Computational Linguistics in the Netherlands Journal 4 (2014) 137-148 Submitted 06/2014; Published 12/2014

Conversions between D and $MCFG$: Logical Characterizations of the Mildly Context-Sensitive Languages

Gijs Jasper Wijnholds* GIJSWIJNHOLDS@HOTMAIL.COM

*ILLC, Amsterdam, The Netherlands

Abstract

The Displacement Calculus (DC) of Morrill et al. (2011) extends Lambek categorial grammar with wrapping operations for handling discontinuous dependencies. In this paper, we study two computationally attractive fragments of DC and relate them via equivalence proofs to the well-nested Multiple Context Free Grammars (MCFG) of Kanazawa (2009). A first fragment, first-order DC, has been shown in earlier work to enjoy polynomial parsability, but it sacrificies the hypothetical reasoning facilities that form a central feature of the syntax-semantics interface in logic grammars. Our main result then is to identify a second fragment, restricted two-dimensional DC, that allows for hypothetical reasoning, and to provide a conversion of Displacement Context-free Grammar, a formalism equivalent to well-nested MCFG, into our restricted two-dimensional Displacement Grammar.

1. Introduction

A central theme in computational linguistics is the balance between expressivity and computational complexity. On the expressivity side, it is well known (Huybregts 1984, Shieber 1987) that natural language exhibits patterns that are not recognizable by context-free grammar. The challenge then is to extend the recognizing capacity beyond context-free, while still maintaining good computational properties, in particular polynomial parsability, a crucial feature for formalisms employed in actual language technology. Formalisms with the desired expressivity/complexity mix are known as *Mildly Context-Sensitive Grammars*, after Joshi et al. (1990). Examples are Tree Adjoining Grammars, Combinatory Categorial Grammars, Linear Indexed Grammars and Multiple Context-free Grammars (MCFG). The latter are our point of reference in this paper. In a MCFG, non-terminals range over *tuples* of strings (rather than simply strings, as in a CFG). The size of these tuples (or the *dimension*) then gives rise to a fine-grained hierarchy of k-MCFG, with growing expressivity as the dimension k increases. Kanazawa (2009) has further refined this hierarchy by distinguishing a subclass of *well-nested* MCFG. Well-nestedness puts restrictions on how the elements of string tuples can be intercalated, and the claim is that well-nestedness would already offer enough expressivity to capture the phenomena of discontinuity that occur. An excellent textbook dealing with the mildly context-sensitive family of grammar formalisms, their linguistic motivation, and their use in language technology (particularly for grammar induction from dependency treebanks and treebanks with discontinuities) is Kallmeyer (2010).

In the tradition of logic grammars, Displacement Calculus (DC) (Morrill and Valentín 2010) has been proposed as a framework that overcomes the context-free limitations of Lambek categorial grammars. Whereas the Lambek calculus can be seen as the logic of strings composed by concatenation, DC adds facilities for dealing with *split strings*: expressions consisting of detached parts. An example would be the idiom 'give – the cold shoulder'. To build the phrase 'give someone the cold shoulder', one *wraps* the discontinuous expression around its object. For the *general* formulation of DC, the generative capacity is unknown. For a restricted *fragment*, namely first-order DC, the corresponding grammars have the same generative capacity as well-nested MCFG, as was shown by Wijnholds (2011) and Sorokin (2013). The proof of one direction of the equivalence as given by

Sorokin (2013) relies on a new grammatical formalism called Displacement Context-Free Grammar (DCFG) which is equivalent to well-nested MCFG. We complement the mentioned results by showing that a different fragment of DC, namely restricted two-dimensional DC, gives grammars that can be obtained from a DCFG. Our motivation for studying this fragment is that it keeps the facilities for hypothetical reasoning which forms an essential ingredient of the syntax-semantics interface in logic grammars: hypothetical reasoning corresponds to lambda binding in the semantic representation associated with derivations, and as such provides the basis for the semantic interpretation of quantifier scope construal and long-distance dependencies.

This paper is organized as follows: in Section 2 we define Displacement Context-Free Grammars as given by Sorokin (2013) and we introduce two restrictions of the Displacement Calculus, first-order Displacement Calculus and restricted two-dimensional Displacement Calculus. In Section 3 we recall the equivalence between Displacement Context-Free Grammar and first-order Displacement Calculus as it is stated in Sorokin (2013) combined with Wijnholds (2011). Section 4 provides our main result, namely the conversion of a Displacement Context-Free Grammar into a restricted two-dimensional Displacement Grammar. We conclude with some suggestions for further research.

2. Preliminaries

We introduce — in order of appearance — Displacement Context-Free Grammar, first-order Displacement Grammar and restricted two-dimensional Displacement Grammar.

Let Σ be an alphabet. We denote by ϵ the empty word and by 1 a separator (a unique element not occurring in Σ). Furthermore, we use the abbreviations $\Sigma_1 = \Sigma \cup \{1\}$, $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$, $\Sigma_{\epsilon,1} = \Sigma \cup \{\epsilon, 1\}$. For k a natural number, define $Op_k = \{\cdot, +_1, ..., +_k\}$. The set of well-formed terms built from $\Sigma_{\epsilon,1}$ using Op_k is denoted by $T_k(\Sigma_{\epsilon,1})$ and is defined inductively, together with the *rank* (denoted by $r(\cdot)$)[1] of a term: for $a \in \Sigma_\epsilon$, a is a well-formed term with $r(a) = 0$. 1 is a term with $r(1) = 1$. If α and β are terms such that $r(\alpha) + r(\beta) \leq k$, then $\alpha \cdot \beta$ is a term with $r(\alpha \cdot \beta) = r(\alpha) + r(\beta)$. For $1 \leq i \leq k$, if α and β are terms such that $r(\alpha) + r(\beta) - 1 \leq k$ and $i \leq r(\alpha)$, then $\alpha +_i \beta$ is a term with $r(\alpha +_i \beta) = r(\alpha) + r(\beta) - 1$. We interpret $\cdot$ as concatenation, i.e. an associative operation with unit ϵ and we interpret $+_i$ as wrapping with respect to the ith separator, i.e. $\alpha \cdot 1 \cdot \beta +_i v = \alpha \cdot v \cdot \beta$ where $r(\alpha) = i - 1$. Note that we have $\alpha +_i 1 = \alpha$ for $i \leq r(\alpha)$ and $1 +_1 \alpha = \alpha$. Naturally, the notation $T_k(\Sigma_{\epsilon,1})$ extends to any alphabet, for instance we could also have $T_k(\Sigma_{\epsilon,1} \cup N)$ where N is an arbitrary set of symbols. This just means that the symbols of N may appear in terms and that they do not increase the rank of a term in any way. Finally, we introduce the notion of *context* of a term, this will be useful when introducing the Displacement Calculus.

