Proceedings of HLT-NAACL 2003
Main Papers , pp. 79-86
Edmonton, May-June 2003

Multitext Grammars and Synchronous Parsers

I. Dan Melamed
Computer Science Department
New York University
{lastname}@cs.nyu.edu

Abstract

Multitext Grammars (MTGs) generate arbitrarily many parallel texts via production rules of arbitrary length. Both ordinary MTGs and their bilexical subclass admit relatively efficient parsers. Yet, MTGs are more expressive than other synchronous formalisms for which parsers have been described in the literature. The combination of greater expressive power and relatively low cost of inference makes MTGs an attractive foundation for practical models of translational equivalence.

1 Introduction

Synchronous grammars are a perspicuous way to describe the multi-dimensional structures that are hidden in parallel texts. As with ordinary grammars, synchronous grammars can be used to guide a parser to infer these structures from data in which they are not directly observable. Such an inference process can be used to collect empirical knowledge about translational equivalence for machine translation and other applications. Unfortunately, the synchronous grammars for which parsers have been described in the literature have insufficient expressive power to be practically useful.

At a minimum, a synchronous grammar that could serve as a foundation for practical translation modeling must be able to express (1) mutual translations that have different lengths, (2) crossing correspondences, (3) bilexical dependencies, and (4) discontinuous constituents, which are surprisingly important for bilexical parsing of parallel texts. At the same time, the grammar's recognition complexity should be lower than that of existing formalisms. Synchronous lexicalized TAGs (Abeillé *et al.*, 1990; Shieber & Schabes, 1990) certainly have the desired expressive power, but the computational complexity of their recognition is prohibitive. Multiple CFGs (Seki *et al.*, 1991) are less complex, but they cannot express (3), and their procedural parsers are difficult to analyze and optimize. Wu (1997) reported that a parser for Inversion Transduction Grammar (ITG) can be reasonably fast, but ITGs cannot express (3) or (4). The head transducers of Alshawi *et al.* (2000) are perhaps the best contenders in the literature, but they cannot express (4) either.

We introduce the class of Multitext Grammars (MTGs) to satisfy the above criteria for expressiveness and tractability. In this paper we focus on MTG parsers and their computational complexity, so we describe MTGs informally in Section 2, omitting all proofs. Section 3 gives a gentle introduction to synchronous parsing for a subclass of MTGs. We describe the parsers in a declarative form, which is amenable to complexity analysis by inspection and to a variety of parsing strategies (Shieber *et al.*, 1995). This approach facilitated the generalization of some recent advances in monolingual parsing, resulting in efficient parsers for bilexical MTGs. Section 4 explains the main source of MTG complexity, and shows why discontinuous constituents are practically unavoidable, even for parsing relatively simple parallel texts from languages with similar word order. Section 5 presents a parser for arbitrary MTGs and its optimization for bilexical MTGs.

2 Multitext Grammars

Texts that are translations of each other are called **parallel texts** or **multitexts**. Each multitext consists of **component texts** or **components**. Every MTG is a D-MTG for some constant D, and it generates multitexts with D components. For example, 2-MTGs generate bitexts.

An MTG has disjoint sets of terminals T and nonterminals N. We often group terminals or nonterminals into vectors that we call **links**. We shall write vector variables **in bold**. Links express the

translational equivalence between their components. Every link generated by a D-MTG has D components. Some (but not all) components of a link may be empty. An empty component indicates that an expression vanishes in translation. To express empty components, we add a special terminal ϵ to T and a special nonterminal ε to N. In MTG applications, the different components of a link will typically come from largely disjoint subsets of T or N, representing vocabularies or sets of grammatical categories from different languages.

Each MTG also has a set of production rules (or just "productions" for short), which fall into one of two categories.[1] YIELD productions have the form

$$\mathbf{X} \Rightarrow \mathbf{t} \tag{1}$$

where $\mathbf{X}$ is a link of D nonterminals and $\mathbf{t}$ is a link of D terminals. t_d is empty if and only if X_d is empty, $1 \leq d \leq D$. DEPEND productions have the form

$$\mathbf{X} \Rightarrow \bowtie \mathbf{PM} \tag{2}$$

where $\mathbf{M}$ is a non-empty vector of nonterminal links, $\mathbf{P}$ is a non-empty vector of D permutations, and $\bowtie$ ("join") is a rendering function, explained below.[2] The **rank** of an MTG production is the number of nonterminal links on its RHS. The **rank** of an MTG is the maximum rank of its production rules. MTG(R) is the class of MTGs of rank R.

Each row of $\mathbf{P}$ and $\mathbf{M}$ corresponds to a different component of multitext. Each permutation is written as a row in $\mathbf{P}$, and each link is written as a column in $\mathbf{M}$, as in Equation 3 below. If X_d is empty, then the dth component of every link in $\mathbf{M}$ must be empty too. If X_d is not empty, then at least one of the links in $\mathbf{M}$ must have a non-empty dth component. The position of a non-empty terminal or nonterminal relative to other non-empty elements of its component is its **role**. If there are m non-empty nonterminals in component (row) d of $\mathbf{M}$ then P_d is a permutation of roles from 1 to m. P_d is empty if and only if X_d is empty.

The D-MTG derivation process begins with the start link $\mathbf{\$}$, which is a vector of D copies of the special start symbol $\$ \in N$. The derivation continues with nondeterministic application of production rules. The semantics of $\Rightarrow$ are the usual semantics of rewriting systems, i.e., that the expression on the LHS can be rewritten as the expression on the RHS. Following convention, we let $\stackrel{*}{\Rightarrow}$ be the reflexive and transitive closure of $\Rightarrow$.

