modalities after each derivation step. However there are two conditions to be satisfied:

Condition 1: At least one operand of functional application must be unmodalized.

Condition 2: Both operands of functional composition and disharmonic functional composition, and the operand of type raising must be unmodalized.

From Condition 1, we modify CCG's functional application rules into Eq. (10).

$$\begin{array}{llcll} m\mathtt{X/Y} & \epsilon\mathtt{Y} & \Rightarrow & m\mathtt{X} & [>] \\ \epsilon\mathtt{X/Y} & m\mathtt{Y} & \Rightarrow & m\mathtt{X} & [>] \\ m\mathtt{Y} & \epsilon\mathtt{X\backslash Y} & \Rightarrow & m\mathtt{X} & [<] \\ \epsilon\mathtt{Y} & m\mathtt{X\backslash Y} & \Rightarrow & m\mathtt{X} & [<] \end{array} \tag{10}$$

These conditions are introduced to preserve the generative power of CCG.

4.2 Memorization

A filler modality is pushed to the top of the memory when a functional application rule is applied, where the filler's syntactic category must be unmodalized. Let m be a modality string, the memorization operation is defined in Eq. (11).

$$\begin{array}{llll} \epsilon\mathtt{X/Y} \quad m\mathtt{Y} & \Rightarrow & \square^{<}_{\mathtt{X/Y}} m\mathtt{X} & [>\mathbf{M}_F] \\ m\mathtt{X/Y} \quad \epsilon\mathtt{Y} & \Rightarrow & \square^{>}_{\mathtt{Y}} m\mathtt{X} & [>\mathbf{M}_A] \\ \epsilon\mathtt{Y} \quad m\mathtt{X\backslash Y} & \Rightarrow & \square^{<}_{\mathtt{Y}} m\mathtt{X} & [<\mathbf{M}_A] \\ m\mathtt{Y} \quad \epsilon\mathtt{X\backslash Y} & \Rightarrow & \square^{>}_{\mathtt{X\backslash Y}} m\mathtt{X} & [<\mathbf{M}_F] \end{array} \tag{11}$$

The operations $\mathbf{M}_F$ and $\mathbf{M}_A$ are read 'memorize the functor' and 'memorize the argument', respectively.

4.3 Induction

A gap modality is pushed to the top of the memory when a gap of such type is induced at either side of the syntactic category. Let m be a modality string, the induction operation is defined in Eq. (12).

$$\begin{array}{rcll} m\mathtt{X/Y} & \Rightarrow & \Diamond^{>}_{\mathtt{Y}} m\mathtt{X} & [>\mathbf{I}_A] \\ m\mathtt{Y} & \Rightarrow & \Diamond^{<}_{\mathtt{X/Y}} m\mathtt{X} & [>\mathbf{I}_F] \\ m\mathtt{X\backslash Y} & \Rightarrow & \Diamond^{<}_{\mathtt{Y}} m\mathtt{X} & [<\mathbf{I}_A] \\ m\mathtt{Y} & \Rightarrow & \Diamond^{>}_{\mathtt{X\backslash Y}} m\mathtt{X} & [<\mathbf{I}_F] \end{array} \tag{12}$$

The operations $\mathbf{I}_F$ and $\mathbf{I}_A$ are read 'induce the functor' and 'induce the argument', respectively.

4.4 Coordination

Because the use of memory mechanism elucidates fillers and gaps hidden in the derivation, we can extend the generative capacity of CCG with the gap resolution. Fillers and gaps are associated in the coordinate structures by the gap resolution connective $\oplus$. For any given modality strings m_1 and m_2 such that $m_1 \oplus m_2 = \epsilon$, we modify CCG's coordination rules into Eq. (13).

$$\begin{array}{llll} \& \quad m_2\mathtt{X} & \Rightarrow & m_2[\mathtt{X}]_{\&} & [>\&] \\ m_1\mathtt{X} \quad m_2[\mathtt{X}]_{\&} & \Rightarrow & \mathtt{X} & [<\&] \end{array} \tag{13}$$

With the memory mechanism and the coordination rules, several patterns of coordinate structures with zero pronouns and gapping can be simply resolved.