Definition 1 *A context $\Delta\langle\rangle$ is a term with a unique occurrence of a hole $\langle\rangle$ in it. The set of contexts is defined as follows:*

$$\Delta\langle\rangle, \Gamma\langle\rangle := \langle\rangle \mid \Delta\langle\rangle * \Gamma \mid \Delta * \Gamma\langle\rangle$$

where $* \in Op_k$.

We then can refer to $\Delta\langle\Gamma\rangle$ as the context $\Delta\langle\rangle$ in which the hole is replaced by Γ (note that this is only defined if it results in a well-formed term).

We are now ready to define Displacement Context-Free Grammars as given by Sorokin (2013):

Definition 2 *A k-Displacement Context-Free Grammar (k-DCFG) is a quadruple $G = \langle \Sigma, N, S, P \rangle$ where Σ is a finite alphabet, N is a finite ranked set of non-terminal symbols with $\Sigma \cap N = \emptyset$, $S \in N$ is a distinguished start symbol with $r(S) = 0$ and P is a set of rules of the form $A \to \alpha$. Here we must have $A \in N$ and $\alpha \in T_k(\Sigma_{\epsilon,1} \cup N)$ and $r(A) = r(\alpha)$.*

1. The rank of a term simply counts the number of separators in it.

Definition 3 *For every $\alpha \in T_k(\Sigma_{\epsilon,1} \cup N)$, we define $yield(A)$ as the smallest set satisfying the following:*

1. $yield(a) = \{a\}$ *for* $a \in \Sigma_{\epsilon,1}$,
2. $yield(\alpha * \beta) = yield(\alpha) * yield(\beta)$ *for* $* \in Op_k$,
3. $yield(\alpha) \subseteq yield(A)$ *for* $A \to \alpha \in P$.

We furthermore define the language of a k-DCFG G as $L(G) = yield(S)$. This concludes the definition of Displacement Context-Free Grammar.

2.1 Displacement Calculus and its Fragments

We define general Displacement Grammar in terms of labelled natural deduction, allowing us to naturally restrict the system to first-order Displacement Grammar and restricted two-dimensional Displacement Grammar.

Labelled natural deduction will tell us what kind of terms can be associated with what kind of types. When a term α is associated with a certain type A, we will say that the term is an *inhabitant* of the type, denoted by $\alpha : A$. We will first give the formal definition of types using the connectives $\bullet, \backslash, /, \odot_k, \uparrow_k, \downarrow_k$ and afterwards explain how natural deduction acts upon types. Just as we define a rank for each term, we provide types with *sorts*, meaning that the sort of the type should correspond to the rank of all of its inhabitants.

Let Tp be a sorted set of atomic types, i.e. such that there is a map $s : Tp \to \mathbb{N}$. The sort of an atomic type will refer to the number of separators its corresponding terms will have. The set of sorted *general Displacement types* $F(Tp)$ is defined as follows: if $A \in Tp$, then $A \in F(Tp)$ and $s(A)$ is given. If $A, B \in F(Tp)$ then $A \bullet B \in F(Tp)$ with $s(A \bullet B) = s(A) + s(B)$. If $A, B \in F(Tp)$ with $s(A) \leq s(B)$, then $B/A, A\backslash B \in F(Tp)$ with $s(B/A) = s(A\backslash B) = s(B) - s(A)$. If $A, B \in F(Tp)$ with $1 \leq s(A)$ we have for every $k \leq s(A)$ that $A \odot_k B \in F(Tp)$ with $s(A \odot_k B) = s(A) + s(B) - 1$. If $A, B \in F(Tp)$ and $s(A) \leq s(B)$ then for every $k \leq s(B) - s(A) + 1$ we have $A \downarrow_k B, B \uparrow_k A \in F(Tp)$ with $s(A \downarrow_k B) = s(B \uparrow_k A) = s(B) - s(A) + 1$. We furthermore have two constants $I, J \in F(Tp)$ with $s(I) = 0$ and $s(J) = 1$.

We need to introduce a notion of order because we use it in our proofs. The order of a type is inductively defined as follows: $ord(A) = 0$ for $A \in Tp$, $ord(A \bullet B) = max(ord(A), ord(B))$, $ord(A\backslash B) = ord(B/A) = max(ord(A) + 1, ord(B))$, $ord(A \odot_k B) = max(ord(A), ord(B))$, and finally $ord(A \downarrow_k B) = ord(B \uparrow_k A) = max(ord(A) + 1, ord(B))$.