[1] This dichotomy imposes a convenient normal form, without loss of generality.

[2] The rendering function is a notational convenience; MTGs can be defined without it.

When no more productions can be applied, i.e., when all nonterminals have been rewritten into terminals, the rendering functions are evaluated in inside-out order. The $\bowtie$ function rearranges the nonempty terminals in each row of a link vector according to that row's permutation. For example,

$$\begin{matrix}[1,2,3]\\ \bowtie\,[1,3,2,4]\\ [3,2,1]\end{matrix}\begin{pmatrix}\epsilon & \text{a} & \text{b} & \text{c} & \epsilon\\ \text{w} & \text{y} & \text{x} & \epsilon & \text{z}\\ \text{t} & \epsilon & \text{u} & \text{v} & \epsilon\end{pmatrix} = \begin{matrix}\text{abc}\\ \text{wxyz}\\ \text{vut}\end{matrix} \tag{3}$$

By reordering the terminals independently in each component, the join operator hides information about which terminals were derived from the same link. Thus, the translational equivalence represented by links is not observable in MTG yields, just as it is not observable in raw multitext.

To avoid spurious ambiguity, we stipulate a normal form for components of $\mathbf{P}$: In each permutation, the first appearance of role x must precede the first appearance of role y for all $x < y$, except where the arrangement is incompatible with a preceding permutation in $\mathbf{P}$. We could, for example, obtain the same result above if we put $\epsilon z \epsilon$ first, put $\epsilon w t$ last, and switch their roles in the 2nd and 3rd permutations. However, the normal form requires the 2nd permutation to be [1, 3, 2, 4], not [4, 3, 2, 1], so $\epsilon z \epsilon$ must be listed last.

Let $\mathbf{Q}$ be an MTG derivation where no more production rules can be applied. Let $\mathsf{Render}(\mathbf{Q})$ be the result of evaluating all the $\bowtie$'s in $\mathbf{Q}$. The (formal) language $L(G)$ of an MTG G is the set of multitexts that can be generated by applying $\stackrel{*}{\Rightarrow}$ to the start link of G and then evaluating all the joins. I.e., $L(G) = \{\mathsf{Render}(\mathbf{Q}) : \mathbf{\$} \stackrel{*}{\Rightarrow} \mathbf{Q}\}$.

Due to the importance of lexical information in disambiguating syntactic structure, we shall pay special attention to lexicalized MTGs (LMTGs) of the bilexical variety (L_2MTGs). A bilexical MTG has a set Λ of "delexicalized" nonterminal labels. Intuitively, Λ corresponds to the nonterminal set of an ordinary CFG. Then, every nonterminal in N has the form $L[t]$ for some terminal $t \in T$ and some label $L \in \Lambda$.[3] The terminal t is the **lexical head** of its constituent, or just the **head**. One link on the RHS of each L_2MTG production serves as the **heir** of the link on the LHS. Each component of the heir link inherits the lexical head of its parent nonterminal. An example of a 2-L_2MTG derivation is in Figure 1.

[3] The nonterminal ε is always lexicalized with the terminal ϵ. Other nonterminals may also be lexicalized with ϵ to represent empty categories. The special start nonterminal \$ is lexicalized with the special start terminal \$. Following Eisner & Satta (1999), we can then define G so that the language of interest is actually $L'(G) = \{\mathbf{Q}' : \mathbf{Q}'\mathbf{\$} \in L(G)\}$.

$$\begin{matrix}\$[\$]\\ \$[\$]\end{matrix} \Rightarrow \bowtie_{\substack{[1,2]\\[1,2]}} \begin{pmatrix} S[\text{fed}] & \$[\$] \\ S[\text{kormil}] & \$[\$]\end{pmatrix} \Rightarrow \bowtie_{\substack{[1,2]\\[1,2]}} \left(\bowtie_{\substack{[1,2,3]\\[1,3,2]}} \begin{pmatrix} Pro[\text{I}] & V[\text{fed}] & NP[\text{cat}] \\ Pro[\text{ya}] & V[\text{kormil}] & NP[\text{kota}]\end{pmatrix} \begin{matrix}\$[\$]\\ \$[\$]\end{matrix}\right) \tag{4}$$

$$\Rightarrow \bowtie_{\substack{[1,2]\\[1,2]}} \left(\bowtie_{\substack{[1,2,3]\\[1,3,2]}} \begin{pmatrix} Pro[\text{I}] & V[\text{fed}] \\ Pro[\text{ya}] & V[\text{kormil}]\end{pmatrix} \bowtie_{\substack{[1,2]\\[1]}} \begin{pmatrix} D[\text{the}] & N[\text{cat}] \\ \varepsilon[\epsilon] & N[\text{kota}]\end{pmatrix}\right) \begin{matrix}\$[\$]\\ \$[\$]\end{matrix} \tag{5}$$

$$\overset{*}{\Rightarrow} \bowtie_{\substack{[1,2]\\[1,2]}} \left(\bowtie_{\substack{[1,2,3]\\[1,3,2]}} \begin{pmatrix} \text{I} & \text{fed} \\ \text{ya kormil}\end{pmatrix} \bowtie_{\substack{[1,2]\\[1]}} \begin{pmatrix} \text{the cat} \\ \epsilon\ \text{kota}\end{pmatrix}\right) \begin{matrix}\$\\ \$\end{matrix} = \bowtie_{\substack{[1,2]\\[1,2]}} \begin{pmatrix} \text{(I fed the cat)} & \$ \\ \text{(ya kota kormil)} & \$\end{pmatrix} = \begin{matrix}\text{I fed the cat \$}\\ \text{ya kota kormil \$}\end{matrix} \tag{6}$$

Figure 1: A 2-L_2MTG derivation in English and transliterated Russian: (4-5) DEPEND productions; (6) YIELD productions, followed by rendering.