Zero pronoun: The memory mechanism is especially designed for dealing with zero pronouns in coordinate structures. Let us consider the use of zero cataphora in the object position aforementioned in Eq. (8). The syntactic derivation of this can be illustrated in Fig. 4. The zero cataphora in the first conjunct can be seen as a gap and it corresponds to the antecendent lûːkcʰāːj 'son' in the second conjunct. Since the gap has the syntactic category $\mathtt{np}$, we can simply induce the gap $\Diamond^{>}_{\mathtt{np}}$ from the zero cataphora and attach it to the first conjunct. In the same time, we memorize its antecedent as the filler $\square^{>}_{\mathtt{np}}$ when combining the verb with it and attach this filler to the second conjunct. We then use the coordination rules to juxtapose both conjuncts and associate the filler and the gap. This results in a complete syntactic derivation.

Gapping: We can solve gapping in a coordinate structure by memorizing the verb in the first conjunct as a filler and inducing a gap in the second conjunct. The filler and the gap will be associated by the coordination rules in Eq. (13). For example, let us consider the CCG+MM derivation of the sentence "John eats sandwiches, and Mary, noodles" in Fig. 5. We memorize the transitive verb "eat" in the first conjunct as $\square^{<}_{\mathtt{s\backslash np/np}}$ and induce a gap of the verb in the second conjunct as $\Diamond^{<}_{\mathtt{s\backslash np/np}}$. Since the filler and the gap share

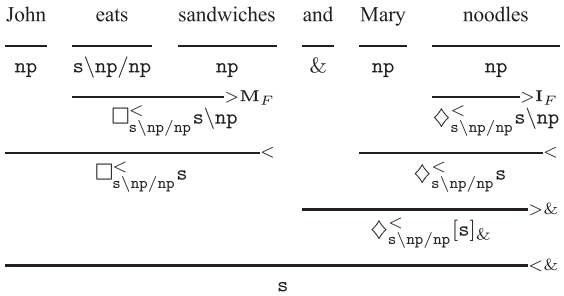

Fig. 5 A CCG+MM derivation of "John eats sandwiches, and Mary, noodles".

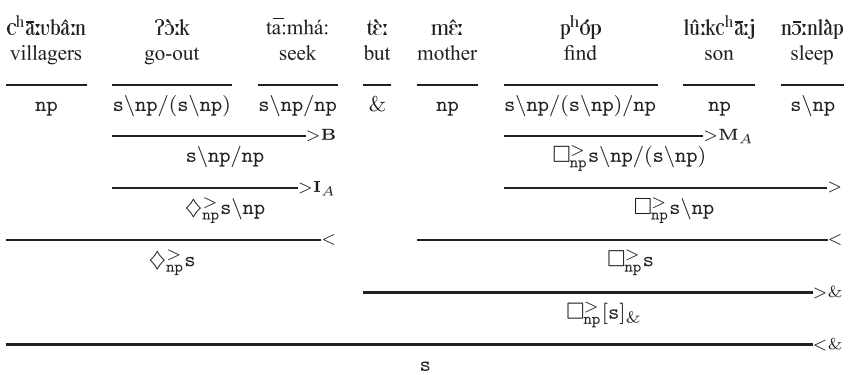

Fig. 4 A CCG+MM derivation of the sentence "Villagers seek for a boy, but his mother finds him sleeping".

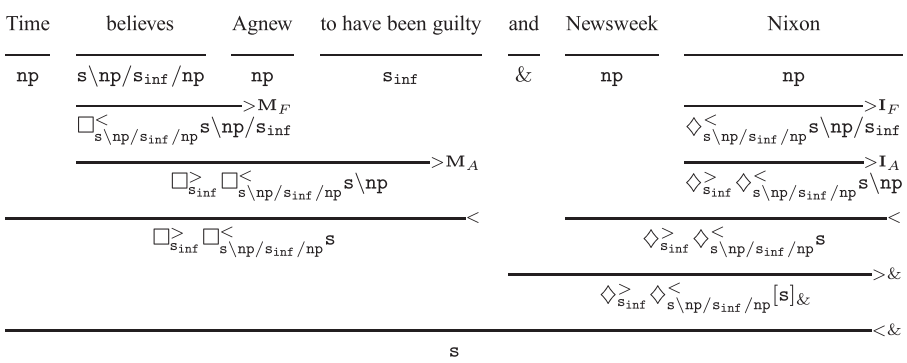

Fig. 6 A CCG+MM derivation of "Time believes Agnew to have been guilty, and Newsweek, Nixon".

the directionality < and the syntactic category s\np/np, we can associate them with the coordination rules.