We will now give an intuitive interpretation of what types do. As noted above, labelled natural deduction associates terms with certain types, and in our system of natural deduction we have constructive and destructive operations on types (and hence on terms). We begin with an explanation of constructive operations: the $\bullet$ is called the *concatenation connective*, intuitively corresponding to the idea that from a term $\alpha : A$ and a term $\beta : B$ we can construct $\alpha \cdot \beta : A \bullet B$. The connectives $\backslash, /$ are then called *residuals* with respect to $\bullet$, which means that a term $\beta : A\backslash B$ will want to concatenate to the left with $\alpha : A$ to give a term $\alpha \cdot \beta : B$. By a similar process, $/$ is the right residual of $\bullet$. The $\odot_k$ connective we will call the *wrapping connective*, corresponding to the idea that from a term α inhabiting A (the *circumfix*) and a term β inhabiting B (the *infix*), we can form the term $\alpha +_k \beta$ inhabiting $A \odot_k B$ (provided that the ranks/sorts are respected). Here we have $\uparrow_k, \downarrow_k$ as residuals with respect to $\odot_k$, meaning that a term $\gamma : B \uparrow_k A$ will want a term $\alpha : A$ to be inserted to give $\gamma +_k \alpha : B$. Similarly, a term $\gamma : A \downarrow_k B$ inserts itself into a term $\alpha : A$ in order to get $\alpha +_k \gamma : B$. These operations correspond to the introduction rules for $\bullet$ and $\odot_k$ and the elimination rules for their residuals, respectively. As for the destructive operations on types, these correspond to the elimination rules for $\bullet$ and $\odot_k$ and the introduction rules for their residuals: if we

can derive from the assumptions that $\alpha : A$ and $\beta : B$ a term $\Delta\langle\alpha \cdot \beta\rangle : C$ and we have derived a term $\gamma : A \bullet B$ then we may derive $\Delta\langle\gamma\rangle : C$. Similarly, we may derive $\Delta\langle\gamma\rangle : C$ given that we have derived $\gamma : A \odot_k B$ and $\Delta\langle\alpha +_k \beta\rangle : C$ from the assumptions $\alpha : A$ and $\beta : B$. For the residuals, we need only one assumption: if we can derive $\alpha \cdot \gamma : B$ from the assumption $\alpha : A$ we may discard α and derive $\gamma : A\backslash B$. Similarly we may derive $\gamma : B/A$ given that we have derived $\gamma \cdot \alpha : B$ from the assumption $\alpha : A$. From a derivation of $\alpha +_k \gamma : B$ out of the assumption $\alpha : A$ we may derive $\gamma : A \downarrow_k B$ and given a derivation $\gamma +_k \alpha : B$ from assumption $\alpha : A$ we may derive $\gamma : B \uparrow_k A$. Finally, we have two constant types I, J that will have as their sole inhabitant ϵ and 1 respectively. This is reflected in the fact that they take the form of axioms in the system. The types I and J act as units with respect to $\bullet$ and $\odot_k$ exactly like the terms ϵ and 1 are units to $\cdot$ and $+_k$.

All the operations on term/type associations are summarized in the following definition, due to Morrill et al. (2011):

Definition 4 (Natural Deduction) *The proof theory for the full General Displacement Calculus is as follows:*

$$\frac{}{\epsilon : I}\ Ax.I \qquad \frac{\alpha : A \quad \beta : B}{\alpha \cdot \beta : A \bullet B}\ I\bullet \qquad \frac{\gamma : A \bullet B \quad \Delta\langle\alpha \cdot \beta\rangle : C}{\Delta\langle\gamma\rangle : C}\ E\bullet$$

$$\frac{\alpha : A}{\gamma : A\backslash B}\ I\backslash \qquad \frac{\alpha : A \quad \gamma : A\backslash B}{\alpha \cdot \gamma : B}\ E\backslash \qquad \frac{\alpha : A}{\gamma : B/A}\ I/ \qquad \frac{\gamma : B/A \quad \alpha : A}{\gamma \cdot \alpha : B}\ E/$$

$$\frac{}{1 : J}\ Ax.J \qquad \frac{\alpha : A \quad \beta : B}{\alpha +_k \beta : A \odot_k B}\ I\odot_k \qquad \frac{\gamma : A \odot_k B \quad \Delta\langle\alpha +_k \beta\rangle : C}{\Delta\langle\gamma\rangle : C}\ E\odot_k$$

$$\frac{\alpha : A}{\alpha +_k \gamma : B}\ I\downarrow_k \qquad \frac{\alpha : A \quad \gamma : A \downarrow_k B}{\alpha +_k \gamma : B}\ E\downarrow_k$$

$$\frac{\alpha : A}{\gamma +_k \alpha : B}\ I\uparrow_k \qquad \frac{\gamma : B \uparrow_k A \quad \alpha : A}{\gamma +_k \alpha : B}\ E\uparrow_k$$

where $k \in \mathbb{N}$.

Now we need a system to characterize sets of strings. This is achieved by defining *Displacement Grammars* over sets of atomic types:

Definition 5 *A Displacement Grammar (DG) over a set of types* Tp *is a triple* $G = \langle\Sigma, \delta, S\rangle$ *where* Σ *is a finite alphabet,* $\delta \subseteq \Sigma \times F(Tp)$ *is a type assignment relation such that for every* $(a, T) \in \delta$ *we have that* $r(a) = s(T) = 0$*, and* $S \in F(Tp)$ *is a distinguished start type with* $s(S) = 0$.

Given a Displacement Grammar G we can reason with strings in the system by adding the rule

$$\frac{(a, A) \in \delta}{a : A}\ Lex.$$

which states that we can associate any letter in the grammar with some type given in the type assignment relation, and construct further derivations with it. We then would like to define the *dimension* of a DG over Tp as the maximal sort of a type in Tp, and call a DG of dimension k a k-DG. Furthermore, we want to define the *language* of a DG as follows: let $\vdash_C w : S$ denote that according to the labelled natural deduction of calculus C (up to this point the only calculus we have considered is the full general Displacement Calculus $\mathbf{D}$) one can derive $w : S$. Then the language generated by a k-DG G is defined as $L(G) = \{w \in \Sigma^* | \vdash_C w : S\}$.