Some subclasses and superclasses of MTG have been studied before. The non-lexicalized class 2-MTG(2) is equivalent to ITG (Wu, 1997). Alshawi *et al.* (2000)'s "collections of finite-state head transducers" can be viewed as a subclass of 2-LMTG where, among other restrictions, Λ contains only one (dummy) nonterminal label. "Syntax-directed translations of order k" (Aho & Ullman, 1969) are equivalent to k-MTG(2). On the other hand, MTG is a subclass of Multiple CFG (Seki *et al.*, 1991) where the functions that render the RHS of production rules may not mix symbols from different components.

3 Synchronous Parsers

Inference of synchronous structures requires a synchronous parser. A **synchronous parser** is an algorithm that can infer the syntactic structure of each component text in a multitext and simultaneously infer the correspondence relation between these structures.[4] To facilitate complexity analysis (below), we specify our parsers using declarative inference rules.[5] "X :- Y, Z" means that X can be inferred from Y and Z. $\frac{Y,Z}{X}$ means the same thing. An item that appears in an inference rule stands for the proposition that the item is in the parse chart. A production rule that appears in an inference rule stands for the proposition that the production is in the grammar. Such specifications are nondeterministic: they do not indicate the order in which a parser should attempt inferences. A deterministic parsing strategy can always be chosen later, to suit the application. Any reasonable parsing strategy will have the same asymptotic complexity (McAllester, 2002).

3.1 Naive Synchronous Bilexical Parsers

For expository purposes, we begin with Parser R2D2A, which is a naive CKY-style chart parser for 2-L_2MTG(2).[6] The chart of

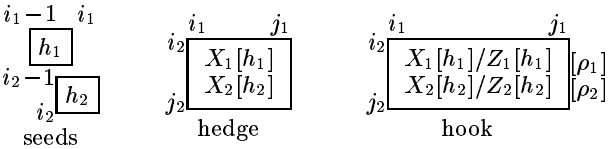

Figure 2: Items used by our parsers for 2-L_2MTG(2).

Parser R2D2A is initialized with "seed" items, illustrated in Figure 2. A one-dimensional seed is put in the chart for every word in every component of the input.

After initialization, the parser can assert the translational equivalence between seeds in different components by firing Y inference rules:

$$\begin{matrix} & i_1-1 & i_1 \\ i_2-1 & \boxed{\begin{matrix}X_1[h_1]\\ X_2[h_2]\end{matrix}} & \\ i_2 & & \end{matrix} \text{ :- } \begin{matrix} i_1-1 & i_1 \\ \boxed{h_1} & \end{matrix}, \begin{matrix} i_2-1 \\ i_2\ \boxed{h_2}\end{matrix}, \begin{matrix} X_1[h_1] \\ X_2[h_2]\end{matrix} \Rightarrow \begin{matrix} h_1 \\ h_2\end{matrix}$$

Y inference rules infer YIELD production rules. Each two-dimensional instantiation expresses the translational equivalence of two word tokens, h_1 and h_2, at positions i_1 and i_2 in their respective components. One-dimensional Y inferences assert that a word vanishes in translation. E.g.:

$$\begin{matrix} i_2-1 \\ i_2 \end{matrix} \boxed{\begin{matrix}\varepsilon[\epsilon]\\ X_2[h_2]\end{matrix}} \text{ :- } \begin{matrix} i_2-1 \\ i_2\ \boxed{h_2}\end{matrix}, \begin{matrix} \varepsilon[\epsilon] \\ X_2[h_2]\end{matrix} \Rightarrow \begin{matrix} \epsilon \\ h_2\end{matrix}$$

Parser R2D2A spends most of its time composing pairs of non-seed items into larger items.[7] A bottom-up one-dimensional parser composes one-dimensional items until it infers an item that covers the input text. A bottom-up synchronous parser composes multi-dimensional items until it infers an item that covers the multitext space spanned by the input multitext. The items composed by naive synchronous parsers are called **hyperedges**, or **hedges** for short. The 2D hedges composed by Parser R2D2A are shown in Figure 2. The particular hedge in the figure represents a constituent between word boundaries $i_1 \ldots j_1$ of the first compo-

[4] A suitable set of monolingual parsers can also infer the syntactic structure of each component, but cannot infer the correspondence relation between these structures.

[5] We use both Horn clauses and sequents to save space.

[6] Parser R2D2A can be compared to Wu (1997)'s procedure for parsing non-lexicalized ITGs, which runs in $O(n^6)$. As Eisner & Satta (1999) have shown, yields of bilexical grammars are generally more expensive to parse than their nonlexicalized counterparts.

[7] The term **item** refers to any partial parse.