The memory mechanism can also resolve more complex phenomena, including discontinuous gapping. For example, let us consider the CCG+MM derivation of the sentence "Time believes Agnew to have been guilty, and Newsweek, Nixon" in Fig. 6. In this case we memorize the verb "believe" and the infinitive phrase "to have been guilty" in the first conjunct as fillers and induce gaps for them in the second conjunct. We then resolve these fillers and gaps with the coordination rules.

4.5 Generative Power

When incorporating with the memory mechanism, CCG becomes flexible enough to handle zero pronouns and gapping in coordinate structures. Boonkwan [13] showed that the integration of memory mechanism into CCG extends the generative power to Partially Linear Indexed Grammar [14]. An interesting trait of PLIG is that it can generate the language $\{w^k | w$ is in a regular language and $k \in \mathcal{N}\}$. This is similar to the pattern of coordinate structures which perhaps contain zero pronouns and gapping. From [13]–[15], we can position CCG+MM in Chomsky's hierarchy as follows. Regular Language < CFG < CCG = TAG = HG = LIG < CCG+MM ≤ PLIG < LCFRS < CSG < Recursively Enumerable Language (recognizable by Turing Machine).

5. Probabilistic Parsing

5.1 Parse Tree with Memory Trace

Probabilistic parsing of CCG+MM is a challenging task regarding the computational complexity of the inference algorithm. With the memory mechanism we can push fillers and gaps into the memory (stack) and resolve them by the pop operation. However PCFG, widely used in the NLP community, becomes impractical if we directly adapt it to CCG+MM, because the stack can be unbounded resulting in an infinite search space of grammar rules. We therefore need to devise a new notion of parse tree with the memory capability and a finite search space.

Our parse tree is defined as follows. Let N be a set of nonterminals (i.e. syntactic categories) and V be a set of terminals (i.e. words). To limit the size of the search space, we annotate into each node a memory operation instead of a stack content. Let D be the set of 17 derivational types containing those of CCG $\{<, >, < \mathbf{B}, > \mathbf{B}, < \mathbf{B}_\times, > \mathbf{B}_\times, < \mathbf{T}, > \mathbf{T}\}$, those of the memory extension $\{< \mathbf{M}_F, < \mathbf{M}_A, > \mathbf{M}_F, > \mathbf{M}_A, < \mathbf{I}_F, < \mathbf{I}_A, > \mathbf{I}_F, > \mathbf{I}_A\}$, and the lexical derivation $\{\perp\}$. Each node label is therefore a member of the set $(N \times D) \cup V$. As exemplified in Fig. 7, this parse tree corresponds to the CCG+MM derivation in Fig. 5. This definition results in $O(|N|^3|D|^3)$ possible rule productions in the search space.

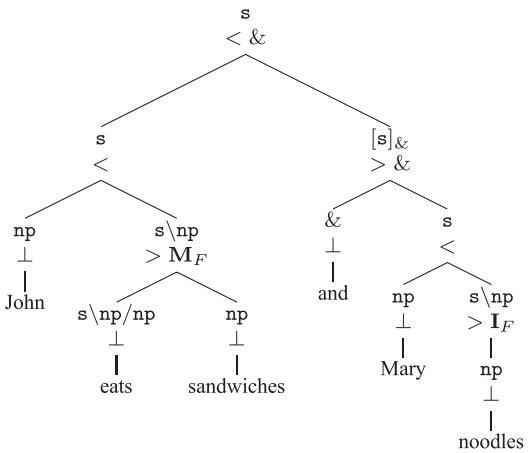

Fig. 7 An example parse tree with memory operations.