We will now define the two (different) fragments of the full general Displacement Calculus that we will investigate:

Definition 6 *We obtain first-order Displacement Calculus, denoted by* $\mathbf{D}^1$*, by dropping the* $E\bullet$,$E\odot_k$, $I\backslash$, $I/$, $I\downarrow_k$, $I\uparrow_k$ *rules. We obtain restricted two-dimensional Displacement Calculus, denoted* 1-$\mathbf{D}_J$ *from first-order Displacement Calculus by dropping the* $I\odot_k$, $E\downarrow_k$, $E\uparrow_k$ *rules for* $k > 1$ *but adding the* $I\uparrow_1$ *rule with the restriction that* $\alpha \neq \epsilon$.

It follows from the proof system of $\mathbf{D}^1$ that it is, in such a grammar, one can do with types that are only of order less than or equal to 1. This means that in effect, using $\mathbf{D}^1$ as a proof system and having only first-order types in the lexicon are essentially the same. For a 1-$\mathbf{D}_J$ grammar we omit the subscripts as we only have 1 as a subscript, i.e. we write $\odot, \uparrow, \downarrow$ instead of $\odot_1, \uparrow_1, \downarrow_1$.

The difference between the two fragments $\mathbf{D}^1$ and 1-$\mathbf{D}_J$ is that the former only uses first-order types but with (possibly) a lot of separators in its terms, while the latter system only has one separator in its terms at a time, but allows for *hypothetical reasoning* as it keeps the $I\uparrow_1$ rule. We will give some examples to illustrate the difference between the two systems, and our main result will show that at least $\mathbf{D}^1$ can be simulated in the system 1-$\mathbf{D}_J$.

This concludes the definition of first-order and restricted two-dimensional Displacement Grammar.

Examples

To illustrate the definitions above, we will provide some examples of phenomena that can be described using the systems $\mathbf{D}^1$ and 1-$\mathbf{D}_J$.

Natural language phenomena like *cross-serial dependencies* in Dutch and Swiss German but also *unbounded scrambling* in German and Korean exemplify the need for going beyond the context-free boundary. By means of a suitable string homomorphism such patterns can be mapped to formal languages like the copy language $\{w^2 \mid w \in \{a, b\}^+\}$, the kind of pattern that can be handled by Tree Adjoining Grammars (Kallmeyer 2010, Chapter 1). To understand how copying patterns can be dealt with in $\mathbf{D}^1$ and 1-$\mathbf{D}_J$, we consider the double copy language $\{w^3 \mid w \in \{a, b\}^+\}$, which goes beyond the expressivity of Tree Adjoining Grammars. The reason for taking w^3 rather than w^2 is that for the standard w^2 copy language one only would need one separator and thus it would not very much distinguish the systems $\mathbf{D}^1$ and 1-$\mathbf{D}_J$.

Any displacement grammar can be denoted by indicating what the distinguished goal type S is, and then writing a lexical entry $(a, A) \in \delta$ by simply writing $a : A$, as in the following examples.

The following D^1 grammar does indeed generate the double copy language:

$$S = (P \odot_2 I) \odot_1 I$$

$a : A$	$a : J\backslash(A\backslash(J\backslash(A\backslash P)))$	$a : J\backslash(P \downarrow_2 Q)$	$a : J\backslash(Q \downarrow_1 (A\backslash P))$
$b : B$	$b : J\backslash(B\backslash(J\backslash(B\backslash P)))$	$b : J\backslash(P \downarrow_2 R)$	$b : J\backslash(R \downarrow_1 (B\backslash P))$

The following 1-$\mathbf{D}_J$ grammar also generates the double copy language:

$$S = (((P \uparrow X_2^P) \odot I) \uparrow X_1^P) \odot I$$

$a : A$	$b : B$	$x_1^P : X_1^P$	$x_2^P : X_2^P$
$x_1^Q : X_1^Q$	$x_2^Q : X_2^Q$	$x_1^R : X_1^R$	$x_2^R : X_2^R$
$a : X_2^P \backslash (A \backslash (X_1^P \backslash (A \backslash P)))$		$b : X_2^P \backslash (B \backslash (X_1^P \backslash (B \backslash P)))$	
$a : X_2^P \backslash (A \backslash (X_1^Q \backslash (A \backslash P)))$		$b : X_2^P \backslash (B \backslash (X_1^Q \backslash (B \backslash P)))$	
$a : X_2^P \backslash (A \backslash (X_1^R \backslash (A \backslash P)))$		$b : X_2^P \backslash (B \backslash (X_1^R \backslash (B \backslash P)))$	
$a : X_2^P \backslash ((P \uparrow X_2^P) \downarrow Q)$		$b : X_2^P \backslash ((P \uparrow X_2^P) \downarrow R)$	
$a : X_2^Q \backslash ((P \uparrow X_2^P) \downarrow Q)$		$b : X_2^R \backslash ((P \uparrow X_2^P) \downarrow R)$	
$a : X_1^Q \backslash ((Q \uparrow X_1^Q) \downarrow (A \backslash P))$		$b : X_1^Q \backslash ((R \uparrow X_1^R) \downarrow (B \backslash P))$	
$a : X_1^R \backslash ((Q \uparrow X_1^Q) \downarrow (A \backslash P))$		$b : X_1^R \backslash ((R \uparrow X_1^R) \downarrow (B \backslash P))$	
$a : X_1^P \backslash ((Q \uparrow X_1^Q) \downarrow (A \backslash P))$		$b : X_1^P \backslash ((R \uparrow X_1^R) \downarrow (B \backslash P))$	

Derivations in any fragment of the Displacement Calculus may be denoted by gluing together the natural deduction rules. This is precisely what is done in Figure 1, where we provide derivations for $bababa : S$ — in each of the two given grammars — to illustrate the conceptual similarity between the two systems.

To illustrate the — from the viewpoint of semantics — interesting property of allowing *hypothetical reasoning*, we notice the numerous examples given for phenomena like *quantification* and *appositive relativization* using type assignments with the type $(A \uparrow B) \downarrow A$ in it (Morrill et al. 2011). These type assignments require hypothetical reasoning (i.e. using the $\uparrow_k I$ rule) to account for different semantic readings (for example, the different readings of *Everybody loves someone.*). The point is, of course, that this hypothetical reasoning is available in the system 1-$\mathbf{D}_J$ while it is not available in $\mathbf{D}^1$, indicating that the former system would be favourable over the latter one. However, the system $\mathbf{D}^1$ already has a known generative capacity, whereas 1-$\mathbf{D}_J$ does not, as the following section shows.