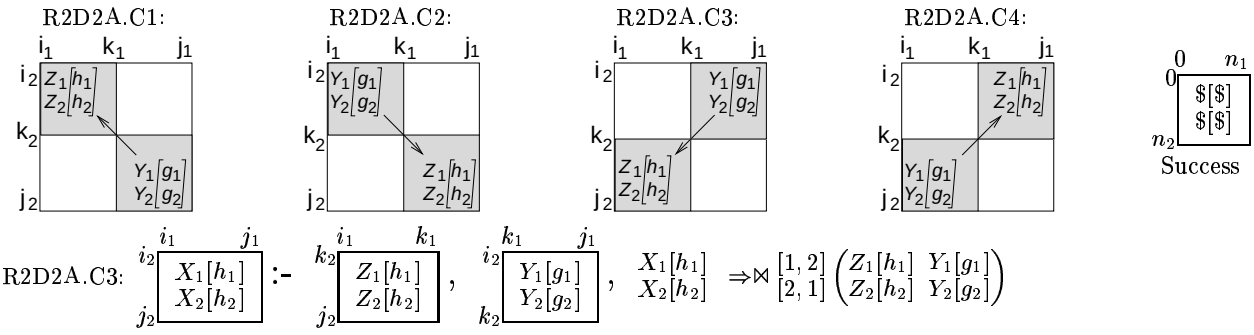

Figure 3: Parser R2D2A has four ways to compose two 2D items represented by the shaded regions into one item represented by their minimum enclosing rectangle. Inference rule R2D2A.C3 is given as an example.

nent and $i_2 \ldots j_2$ of the second component, labeled by the nonterminal link inside the rectangle. Since MTGs can generate multitexts with components of unequal length, the height of the hedge need not equal the width. In particular, if one of the nonterminals in the label is empty, then that dimension has zero width, and its boundary variables are dummies. Such one-dimensional hedges are necessary for representing synchronous parse tree branches that are non-empty in only one component.

Parser R2D2A can compose two hedges whenever they satisfy both of the following constraints.

ID: The grammar must allow the label of one hedge to depend on the label of the other.

LP: The two hedges must be adjacent without overlapping in the bitext space. In particular, if neither hedge has empty components, then they must be corner-to-corner.

Figure 3 illustrates the four ways that these constraints can be satisfied in 2D. Each of the shaded diagrams in the figure corresponds to one of four composition (C) inference rules. Parser R2D2A accepts a bitext with component lengths n_1 and n_2 if and only if it derives item Success, also in Figure 3.

The worst-case space complexity of a chart parser is within a constant factor of the maximum number of items that can be in its chart at the same time. An item's **signature** uniquely determines how the item can combine with other items, so two items with the same signature never differ on whether they can participate in a successful parse. Therefore, we never need to store more than one item with the same signature. The number of possible unique signatures is the product of the ranges of their free variables.

For Parser R2D2A, the free variables in an item's signature are its nonterminals and boundaries in each dimension. Let $l = |\Lambda|$, and let n be the length of the longer component of the input bitext. We assume that n is always smaller than $|T|$. Then the number of possible lexicalized nonterminals in each dimension is in $O(ln)$, for a product of $O(l^2n^2)$ in 2D. Each of the two item boundaries in each of the two dimensions also ranges over $O(n)$ possible word positions, for a product of $O(n^4)$. We conclude that Parser R2D2A requires $O(l^2n^6)$ space.

If we assume that there exists an ordering of inferences that guarantees correctness and avoids duplication of effort, then the worst-case running time of the parser is within a constant factor of the number of possible unique inference rule instantiations (McAllester, 2002). This number is the product of the ranges of the free variables that appear in the inference rules. The C rules dominate the computational expense of Parser R2D2A, with 6 free item boundaries, each of which ranges over $O(n)$ possible word positions. In addition, each C rule involves 6 nonterminal labels, which share 4 different lexical heads, at a cost in $O(l^6n^4)$. We conclude that the running time of Parser R2D2A is in $O(l^6n^{10})$.

Parser R2A is the generalization of Parser R2D2A to arbitrary dimensionality. Parser R2A derives d-dimensional hyperrectangles ($1 \leq d \leq D$), corresponding to d-tuples of substrings of the input multitext. Its items still store two boundaries and one nonterminal per dimension. Each composition rule still involves two items, which share one boundary per dimension, for a rate of $2 \times 2 - 1 = 3$ free boundaries per dimension. The permutation vectors are uniquely determined by the item boundaries, so they add no complexity. Thus, Parser R2A runs in $O(l^D n^{3D})$ space and $O(l^{3D} n^{5D})$ time.

3.2 An Optimization

Eisner & Satta (1999) suggest that any monolingual head-automaton grammar of interest in NLP can be construed as a "split" grammar. Roughly speaking, split grammars are those where every head generates all its dependents on one side before generating any dependents on the other side. Taking advantage of this property, Eisner & Satta present an

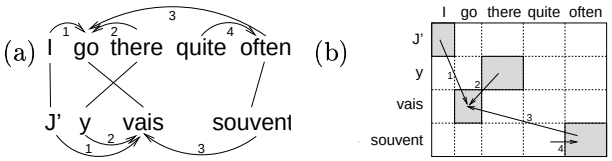

Figure 4: A synchronous bilexical dependency tree in English and French that cannot be generated by a split grammar. The two representations are equivalent: (a) Horizontal arcs are dependencies; vertical lines indicate links. (b) Shaded rectangles are 2D constituents; arcs are 2D dependencies.

$O(n^3)$ parser for split monolingual bilexical grammars. Unfortunately, split *synchronous* grammars preclude many crossing correspondences. E.g., assuming that each link is generated as an atomic unit, there is no way to generate the dependents of "go/vais" in Figure 4 without switching sides twice in one of the components. Crossing correspondences are common in multitexts, so split synchronous grammars are not suitable for our purposes. Since MTGs cannot be construed as "split" grammars, we conjecture that their recognition complexity is higher than $O(n^{3D})$. The finding that Eisner & Satta's $O(n^3)$ optimization does not generalize to the synchronous case has wider implications: A method for analyzing text in $O(n^k)$ is not necessarily capable of analyzing D-dimensional multitext in $O(n^{Dk})$.