This kind of parse tree however seems problematic for parsing inference to a CKY-based algorithm, such as Inside-Outside algorithm. In such algorithm, the chart for a sentence of length n composes of a set of rule productions in the form:

$$\langle c : d \rightarrow c_1 c_2 : d_1 d_2, i, k, j \rangle \tag{14}$$

where $c, c_1, c_2 \in N$, $d, d_1, d_2 \in D$, and $1 \leq i \leq k < j \leq n$. The derivational type d is the combinatory operation that is used for combining c_1 and c_2 to obtain c, and c_1 and c_2 are produced by the combinatory operations d_1 and d_2, respectively. There are thus $O(|N|^3|D|^3n^3)$ such rule productions in the chart. Computing this large volume of productions requires extensive time and space, while training requires as

much data as a multiple of $O(|N|^3|D|^3)$ to efficiently cope with the data sparsity issue. This limitation has hindered many complex statistical models from practical use.

On the one hand, we can approximate the inference of CCG+MM using the dual decomposition algorithm [16]. We simplify the parsing problem by decomposing its model into two less complex interweaving subproblems: CCG derivation model and memory operation model, and then separately train these models with the same data set. The inference of CCG+MM can be done by intersection of the inference of each model; that is, we are finding the most likely parse tree such that both models agree with each other. By doing so, we can reduce the complexity of the inference from $O(|N|^3|D|^3)$ to $O(|N|^3\,|D| + |D|^3)$.

5.2 Split Models

We have learned that the canonical PCFG is too complex for CCG+MM's parse trees. One way to tackle this limitation is to decompose the parsing model into several subproblems and solve them with the dual decomposition algorithm [16].

We develop this idea by perceiving the parse trees in two different perspectives. In the first perspective, we perceive a CCG+MM's parse tree as an ordinary CCG tree. We then obtain that we choose to generate a new syntactic category and the derivational type used for combination from two syntactic categories. In the second perspective, we perceive it as a series of memory operations used to build the parse tree. We then obtain that we choose to generate a new derivational type from two syntactic categories and the derivational types used to generate them. By projecting the trees from both perspectives onto each other, they agree upon the derivational type of each node. This results in the agreement of derivational types on each span of corresponding trees.

Let us formalize our models as follows. We will follow [11]'s modeling method in defining the parsing models.

Model 1: CCG Derivation Model

Model 1 is a generative model for CCG derivation. For each step it predicts a syntactic category and a derivational type from daughter syntactic categories. Hierarchically structured as a context-free grammar, it contains all rules of the forms $c : d \to c_1c_2$ and $c : d \to c_1$, where $c, c_1, \ldots, c_m \in N$ are syntactic categories and $d \in D$ is a derivational type.

Now let us define the notion of parse tree in Model 1. Suppose we have a sentence of n words, $w_1 \ldots w_n$, a parse tree is a set of rule productions of the forms $\langle c : d \to c_1c_2, i, k, j\rangle$ and $\langle c : d \to c_1, i, k\rangle$, where $1 \le i \le k < j \le n$, resulting in $O(|N|^3\,|D|n^3)$ such rule productions. Let us define the search space of parsing, or *index set*, $\mathcal{I}_1$ such that it contains all possible rule productions at any spans (i, j) of the sentence. Each parse tree $x = \{x_r\}$ is a binary vector in $\{0, 1\}^{|\mathcal{I}_1|}$, where all x_r, the value of a rule production r used in the parse tree, are set to 1 while the rest are set to 0. Let $\mathcal{X} \subseteq \{0, 1\}^{|\mathcal{I}_1|}$ denote the set of all valid parse trees.

For the statistical part of Model 1, we define a weight vector $\theta^{\text{ccg}} = \{\theta^{\text{ccg}}(r)|r \in \mathcal{I}_1\}$ that specifies a real-valued weight for each rule production. For a given sentence $w_1 \ldots w_n$, we are finding the optimal parse tree x^*, such that

$$x^* = \arg\max_{x \in \mathcal{X}} x \cdot \theta^{\text{ccg}} \tag{15}$$

where $x \cdot \theta^{\text{ccg}}$ is the inner product between x and θ^{ccg}. We will use the notation $\theta^{\text{ccg}}(r)$ to represent the weight of the rule production r. Under PCFG, we define $\theta^{\text{ccg}}(r)$ as follows.