3. Related Results

In this section, we recall the result of Sorokin (2013) combined with the result of Wijnholds (2011) to obtain an equivalence between DCFG and first-order DG. Moreover, this equivalence is *dimension-specific* in the sense that every k-DCFG corresponds to a first-order k-DG and vice versa.

As is usual in formal language theory, the notation X^i for some $i \in \mathbb{N}$ means i copies of X, i.e. $X^1 = X$, $X^{n+1} = X^n X$.

Definition 7 (Greibach Normal Form) *A k-DCFG is said to be in modified Greibach Normal Form if all of its rules are of one of the following forms:*

1. $A \to X^i a$ *where* $A \in N - \{X\}$,
2. $A \to X^i aB$ *or* $A \to BX^i a$ *where* $A \in N - \{X\}$,$B \in N$,
3. $A \to BX^i aC$ *where* $A \in N - \{X\}$,$B, C \in N$,
4. $A \to B +_j (X^i aC)$ *or* $A \to B +_1 (X^i a)$ *where* $A, B \in N - \{X\}$,$C \in N$,
5. $A \to BX^i$ *or* $A \to X^i B$ *where* $A, B \in N$,
6. $S \to \epsilon$ *or* $X \to 1$.

where $a \in \Sigma$ and $i, j \in \mathbb{N}$ and X is a distinguished non-terminal that generates 1s.

We then get the following lemma, due to Sorokin (2013):

Lemma 8 *Every k-DCFG is equivalent to some k-DCFG in modified Greibach Normal Form.*

To properly understand Sorokin's construction and our subsequent proof, we need to characterize the set of *simple types* and define the *head* of such a simple type. Simple types are characterized by the following grammar:

$$\begin{array}{lcl} T_1 & := & A \mid A\backslash T_1 \mid T_1/A \mid J\backslash T_1 \mid T_1/J \mid A \downarrow_k T_2 \\ T_2 & := & A \mid A\backslash T_2 \mid T_2/A \mid J\backslash T_2 \mid T_2/J \\ & & \text{where } A \text{ is atomic and } k \in \mathbb{N}. \end{array}$$

We define the head of a simple type inductively: $hd(A) = A$ for A atomic, $hd(A\backslash B) = hd(B/A) = hd(J\backslash B) = hd(B/J) = hd(B)$ and $hd(A \downarrow_k B) = hd(B)$.

We then get the main theorem of Sorokin's paper (Sorokin 2013):

Theorem 9 *Every k-DCFG is equivalent to some first-order k-DG.*

We give the construction of Sorokin because we will use it explicitly in our following proofs. So, given a k-DCFG $G = \langle \Sigma, N, P, S \rangle$ in modified Greibach Normal Form, we define a k-DG $G' = \langle \Sigma', \delta, S' \rangle$ over N (where the sorts of the basic types are exactly the ranks of the non-terminals) as follows:

1. $\Sigma' = \Sigma$
2. $S' = S$
3. δ is constructed as follows:
   (a) For every rule $A \to X^i a$, we add $(a, J_1\backslash(...(J_i\backslash A)))$ to δ,
   (b) For every rule $A \to X^i aB$ or $A \to BX^i a$, we respectively add $(a, J_1\backslash(...(J_i\backslash(A/B))))$ or $(a, J_1\backslash(...(J_i\backslash(B\backslash A))))$ to δ,
   (c) For every rule $A \to BX^i aC$ we add $(a, J_1\backslash(...(J_i\backslash(B\backslash(A/C)))))$ to δ,
   (d) For every rule $A \to B +_j (X^i aC)$ or $A \to B +_1 (X^i a)$ we respectively add $(a, J_1\backslash(...(J_i\backslash((B \downarrow_k A)/C))))$ or $(a, J_1\backslash(...(J_i\backslash(B \downarrow_1 A))))$ to δ,
   (e) For every rule $A \to BX^i$ or $A \to X^i B$, we add for every assignment $(c, C) \in \delta$ such that $hd(C) = B$, the assignment (c, C') where C' is C but with the head replaced by $(J_1\backslash(...(J_i\backslash A)))$ or $(((A/J_i)...)/J_1)$ respectively.

It is not hard to see that the type lexicon that comes out of Sorokin's construction has only (first-order) simple types.

To obtain the equivalence, we notify the reader of the following theorem, which is due to Wijnholds (2011):

Theorem 10 *Every first-order k-DG is equivalent to some k-DCFG.*

The proof of this theorem amounts to a staged construction in which one first adds rules $R^A \to a$ (meaning a rule R *labelled* with the type A) for every $(a, A) \in \delta$ and in each stage decomposing the types until finally, one reaches a fixed point and the grammar is complete. For a detailed exposition of the proof, we refer the reader to Wijnholds (2011, Lemma 6).

4. Main Result

In this section we show our main result: For every Displacement Context Free Grammar there is an equivalent restricted two-dimensional Displacement Grammar. We rely on the fact that Sorokin's conversion results in a first-order Displacement Grammar with an atomic start type and with only *simple types* in its lexicon. We will use this fact to convert these kind of grammars into restricted two-dimensional ones.

Next to the head of a (simple) type, we define the left and right degree[2] of a simple type as follows: $lDeg(A) = 0$ for A atomic, $lDeg(A\backslash B) = s(A) + lDeg(B)$, $lDeg(J\backslash B) = 1 + lDeg(B)$, $lDeg(B/A) = lDeg(B/J) = lDeg(B)$, $lDeg(A \downarrow_k B) = (k-1) + lDeg(B)$ and $rDeg(A) = 0$ for A atomic, $rDeg(A\backslash B) = rDeg(J\backslash B) = rDeg(B)$, $rDeg(B/A) = s(A) + rDeg(B)$, $rDeg(B/J) = 1 + rDeg(B)$, $rDeg(A \downarrow_k B) = (s(A) - k) + rDeg(B)$.