Eisner & Satta (1999) also presented a bilexical parser that runs in $O(n^4)$ and does not rely on the "split" property. The optimization in that parser is based on the insight that ID and LP are independent constraints. It is possible to express each constraint in a separate inference rule, such that each rule involves fewer free variables. The worst-case running time of the resulting parser is lower than that of a naive parser.

Parser R2B is a generalization of this quartic-time parser for $\mathrm{L_2MTG}(2)$. Again, we illustrate the 2D case, which we call Parser R2D2B. This parser uses another kind of item, labeled "hook" in Figure 2. Hooks are logically equivalent to CCG-style "slashed" constituents, where the "missing" subconstituent is the heir. The missing heir follows a backslash in the hook's label if it is the first link in its DLV; otherwise it follows a forward slash. Each hook includes the permutation vector that indicates how the hook expects to compose with its missing heir. Hooks represent parser states where an ID constraint has been satisfied, but a suitable LP constraint has not (yet). Using this type of item, each of the naive composition rules can be decomposed into an ID rule and an LP rule. E.g., Rule R2D2A.C3 would decompose into R2D2B.ID3:

$$\begin{array}{l} {\scriptstyle i_1}\quad\quad\quad {\scriptstyle j_1} \\ {\scriptstyle i_2}\boxed{\begin{array}{l} X_1[h_1]\backslash Z_1[h_1] \\ X_2[h_2]\backslash Z_2[h_2] \end{array}}{\scriptstyle \begin{array}{l}[1,2]\\[2,1]\end{array}} \text{ :-} \\ {\scriptstyle j_2} \end{array}$$

$$\begin{array}{l} {\scriptstyle i_1}\quad {\scriptstyle j_1} \\ {\scriptstyle i_2}\boxed{\begin{array}{l} Y_1[g_1] \\ Y_2[g_2] \end{array}}, \begin{array}{l} X_1[h_1] \\ X_2[h_2] \end{array} \Rightarrow\!\bowtie {\scriptstyle \begin{array}{l}[1,2]\\[2,1]\end{array}} \begin{pmatrix} Z_1[h_1]\, Y_1[g_1] \\ Z_2[h_2]\, Y_2[g_2] \end{pmatrix} \\ {\scriptstyle j_2} \end{array}$$

and R2D2B.LP3:

$$\begin{array}{l} {\scriptstyle i_1}\quad {\scriptstyle j_1} \\ {\scriptstyle i_2}\boxed{\begin{array}{l} X_1[h_1] \\ X_2[h_2] \end{array}} \text{ :-} \\ {\scriptstyle j_2} \end{array} \quad \begin{array}{l} {\scriptstyle i_1}\quad\quad {\scriptstyle k_1} \\ {\scriptstyle k_2}\boxed{\begin{array}{l} Z_1[h_1] \\ Z_2[h_2] \end{array}} \\ {\scriptstyle j_2} \end{array}, \quad \begin{array}{l} {\scriptstyle k_1}\quad\quad\quad {\scriptstyle j_1} \\ {\scriptstyle i_2}\boxed{\begin{array}{l} X_1[h_1]\backslash Z_1[h_1] \\ X_2[h_2]\backslash Z_2[h_2] \end{array}}{\scriptstyle \begin{array}{l}[1,2]\\[2,1]\end{array}} \\ {\scriptstyle k_2} \end{array}$$

The other C rules can be decomposed analogously.

Hooks need to keep track of one more nonterminal label per dimension than hedges. Each hook must also store a permutation vector,[8] at a cost in $O(D)$. So, the space complexity of Parser R2B is in $O(Dl^{2D}n^{3D})$. On the other hand, LP rules need to keep track of only one lexical head per dimension, and ID rules have only 2 free boundaries per dimension. Thus, the time complexity of Parser R2B is in $O(l^{3D}n^{4D})$, an $O(n^D)$ savings over Parser R2A.

4 MTG Binarization

Productions with more than two nonterminal links on the RHS are useful for representing subcategorization frames with more than two dependents. For inferring such productions, it is always more efficient to binarize an MTG than to allow a parser to compose more than two items at a time. Grammar binarization can be regarded as an inference process. In contrast to the kind of inference that happens during parsing, binarization *replaces* the antecedent production in the grammar with the consequent productions, rather than merely adding the consequents to the production rule set. Binarization also adds new nonterminals to the nonterminal set N.

A CFG can always be binarized into another CFG. In contrast, binarization of MTG DEPEND productions can require nonterminals with discontinuous yields. For every dimensionality $D \geq 2$ and rank $R \geq 4$, there are some correspondence patterns that can be generated by D-MTG(R), but not by D-MTG($R-1$) (Rambow & Satta, 1999). For example, a grammar in 2-MTG(4) can generate the synchronous parse tree in Figure 5 via Production 7, but no grammar in 2-MTG(3) can generate it. The distinguishing characteristic of such trees is apparent in Figure 5(b): no two constituents are adjacent in more than one dimension. Each set of sibling

[8] The permutation vector in a hook is the same as the one in the production rule from which the hook was inferred.