$$\begin{aligned}
&\theta^{\text{ccg}}(c : d \to c_1c_2, i, k, j) \\
&\quad = \log P(c : d \to c_1c_2) \\
&\qquad +\delta_i^k \log P(c_1 :\perp\!\!\to w_i) \\
&\qquad +\delta_{k+1}^j \log P(c_2 :\perp\!\!\to w_j) \\
&\theta^{\text{ccg}}(c : d \to c_1, i, j) \\
&\quad = \log P(c : d \to c_1) \\
&\qquad +\delta_i^j \log P(c_1 :\perp\!\!\to w_i)
\end{aligned} \tag{16}$$

where $P(r)$ is the probability (parameter) of the rule r and δ_i^j is the Kroneker's notation defined as $\delta_i^j = 1$ if $i = j$, 0 otherwise.

Model 2: Memory Operation Model

Model 2 is a generative model for memory operations. For each step it predicts a derivational type from previous derivational types. It contains all rules of the forms $d \to d_1d_2$ and $d \to d_1$, where $d, d_1, d_2 \in D$ are derivational types.

We define the notion of derivation tree in Model 2 as follows. Suppose we have a sentence of n words, $w_1 \ldots w_n$, a derivation tree is a set of rule productions of the forms $\langle d \to d_1d_2, i, k, j\rangle$ and $\langle d \to d_1, i, k\rangle$, where $1 \le i \le k < j \le n$, resulting in $O(|D|^3n^3)$ such rule productions. Let us define the search space of parsing $\mathcal{I}_2$ such that it contains all possible rule productions at any spans (i, j) of the sentence. Each derivation tree $y = \{y_r\}$ is a binary vector in $\{0, 1\}^{|\mathcal{I}_2|}$, where all y_r, the value of a rule production r used in the derivation tree, are set to 1 while the rest are set to 0. Let $\mathcal{Y} = \{0, 1\}^{|\mathcal{I}_2|}$ denote the set of all valid derivation trees.

We also define a weight vector $\theta^{\text{mem}} = \{\theta^{\text{mem}}(r)|r \in \mathcal{I}_2\}$ that specifies a real-valued weight for each rule production. For a given sentence $w_1 \ldots w_n$, we are finding the optimal derivation tree y^*, such that

$$y^* = \arg\max_{y \in \mathcal{Y}} y \cdot \theta^{\text{mem}} \tag{17}$$

where $y \cdot \theta^{\text{mem}}$ is the inner product between y and θ^{mem}. We will use the notation $\theta^{\text{mem}}(r)$ to represent the weight of the rule production r. Under PCFG, we define θ^{mem} as follows.

$$\begin{aligned}
\theta^{\text{mem}}(d \to d_1d_2, i, k, j) &= \log P(d \to d_1d_2) \\
\theta^{\text{mem}}(d \to d_1, i, j) &= \log P(d \to d_1)
\end{aligned} \tag{18}$$

where $P(r)$ is the probability of the rule r.

5.3 Inference with Dual Decomposition

We are now going to incorporate Models 1 and 2 using the dual decomposition algorithm. We unify the search spaces of the two models and mutually optimize each model iteratively until they all converge. We extend the search spaces $\mathcal{I}_1$ and $\mathcal{I}_2$ with the extension set $\mathcal{I}_{\text{uni}}$, where

$$\mathcal{I}_{\text{uni}} = \{(d, i, j) | 1 \leq i \leq j \leq n\} \tag{19}$$

Each (d, i, j) represents a span (i, j) having a derivational type $d \in D$. We choose (d, i, j) because it is a common part for both models. By extending the search spaces, each CCG derivation and memory operation trees will have additional components, called *marginal polytopes*, $x(d, i, j)$ and $y(d, i, j)$ conveying the same semantics as (d, i, j) on each type of parse trees, respectively.