If l is a list and i an index, we denote by $len(l)$ the length of the list and by $l!i$ the $(i+1)$th element of the list. We furthermore denote by $l(i,j)$ the sublist of l from and including the ith element up to and including the jth element. Finally, we denote by $l_1 + l_2$ the list concatenation of lists l_1 and l_2. We define the following *conversion map* of simple types, given a list of basic types:

$$
\begin{array}{lcll}
C(A,l) & = & A & \text{for } A \text{ atomic,} \\
C(A\backslash B,l) & = & A\backslash C(B,l) & \\
C(J\backslash B,l) & = & (l!i)\backslash C(B,l) & \text{where } i = lDeg(B) \\
C(B/A,l) & = & C(B,l)/A & \\
C(B/J,l) & = & C(B,l)/(l!i) & \text{where } i = len(l) - rDeg(B) - 1 \\
C(A \downarrow_k B,l) & = & (A \uparrow X_k^A) \downarrow C(B,l) &
\end{array}
$$

We proceed to define a map[3] that takes as input a lexical entry, a lexical assignment, and a list (containing basic types) and returns a list of new lexical entries without separator unit J in it:

$$F((a,A),\delta,l_1) = \{(a, C(A,l_1))\} \cup \bigcup\{F(e,\delta,l_2) \mid (e,l_2) \in H(A,\delta,l_1)\}$$

where

$$
\begin{array}{lcl}
H(A,\delta,l) & = & \emptyset \quad \text{for } A \text{ atomic} \\
H(A\backslash B,\delta,l) & = & \{((c,C),l(i,j)) \mid (c,C) \in \delta, hd(C) = A\} \cup H(B,\delta,l) \\
 & & \text{where } i = lDeg(B) + 1 \text{ and } j = i + s(A) - 1 \\
H(J\backslash B,\delta,l) & = & H(B,\delta,l), \\
H(B/A,\delta,l) & = & \{((c,C),l(i,j)) \mid (c,C) \in \delta, hd(C) = A\} \cup H(B,\delta,l) \\
 & & \text{where } i = len(l) - rDeg(B) \text{ and } j = i + s(A) - 1 \\
H(B/J,\delta,l) & = & H(B,\delta,l) \\
H(A \downarrow_k B,\delta,l) & = & \begin{cases} \{((c,C),l') \mid (c,C) \in \delta, hd(C) = A\} \cup H(B,\delta,l) & \text{if } s(A) > s(B) \\ \{((c,C),l'') \mid (c,C) \in \delta, hd(C) = A\} \cup H(B,\delta,l) & \textit{otherwise} \end{cases}
\end{array}
$$

where $l' = l(1,k-1) + [X_k^A] + l(k, len(l))$ and l'' is $l(1,s(A))$ but with the kth element replaced by X_k^A.

We then define $G((a,A),\delta)$ as the fixed point of $F((a,A),\delta,[X_1^{hd(A)},\ldots,X_{s(hd(A))}^{hd(A)}])$.

Definition 11 *Let $G = \langle \Sigma, \delta, S\rangle$ be a $\mathbf{D}^1$ grammar such that S is atomic and δ contains only simple types. We construct a $\mathbf{D}_J$ grammar $G' = \langle \Sigma', \delta', S'\rangle$ over Tp' as follows:*

1. *For each $A \in Tp$ of sort n, we introduce $A_{var} = \{x_i^A \mid 1 \le i \le n\}$ and $A_{type} = \{X_i^A \mid 1 \le i \le n\}$ (where each x_i^A is not in Σ and each X_i^A is not in Tp),*

2. The left (right) degree measures how many separators are yet to be inserted on the left (resp. right).
3. For a Haskell implementation of the whole conversion algorithm, see `https://github.com/Holdwin/clin/blob/master/clin2`

2. $\Sigma' = \Sigma \cup \bigcup_{A \in Tp} A_{var}$,

3. $Tp' = Tp \cup \bigcup_{A \in Tp} A_{type}$,

4. *Given $A \in Tp'$, if $A \in Tp$ with $s(A) = 0$, then A also has sort 0 in Tp', otherwise the sort of A in Tp' is 1,*

5. $\delta' = \{(x_i^A, X_i^A) \mid X_i^A \in A_{type}, x_i^A \in A_{var}\} \cup \bigcup \{G(e, \delta) \mid e \in \delta\}$

6. $S' = S$.

Before we prove correctness of the construction, let us note that the newly constructed lexicon is a subset of the lexicon one would get when inserting all possible separator variables for J types. This means that the construction is always bounded in space by the size of the lexicon times the dimension of the grammar. Thus, we have that the construction does not give exponential blowup in the size of the grammar.

Now we only need to define one more notion, that of a *principal leaf* of a derivation tree. Let t be a derivation tree in a first-order Displacement Grammar G. If t just consists of a leaf l holding a lexical entry, then l is the principal leaf of t. Otherwise, the principal leaf of t is the first lexical entry encountered when following the most complex formula from the root of t. That is, if t is made up of a deduction rule having two subtrees t_1 and t_2 then the roots of t_1 and t_2 are labelled with a type. One of them must have an atomic type, and so we consider the principal leaf of the other subtree to be the principal leaf of t.

Lemma 12 *Let $G = \langle \Sigma, \delta, S \rangle$ be a $\mathbf{D}^1$ grammar over Tp (such that S is atomic and δ contains only simple types) and let $G' = \langle \Sigma', \delta', S' \rangle$ over Tp' be the corresponding $\mathbf{D}_J$ grammar. Then $L(G) = L(G')$.*

Proof We show that for every derivation $w_1 \cdot 1 \cdot ... \cdot 1 \cdot w_{n+1} : A$ (where A is atomic, $s(A) = n$ and for each w_i for $1 \leq i \leq n+1$ we have $r(w_i) = 0$) in G there exists a derivation $w_1 \cdot x_1^A \cdot ... x_n^A \cdot w_{n+1} : A$ and vice versa. As $s(S) = 0$ and $S' = G(S) = S$ we get that for every $w \in L(G)$ we have that $w \in L(G')$ and vice versa.