$$\begin{matrix} S[\text{went}] \\ S[\text{gayee}] \end{matrix} \Rightarrow\bowtie \begin{matrix} [1,2,3,4] \\ [3,1,4,2] \end{matrix} \begin{pmatrix} N[\text{Pat}] & V[\text{went}] & P[\text{home}] & A[\text{early}] \\ N[\text{Pat-nay}] & V[\text{gayee}] & P[\text{ghar}] & A[\text{juldee}] \end{pmatrix} \tag{7}$$

$$\begin{matrix} S[\text{went}] \\ S[\text{gayee}] \end{matrix} \Rightarrow\bowtie \begin{matrix} [1,2] \\ [2,1,2] \end{matrix} \begin{pmatrix} N[\text{Pat}] \\ N[\text{Pat-nay}] \end{pmatrix} \left(\bowtie \begin{matrix} [1,2] \\ [1],[2,1] \end{matrix} \left(\bowtie \begin{matrix} [1,2] \\ [2],[1] \end{matrix} \begin{pmatrix} V[\text{went}] & P[\text{home}] \\ V[\text{gayee}] & P[\text{ghar}] \end{pmatrix} A[\text{early}] \right) \right) \tag{8}$$

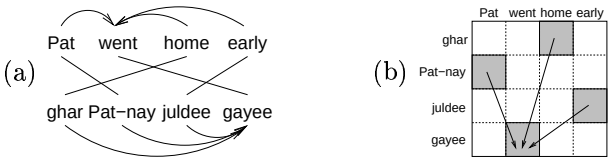

Figure 5: Discontinuous constituents are practically unavoidable in synchronous parsing, as for this bitext of English and transliterated Hindi.

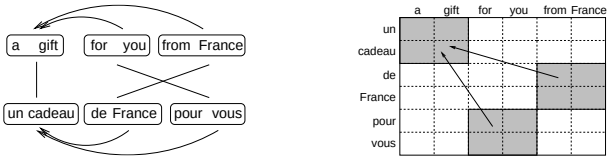

Figure 6: Discontinuous constituents are required for bilexical parsing, even for simple bitexts in English and French.

constituents in such an arrangement must be encapsulated in the RHS of a single DEPEND production.

Bilexical grammars are usually parsed by attaching dependents to their head, rather than to each other. This practice may require the use of discontinuous constituents for inferring L_2MTG productions with only two dependents on the RHS (not counting the heir). For example, as illustrated in Figure 6, bilexical parsing of discontinuous constituents is required whenever a bitext involves two prepositional phrases whose order is switched in translation. Synchronous bilexical parsers that cannot handle discontinuous constituents are unlikely to have good coverage, even for syntactically similar languages such as English and French!

To binarize MTGs, we must generalize the definition of an MTG DEPEND production. We do so by generalizing the kinds of objects that are input to and output from the rendering function $\bowtie$. First, for each discontinuous nonterminal in $\mathbf{M}$, we specify a cardinality, which indicates the number of contiguous spans in the nonterminal. Links of possibly discontinuous constituents are called **discontinuous links** or **d-links**. Second, we partition the permutations in $\mathbf{P}$ to indicate where discontinuities should appear in the output of $\bowtie$. Brackets group contiguous spans in each permutation. Since each nonterminal can now comprise more than one span, we must allow roles to repeat in a given component of $\mathbf{P}$. A permutation cannot have repeating elements, so we call it a **role template** instead. If a role appears in a role template more than once, then the leftmost instance corresponds to the first span of the relevant nonterminal, the second instance to the second span, and so on. See Productions 11-13 below for examples. The links and permutations described in Section 2 are special cases of d-links and role templates, respectively, without discontinuities. The result of joining a d-link vector (DLV) according to a role template vector (RTV) is a d-link.

Using this generalized join operator, we can decompose the RHS of any MTG DEPEND production into nested joins. For example, we can decompose (7) into (8). A production decomposed into nested joins of two d-links each can be immediately binarized according to the MTG Binarization Rules, in which $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z_E}$ are d-links, ρ is an RTV, and Γ is a nested join:

$$\frac{\mathbf{X} \Rightarrow\bowtie \rho(\mathbf{Y}\Gamma)}{\mathbf{Z_E}\ ,\ \mathbf{X} \Rightarrow\bowtie \rho(\mathbf{Y}\mathbf{Z_E})\ ,\ \mathbf{Z_E} \Rightarrow \Gamma} \tag{9}$$

$$\frac{\mathbf{X} \Rightarrow\bowtie \rho(\Gamma\mathbf{Y})}{\mathbf{Z_E}\ ,\ \mathbf{X} \Rightarrow\bowtie \rho(\mathbf{Z_E}\mathbf{Y})\ ,\ \mathbf{Z_E} \Rightarrow \Gamma} \tag{10}$$

These rules replace each production that involves a nested join with two new productions, one of which has no nested joins and the other of which has fewer nested joins than the antecedent. The rules also create a new nonterminal d-link $\mathbf{Z_E}$, which is associated with the equivalence class $\mathbf{E}$ of syntactic contexts in which $\mathbf{Z_E}$ appears.

The Binarization Rules are applied recursively until every production in the grammar has rank ≤ 2. Each discontinuous nonterminal in the LHS of each binarized production rule is annotated with its cardinality, to match the number of partitions in the relevant role template. For example, binarization would replace Production 8 with 3 new production rules, where VP and $\bar{V}$ are new nonterminals:[9]

$$\begin{matrix} S \\ S \end{matrix} \Rightarrow \bowtie \begin{matrix} [1,2] \\ [2,1,2] \end{matrix} \begin{pmatrix} N & VP \\ N & VP(2) \end{pmatrix} \tag{11}$$

$$\begin{matrix} VP \\ VP(2) \end{matrix} \Rightarrow \bowtie \begin{matrix} [1,2] \\ [1],[2,1] \end{matrix} \begin{pmatrix} \bar{V} & A \\ \bar{V}(2) & A \end{pmatrix} \tag{12}$$

$$\begin{matrix} \bar{V} \\ \bar{V}(2) \end{matrix} \Rightarrow \bowtie \begin{matrix} [1,2] \\ [2],[1] \end{matrix} \begin{pmatrix} V & P \\ V & P \end{pmatrix} \tag{13}$$

Binarization does not affect YIELD productions, so it does not change the generative power of MTGs.