We then formalize the parsing as a linear programming problem in which Models 1 and 2 are mutually used. We first define the search space of the integrated model $\mathcal{Q}$ as follows.

$$\mathcal{Q} = \{(x, y) | x \in \mathcal{X}, y \in \mathcal{Y}, \text{ and } x(d, i, j) = y(d, i, j)\} \tag{20}$$

That is, every $(x, y) \in \mathcal{Q}$ agrees on their memory operations on each span. We then have that our parsing problem is to solve the following objective function:

$$(x^*, y^*) = \arg \max_{(x,y) \in \mathcal{Q}} (x \cdot \theta^{\text{ccg}} + y \cdot \theta^{\text{mem}}) \tag{21}$$

As a linear programming problem, we also define the linear constraints for the marginal polytopes in Eq. (22).

$$\begin{aligned} x(d, i, j) = &\sum_{k=i}^{j-1} \sum_{c, c_1, c_2} x(c : d \to c_1 c_2, i, k, j) \\ &+ \sum_{c, c_1} x(c : d \to c_1, i, j) \end{aligned} \tag{22}$$

$$\begin{aligned} y(d, i, j) = &\sum_{k=i}^{j-1} \sum_{c, d_1, d_2} y(c : d \to d_1 d_2, i, k, j) \\ &+ \sum_{c, d_1} y(c : d \to d_1, i, j) \end{aligned}$$

The objective function in Eq. (21) can be optimized by the dual decomposition algorithm illustrated in Fig. 8. In this approach, we iteratively solve the two parsing subproblems separately. We solve the parsing problems for CCG derivation model (line 5) and memory operation model (line 6) by using a simple dynamic-programming-based decoding algorithm such as Inside-Outside algorithm. At the end of each iteration (lines 10–12), we update the Lagrange multipliers $u^{(k)}(d, i, j)$ with the difference between the corresponding marginal polytopes of the two models. The more diverging the models, the more increased the Lagrange multipliers. These multipliers are then used for endorsing the two models to converge in the next iteration. The algorithm iterates until all the marginal polytopes converge or it reaches the maximum number of iterations. In case that the algorithm terminates before the models converge, $x^{(K)}$ and $y^{(K)}$ are heuristically chosen as the solution, as suggested by [11], [17]. The validity and convergence of the algorithm were guaranteed in [11].

1: **for all** $(d, i, j) \in \mathcal{I}_{\text{uni}}$ **do**
2: **let** Lagrange multiplier $u^{(1)}(d, i, j) \leftarrow 0$
3: **end for**
4: **for** $k = 1$ **to** K **do**
5: **let** $x^{(k)} \leftarrow \arg\max_{x \in \mathcal{X}} (x \cdot \theta^{\text{ccg}} - \sum_{(d,i,j) \in \mathcal{I}_{\text{uni}}} u^{(k)}(d, i, j) x(d, i, j))$
6: **let** $y^{(k)} \leftarrow \arg\max_{y \in \mathcal{Y}} (y \cdot \theta^{\text{mem}} + \sum_{(d,i,j) \in \mathcal{I}_{\text{uni}}} u^{(k)}(d, i, j) y(d, i, j))$
7: **if** $x^{(k)}(d, i, j) = y^{(k)}(d, i, j)$ for all $(d, i, j) \in \mathcal{I}_{\text{uni}}$ **then**
8: **return** $(x^{(k)}, y^{(k)})$
9: **end if**
10: **for all** $(d, i, j) \in \mathcal{I}_{\text{uni}}$ **do**
11: **let** $u^{(k+1)}(d, i, j) \leftarrow u^{(k)}(d, i, j) + \alpha_k (x^{(k)}(d, i, j) - y^{(k)}(d, i, j))$
12: **end for**
13: **end for**
14: **return** $(x^{(K)}, y^{(K)})$

Fig. 8 The decoding algorithm for solving the parsing problem for CCG+MM. K is the maximum number of iterations, and each parameter α_k specifies step sizes for each iteration.

By splitting the parsing model into two polynomial subproblems, we can parse CCG+MM in a polynomial time complexity. This leads us to the possibility of statistically solving the syntactic gaps in analytic languages.

6. Conclusion

We have presented a syntax-based framework called CCG+MM for probabilistically solving deletion under coordination in analytic languages. We extended Combinatory Categorial Grammar with a memory mechanism similar to the slot-and-filler concept. In this framework, gaps and their antecedents are identified and associated in parallel with syntactic derivation. Its generative power was proven to be bounded by Partially Linear Indexed Grammar. CCG+MM is also shown practical for probabilistic parsing because it can be parsed in polynomial time using the dual decomposition algorithm.