($\Rightarrow$) We show that for every derivation tree in G for some $w_1 \cdot 1 \cdot ... \cdot 1 \cdot w_{n+1} : A$ there exists a derivation tree in G' for $w_1 \cdot x_1^A \cdot ... x_n^A \cdot w_{n+1} : A$. So, let t be a derivation tree in G for $w = w_1 \cdot 1 \cdot ... \cdot 1 \cdot w_{n+1} : A$. We proceed by recursing over the map G: Consider the *principal leaf* of t holding $b : B$. As $b : B$ is a lexical entry (and $hd(B) = A$), we have that $(b : C(B, [X_1^A, ..., X_{s(A)}^A]))$ is in δ'. We can replace $b : B$ by exactly this new lexical entry, thereby enforcing to change the types and derived strings in the rest of the tree (in particular, the final derived string will be transformed into $w_1 \cdot x_1^A \cdot ... x_n^A \cdot w_{n+1}$). Then one can replace every $1 : J$ on the way up in the tree by the corresponding separator variable assignment. Moreover, for each subtree t' such that its root is used in the derivation of $w : A$ from the entry $b : B$, let $v = v_1 \cdot 1 \cdot ... \cdot 1 \cdot v_{m+1} : C$ (where $s(C) = m$) be the element at the root of t' and let D be the subformula of B such that it used to deduce together with C a some new type.

Then, if $D = C \backslash E$, then we should replace v by $v_1 \cdot X_i^A \cdot ... \cdot X_{i+m}^A v_{m+1}$ (where $i = lDeg(E)$). But then we can use the recursive definition of F to claim that the principal leaf of t' holds a lexical entry $f : F$ such that $(c, [X_i^A, ..., X_{i+k}^A]) \in H(A, \delta, [X_1^A, ..., X_{s(A)}^A])$, and repeat the process. The argument is exactly similar if $D = E/C$. When $D = C \uparrow_k E$, then either $s(C) = s(E) + 1$ or $s(C) < s(E)$. When $s(C) = s(E) + 1$ we should replace v by $v_1 \cdot X_1^A \cdot ... \cdot X_k^C \cdot ... \cdot X_m^A \cdot v_{m+1}$ (and of course, insert an instance of the $I \uparrow$ rule to get rid of X_k^C). But in this case, the definition of H gives us exactly the right list to apply the conversion map C to for the lexical entry in the principal leaf of t'. A same argument goes for the case where $s(C) < s(E)$. So, recursively applying the map G gives us

a new derivation tree in G' for $w_1 \cdot X_1^A \cdot ... \cdot X_{s(A)}^A \cdot w_{s(A)+1} : A$. As $s(S) = 0$, this means that for every $w \in L(G)$ we obviously have a proof of $w \in L(G')$.

($\Leftarrow$) We show that for every derivation tree in G' for some $w \in L(G')$ there exists a derivation tree in G for $w \in L(G)$. So, let t be a derivation tree for some $w \in L(G')$. As for each introduced separator variable, there was a J basic type in its place in the original grammar, we can map the following function over the tree to obtain a legitimate tree in G for w showing that $w \in L(G)$: $f(v : A) = g(v) : h(A)$ where g is the following string homomorphism: $g(a) = a$ if $a \in \Sigma$ and $g(a) = 1$ otherwise (i.e. g maps separator variables to separators and other letters to itself) and where h maps types *back* to their original type:

$$h(A) = A \text{ if } A \in Tp,\ h(A) = J \text{ otherwise},$$
$$h(A\backslash B) = h(A)\backslash h(B),$$
$$h(B/A) = h(B)/h(A),$$
$$h((A \uparrow X_k^A) \downarrow B) = A \downarrow_k h(B)$$

It should be clear that mapping this over t and then conflating the pieces in the derivation where the $I \uparrow$ is used gives a correct derivation in G as each of the leaves (representing a lexical entry) is mapped either to $1 : J$ or to a lexical entry of G. Furthermore, the string homomorphism g maps w to itself (as w cannot contain separator variables). Thus we have shown $L(G') \subseteq L(G)$.

Theorem 13 *For every Displacement Context-Free Grammar, there is an equivalent restricted two-dimensional Displacement Grammar.*

5. Conclusion & Future Directions

We have shown that, besides the already known equivalence result between the frameworks of well-nested Multiple Context Free Grammar and first-order Displacement Calculus, we can simulate the use of separators by means of separator variables and thus have obtained that well-nested Multiple Context Free Grammars are included in restricted two-dimensional Displacement Grammars. We state here the conjecture that the converse is also true: every restricted two-dimensional Displacement Grammar can be converted to a first-order Displacement Grammar, and thus to a well-nested Multiple Context Free Grammar. If this conjecture were to be proven true, one would obtain a second logical characterization of the Mildly Context-Sensitive Languages. If the conjecture were to be given by a polynomial conversion, we would also obtain polynomial parsability for restricted two-dimensional Displacement Grammars. Furthermore, it would be interesting to see how far one can go when employing the general Displacement Calculus, e.g. whether its generative capacity is equal to, or goes beyond that of general Multiple Context-Free Grammar. Some clues are given, for instance that the general Displacement Calculus generates the permutation closures of context-free languages (see Morrill and Valentín (2010)), and that $MIX_3 = \{w \in \{a, b, c\}^+ \mid |w|_a = |w|_b = |w|_c\}$ (which is the permutation closure of a context-free language) can be generated by a (non-well-nested) 2-MCFG (Salvati 2011).