[9] We omit their lexical heads and equivalence classes for clarity.

rank	with distinguished heir	without
2	2	2
3	3	2
4 or 5	3	3
6	4	3
7 to 9	4	4

Table 1: Highest possible cardinality of minimizing decompositions over all 2D productions of the given rank. Figures 6 and 5 exemplify highest-cardinality productions of ranks 3 and 4, respectively.

As we shall see in Section 5, however, bad binarization can worsen recognition complexity. The Binarization Rules apply deterministically,[10] but there are multiple ways to decompose the RHS of a non-binary DEPEND production into nested joins.[11] Some decompositions may give rise to more discontinuities than others. Let the **cardinality of an RTV** be the total number of partitions in all its components, and let the **cardinality of a decomposition** be the maximum cardinality of the RTVs that it contains. A **minimizing decomposition** for a given production is one of those with lowest cardinality. Then, the **cardinality of a production** is the cardinality of its minimizing decomposition. The cardinality of a production is bounded by its rank, as Table 1 shows for the 2D case. Finally, the **cardinality $\mathcal{C}(G)$ of an MTG** G is the maximum of the cardinalities of its productions.

5 Inference of Discontinuous Constituents

Parser A is a parser for arbitrary MTGs. It initializes its chart and fires Y inferences just like Parser R2A. It then composes pairs of items into larger items using inference rule A.C (see below). Just like items in ordinary parsers, Parser A items need to know their positions in the input multitext, but not their internal structure. However, items with discontinuities need to remember all their boundaries, not just the outermost ones. Expanding on Johnson (1985), we define a **discontinuous span** (or **d-span**, for short) as a list of zero or more intervals $\sigma = (l_1, r_1; \ldots; l_m, r_m)$, where

- the l_i are left boundaries and the r_i are right boundaries between word positions in a text, so that $l_i < r_i$ for $1 \leq i \leq m$;
- $r_i \leq l_{i+1}$ for $1 \leq i \leq m-1$, which means that the intervals do not overlap.

[10] Given predefined equivalence classes for new nonterminals.

[11] For correct binarization of productions with a distinguished heir, the decomposition must put the heir in the most deeply nested DLV. This requirement tends to increase the cardinality of L_2MTGs, as shown in Table 1.

In addition, we say that a d-span is in normal form if all the inequalities between r_i and l_{i+1} are strict, i.e. there is a gap between each pair of consecutive intervals. Now, a hedge item $X(\sigma)$ in Parser A is a d-link X together with a vector of d-spans σ in normal form. The **cardinality of an item** is the total number of intervals in its d-span vector.

Binarized MTG productions can be inferred under generalizations of the ID and LP constraints described in Section 3. We use two helper functions to express these constraints. $+$ is the concatenation operator for d-spans: Given two d-spans, it outputs the union of their intervals in normal form.[12] The $\otimes$ function computes the role template that describes the relative positions of the intervals in two d-spans. E.g., if $\upsilon = (1,3;8,9)$ and $\sigma = (7,8)$, then $\upsilon + \sigma = (1,3;7,9)$ and $\upsilon \otimes \sigma = [1],[2,1]$. Both operators apply componentwise to vectors of d-spans. With their help, we state the composition inference rule of Parser A:

$$\text{A.C:} \quad \frac{\mathbf{Y}(\upsilon), \mathbf{Z}(\sigma), \mathbf{X} \Rightarrow \bowtie [\upsilon \otimes \sigma](\mathbf{Y}, \mathbf{Z})}{\mathbf{X}(\upsilon + \sigma)}$$

The space complexity of Parser A is a function of the maximum number of boundaries stored in its item signatures, and the number $|N|$ of nonterminals in the grammar. The maximum number of required boundaries is exactly twice the cardinality of the MTG, and each of the boundaries can range over $O(n)$ possible positions. Thus, the space complexity of Parser A for an MTG G is in $O(|N|^D n^{2\mathcal{C}(G)})$. If G is bilexical, then the number of possible nonterminals hides a factor of n^D, raising the space complexity of Parser A to $O(l^D n^{D+2\mathcal{C}(G)})$.

The time complexity of Parser A depends on how many boundaries are shared between antecedent items in A.C rules. In the best case, all the boundaries are shared except the two outermost boundaries in each dimension, and the inferred item is contiguous. In the worst case, no boundaries are shared, and the inferred item stores all the boundaries of the antecedent items. In any case, if y and z are the cardinalities of the composed items, and x is the cardinality of the inferred item, then the number of free boundaries in an A.C inference is $x + y + z$. Thus, in the worst case, the number of free boundaries involved in an A.C inference is $3\mathcal{C}(G)$. As before, each boundary can range over $O(n)$ possible values, where n is the length of the longest component of the input multitext. We still have 3 nonterminal labels per dimension per inference. Also, each inference now needs to compute an RTV at a cost

[12] The inputs of $+$ must have no overlapping intervals, or else the output is undefined.

in $O(\mathcal{C}(G))$. Thus, the time complexity of Parser A is in $O(\mathcal{C}(G)|N|^{3D}n^{3\mathcal{C}(G)})$. For a binarized L_2MTG, which also needs to keep track of two lexical heads per dimension per inference, this complexity rises to $O(\mathcal{C}(G)l^{3D}n^{2D+3\mathcal{C}(G)})$.