Acknowledgement

The authors are grateful for the help of Prof. Mark Steedman, University of Edinburgh, for invaluable discussion about Combinatory Categorial Grammar and his guide for the proof of the generative power. We appreciate the help of Prof. Michael Collins, MIT, for a discussion about his application of the dual decomposition algorithm in dependency parsing which motivated the parsing method proposed in this paper. We also would like to thank the anonymous reviewers for their comments and suggestions.

References

[1] M. Steedman, The Syntactic Process, The MIT Press, Cambridge, Massachusetts, 2000.
[2] M. Steedman, "Gapping as constituent coordination," Linguistics and Philosophy, vol.13, pp.207–263, 1990.

[3] P. Hendriks, "Ellipsis and multimodal categorial type logic," Proc. Formal Grammar Conference, pp.107–122, Barcelona, Spain, 1995.
[4] M. Moortgat, "Categorial type logics," in Handbook of Logic and Language, ed. van Benthem and ter Meulen, ch. 2, pp.163–170, Elsevier/MIT Press, 1997.
[5] G. Morrill, "Type logical grammar," in Categorial Logic of Signs, Kluwer, Dordrecht, 1994.
[6] G. Jäger, "Anaphora and ellipsis in type-logical grammar," Proc. 11th Amsterdam Colloquium, pp.175–180, ILLC, Universiteit van Amsterdam, Amsterdam, the Netherland, 1997.
[7] G. Jäger, "Anaphora and quantification in categorial grammar," Lect. Notes Comput. Sci.; Selected papers from the 3rd International Conference, on Logical Aspects of Computational Linguistics, pp.70–89, 2001.
[8] R.T. Oehrle, Non-Transformational Syntax: A Guide to Current Models, ch. Multi-modal Type Logical Grammar, Oxford: Blackwell, 2007.
[9] P. Jacobson, "Towards a variable-free semantics," Linguistics and Philosophy, vol.22, pp.117–184, Oct. 1999.
[10] P. Boonkwan and T. Supnithi, "Memory-inductive categorial grammar: An approach to gap resolution in analytic-language translation," Proc. 3rd International Joint Conference on Natural Language Processing, pp.80–87, Hyderabad, India, Jan. 2008.
[11] A.M. Rush, D. Sontag, M. Collins, and T. Jaakkola, "On dual decomposition and linear programming relaxations for natural language processing," Proc. EMNLP 2010, 2010.
[12] W.A. Woods, "Transition network grammars for natural language analysis," Commun. ACM, vol.13, no.10, pp.591–606, Oct. 1970.
[13] P. Boonkwan, "A memory-based approach to the treatment of serial verb construction in combinatory categorial grammar," EACL '09: Proc. 12th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop, pp.10–18, Association for Computational Linguistics, Morristown, NJ, USA, 2009.
[14] B. Keller and D. Weir, "A tractable extension of linear indexed grammars," Proc. 7th European Chapter of ACL Conference, 1995.
[15] K. Vijay-Shanker and D.J. Weir, "The equivalence of four extensions of context-free grammars," Mathematical Systems Theory, vol.27, no.6, pp.511–546, 1994.
[16] G.B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," Oper. Res., vol.8, pp.101–111, 1960.
[17] N. Komodakis, N. Paragios, and G. Tziritas, "MRF optimization via dual decomposition: Message-passing revisited," International Conference on Computer Vision, 2007.

Prachya Boonkwan was born in August 1981. He obtained his bachelor degree (with honors) and master degree in Computer Engineering from Kasetsart University, Thailand. He is working as a research fellow at Human Language Technology Lab at NECTEC in Thailand. He is now on leave to pursue his Ph.D. in Informatics at University of Edinburgh in United Kingdom since September 2008. His interest includes grammar induction, formal syntax, and probabilistic parsing.

Thepchai Supnithi received the B.S. degree in Mathematics from Chulalongkorn University in 1992. He received the M.S. and Ph.D. degrees in Engineering from the Osaka University in 1997 and 2001, respectively. Since 2001, he has been with the Human Language Technology Lab at NECTEC in Thailand.