As for linguistic applications of the systems $\mathbf{D}^1$ and 1-$\mathbf{D}_J$, it is clear that the former system does not preserve the desirable property of hypothetical reasoning, while the latter keeps it (in a restricted form). Thus, for most of the semantic phenomena addressed by Morrill et al. (2011, Chapter 3) one will want to opt for the system 1-$\mathbf{D}_J$. For instance, the approach given there to quantification leads to a promising semantic analysis of the words *every* and *some*, which could probably not be treated as elegantly within the system $\mathbf{D}^1$.

References

Huybregts, Riny (1984), The weak inadequacy of context-free phrase structure grammars, *in* de Haan, Ger, Mieke Trommelen, and Wim Zonneveld, editors, *Van periferie naar kern*, Foris, Dordrecht, pp. 81–99.

Joshi, Aravind K., K. Vijay Shanker, and David Weir (1990), The convergence of mildly context-sensitive grammar formalisms.

Kallmeyer, Laura (2010), *Parsing Beyond Context-Free Grammars*, 1st ed., Springer Publishing Company, Incorporated.

Kanazawa, Makoto (2009), The pumping lemma for well-nested multiple context-free languages, *Developments in Language Theory*, Springer, pp. 312–325.

Morrill, Glyn and Oriol Valentín (2010), On calculus of displacement, *Proceedings of the 10th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pp. 45–52.

Morrill, Glyn, Oriol Valentín, and Mario Fadda (2011), The displacement calculus, *Journal of Logic, Language and Information* **20** (1), pp. 1–48, Springer.

Salvati, Sylvain (2011), MIX is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is solved by a third-order collapsible pushdown automaton, *Rapport de recherche.* http://hal.inria.fr/inria-00564552.

Shieber, Stuart M (1987), Evidence against the context-freeness of natural language, *The Formal complexity of natural language*, Springer, pp. 320–334.

Sorokin, Alexey (2013), Normal forms for multiple context-free languages and displacement lambek grammars, *Logical Foundations of Computer Science*, Springer, pp. 319–334.

Wijnholds, G.J. (2011), Investigations into categorial grammar: Symmetric pregroup grammar and displacement calculus. http://dspace.library.uu.nl/handle/1874/207634.

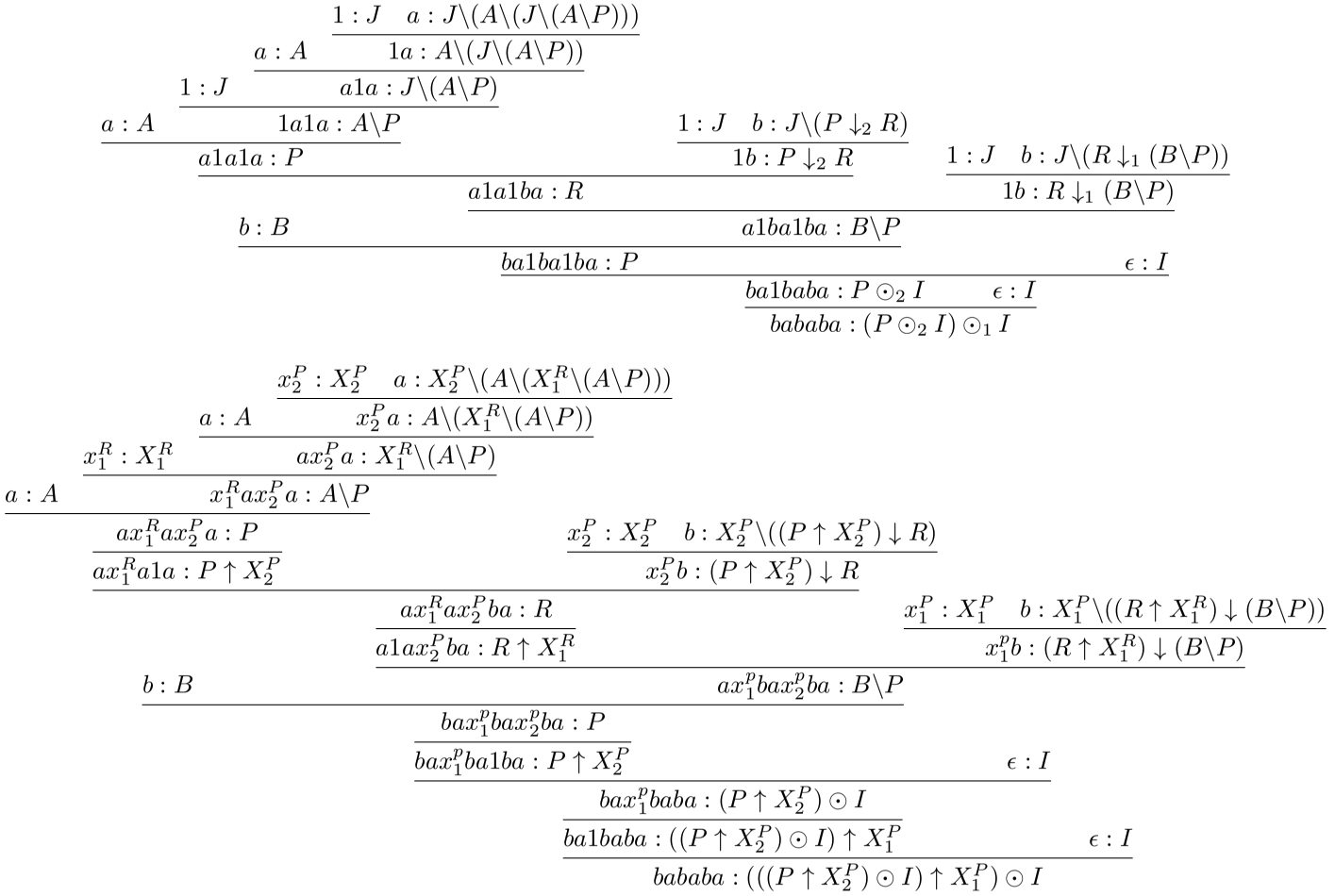

Figure 1: Derivations for $bababa : S$