Parser B is a generalization of Parser R2B for binarized L_2MTGs of arbitrary rank. It decomposes inference rule A.C into ID and LP subrules, using generalized hooks that carry an RTV. The decomposition can happen in one of two ways, depending on the heir's role (1 or 2) in the DLV.

B.ID1: $$\frac{\mathbf{Y}[\mathbf{g}](\sigma), \mathbf{X}[\mathbf{h}] \Rightarrow\!\bowtie [\upsilon \otimes \sigma](\mathbf{Z}[\mathbf{h}], \mathbf{Y}[\mathbf{g}])}{\mathbf{X}[\mathbf{h}](\sigma)[\upsilon \otimes \sigma]\backslash\mathbf{Z}[\mathbf{h}]}$$

B.LP1: $$\frac{\mathbf{Z}[\mathbf{h}](\upsilon), \mathbf{X}[\mathbf{h}](\sigma)[\upsilon \otimes \sigma]\backslash\mathbf{Z}[\mathbf{h}]}{\mathbf{X}[\mathbf{h}](\upsilon + \sigma)}$$

B.ID2: $$\frac{\mathbf{Y}[\mathbf{g}](\upsilon), \mathbf{X}[\mathbf{h}] \Rightarrow\!\bowtie [\upsilon \otimes \sigma](\mathbf{Y}[\mathbf{g}], \mathbf{Z}[\mathbf{h}])}{\mathbf{X}[\mathbf{h}](\upsilon)[\upsilon \otimes \sigma]/\mathbf{Z}[\mathbf{h}]}$$

B.LP2: $$\frac{\mathbf{Z}[\mathbf{h}](\sigma), \mathbf{X}[\mathbf{h}](\upsilon)[\upsilon \otimes \sigma]/\mathbf{Z}[\mathbf{h}]}{\mathbf{X}[\mathbf{h}](\upsilon + \sigma)}$$

The rules in Section 3.2 are simple examples of B.ID1 and B.LP1.

Parser B is faster than Parser A, but takes more space. The hooks of Parser B must keep track of one more nonterminal label per dimension than hedges. The size of an RTV is bounded by the cardinality of the grammar. Thus, the space complexity of Parser B is in $O(\mathcal{C}(G)l^{2D}n^{D+2\mathcal{C}(G)})$. On the other hand, The B.ID rules involve only one d-span instead of two, reducing the number of free variables by $O(\mathcal{C}(G))$. The B.LP rules again involve only one lexical head instead of two, reducing the number of free variables by a factor of D. Since $D \leq \mathcal{C}(G)$, it turns out that the worst-case running time of Parser B is less than that of Parser A by a factor of n^D under L_2MTGs of any rank and dimensionality.

6 Conclusion

We have proposed Multitext Grammars (MTGs) as a convenient and relatively expressive foundation for building practical models of translational equivalence. To encourage their use for this purpose, we have explored algorithms for parsing bilexical MTGs of arbitrary rank and dimensionality. Our exploration highlighted some little-known properties of synchronous parsing: (1) some optimizations of monolingual parsers generalize to the synchronous case, but others do not; (2) discontinuous constituents are essential for parsing bitexts even in similar Western languages; (3) different binarization schemes lead to different time and space complexity.

There are many aspects of translational equivalence that MTG cannot express, such as some of those described by Dorr (1994). In future work, we hope to extend the formalism to cover some of the aspects that would not raise the computational complexity of its recognition, such as discontinuous and/or phrasal terminals. Concurrently, we shall explore the empirical properties of MTG, by inducing stochastic MTGs from real multitexts.

Acknowledgments

Thanks to Jason Eisner, Sanjeev Khudanpur, Owen Rambow, Giorgio Satta, and members of NYU's Proteus Project for helpful discussions. The idea of treating binarization as an optimization problem is due to Wei Wang. Dan Klein proposed the term "hook." This research was supported by the DARPA TIDES program, by an NSF CAREER award, and by a gift from Sun Microsystems.

References

A. Abeillé, Y. Schabes, and A. Joshi (1990) "Using Lexicalized Tree Adjoining Grammars for Machine Translation," *Proceedings of COLING.*

A. Aho and J. Ullman (1969) "Syntax Directed Translations and the Pushdown Assembler," *Journal of Computer and System Sciences 3*, 37-56.

H. Alshawi, S. Bangalore, and S. Douglas (2000) "Learning Dependency Translation Models as Collections of Finite State Head Transducers," *Computational Linguistics 26(1)*:45-60.

B. Dorr (1994) "Machine Translation Divergences: A Formal Description and Proposed Solution," *Computational Linguistics, 20:4*, 597-633.

J. Eisner and G. Satta (1999) "Efficient Parsing for Bilexical Context-Free Grammars and Head-Automaton Grammars," *Proceedings of the ACL.*

M. Johnson (1985) "Parsing with Discontinuous Constituents," *Proceedings of the ACL.*

D. McAllester (2002) "On The Complexity Analysis of Static Analyses" *Journal of the ACM 49(4).*

O. Rambow and G. Satta (1999) "Independent Parallelism in Finite Copying Parallel Rewriting Systems," *Theoretical Computer Science 223*:87-120.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami (1991) "On Multiple Context-Free Grammars," *Theoretical Computer Science 88*:191-229.

S. Shieber and Y. Schabes (1990) "Synchronous Tree-Adjoining Grammars," *Proceedings of COLING.*

S. Shieber, Y. Schabes, and F. Pereira (1995) "Principles and Implementation of Deductive Parsing," *Journal of Logic Programming 24*:3-36.

D. Wu (1997) "Stochastic inversion transduction grammars and bilingual parsing of parallel corpora," *Computational Linguistics 23(3)*:377-404.