

Journal of Language Modelling

VOLUME 5 ISSUE 1
JUNE 2017

Editorials

Finite-state methods and mathematics of language.

Introduction to the special issue 1

Marco Kuhlmann, Christian Wurm

Articles

Chomsky-Schützenberger parsing

for weighted multiple context-free languages 3

Tobias Denking

Relative clauses as a benchmark for Minimalist parsing 57

Thomas Graf, James Monette, Chong Zhang

Rewrite rule grammars with multitape automata 107

Mans Hulden

A probabilistic model of Ancient Egyptian writing 131

Mark-Jan Nederhof, Fahrurrozi Rahman



JOURNAL OF
LANGUAGE MODELLING

ISSN 2299-8470 (electronic version)

ISSN 2299-856X (printed version)

<http://jlm.ipipan.waw.pl/>

MANAGING EDITOR

Adam Przepiórkowski IPI PAN

GUEST EDITORS OF THIS SPECIAL ISSUE

Marco Kuhlmann Linköping University

Christian Wurm University of Düsseldorf

SECTION EDITORS

Elżbieta Hajnicz IPI PAN

Agnieszka Mykowiecka IPI PAN

Marcin Woliński IPI PAN

STATISTICS EDITOR

Łukasz Dębowski IPI PAN



Published by IPI PAN

Institute of Computer Science, Polish Academy of Sciences
ul. Jana Kazimierza 5, 01-248 Warszawa, Poland

Circulation: 100 + print on demand

Layout designed by Adam Twardoch.

Typeset in X_YL^AT_EX using the typefaces: *Playfair Display*
by Claus Eggers Sørensen, *Charis SIL* by SIL International,
JLM monogram by Łukasz Dziedzic.

*All content is licensed under
the Creative Commons Attribution 3.0 Unported License.*
<http://creativecommons.org/licenses/by/3.0/>



EDITORIAL BOARD

Steven Abney University of Michigan, USA

Ash Asudeh Carleton University, CANADA;
University of Oxford, UNITED KINGDOM

Chris Biemann Technische Universität Darmstadt, GERMANY

Igor Boguslavsky Technical University of Madrid, SPAIN;
Institute for Information Transmission Problems,
Russian Academy of Sciences, Moscow, RUSSIA

António Branco University of Lisbon, PORTUGAL

David Chiang University of Southern California, Los Angeles, USA

Greville Corbett University of Surrey, UNITED KINGDOM

Dan Cristea University of Iași, ROMANIA

Jan Daciuk Gdańsk University of Technology, POLAND

Mary Dalrymple University of Oxford, UNITED KINGDOM

Darja Fišer University of Ljubljana, SLOVENIA

Anette Frank Universität Heidelberg, GERMANY

Claire Gardent CNRS/LORIA, Nancy, FRANCE

Jonathan Ginzburg Université Paris-Diderot, FRANCE

Stefan Th. Gries University of California, Santa Barbara, USA

Heiki-Jaan Kaalep University of Tartu, ESTONIA

Laura Kallmeyer Heinrich-Heine-Universität Düsseldorf, GERMANY

Jong-Bok Kim Kyung Hee University, Seoul, KOREA

Kimmo Koskenniemi University of Helsinki, FINLAND

Jonas Kuhn Universität Stuttgart, GERMANY

Alessandro Lenci University of Pisa, ITALY

Ján Mačutek Comenius University in Bratislava, SLOVAKIA

Igor Mel'čuk University of Montreal, CANADA

Glyn Morrill Technical University of Catalonia, Barcelona, SPAIN

Stefan Müller Freie Universität Berlin, GERMANY

Mark-Jan Nederhof University of St Andrews, UNITED KINGDOM

Petya Osenova Sofia University, BULGARIA

David Pesetsky Massachusetts Institute of Technology, USA

Maciej Piasecki Wrocław University of Technology, POLAND

Christopher Potts Stanford University, USA

Louisa Sadler University of Essex, UNITED KINGDOM

Agata Savary Université François Rabelais Tours, FRANCE

Sabine Schulte im Walde Universität Stuttgart, GERMANY

Stuart M. Shieber Harvard University, USA

Mark Steedman University of Edinburgh, UNITED KINGDOM

Stan Szpakowicz School of Electrical Engineering
and Computer Science, University of Ottawa, CANADA

Shravan Vasishth Universität Potsdam, GERMANY

Zygmunt Vetulani Adam Mickiewicz University, Poznań, POLAND

Aline Villavicencio Federal University of Rio Grande do Sul,
Porto Alegre, BRAZIL

Veronika Vincze University of Szeged, HUNGARY

Yorick Wilks Florida Institute of Human and Machine Cognition, USA

Shuly Wintner University of Haifa, ISRAEL

Zdeněk Žabokrtský Charles University in Prague, CZECH REPUBLIC

Finite-state methods and mathematics of language. Introduction to the special issue

*Marco Kuhlmann*¹ and *Christian Wurm*²

¹ Department of Computer and Information Science
Linköping University, Sweden

² Department of Computational Linguistics
University of Düsseldorf, Germany

There is a long and fertile interaction between research on finite-state methods and the mathematics of language: many central results in mathematical linguistics are based on finite-state models such as automata and grammars, and mathematical linguistics in turn has laid the foundations for the application of finite-state methods in natural language processing (NLP). One important outcome of the cross-semination between the two fields is the characterisation of adequate classes of string languages and tree languages for linguistic modelling.

Our intention with this special issue is to highlight current work in the intersection between mathematics of language and finite-state methods, as presented at two premier conferences in the respective fields: the 12th International Conference on Finite-State Methods and Natural Language Processing (FSM/NLP), which was held 22–24 June 2015 in Düsseldorf, Germany; and the 14th Meeting on Mathematics of Language (MOL), which was held 25–26 July 2015 in Chicago, USA. To this end we invited the authors of the two conferences to submit revised and extended versions of their contributions, which were then subjected to an entirely new peer-review process – something that would have been impossible without the dedication and thorough work of our reviewers, to whom we owe our sincere gratitude.

At the end of the peer-review process, we selected four submissions for publication in this special issue:

“Chomsky–Schützenberger parsing for weighted multiple context-free languages” by Tobias Denkinger generalises the well-known char-

acterisation of context-free languages as the homomorphic image of the intersection of a Dyck language and a regular language to an expressive class of weighted languages, and then uses this characterisation to derive a parsing algorithm.

“Relative clauses as a benchmark for Minimalist parsing” by Thomas Graf, James Monette, and Chong Zhang presents a careful and comprehensive evaluation of a large number of complexity metrics that have been proposed to relate parsing difficulty to memory usage. The results show that only a handful of these metrics can explain observed contrasts in human sentence processing.

“Rewrite rule grammars with multitape automata” by Mans Hulden addresses the following problem: relation composition is one of the most frequently used methods in finite-state approaches; in particular, it allows to construct complex transformations out of simpler ones via intermediate steps, which then are discarded. This discarding is not desirable in some applications, such as the reconstruction of old languages. However, if one does not discard intermediate steps, then relations become more than binary, which is a problem for existing program libraries. The article addresses this problem both from a theoretical and practical point of view, by encoding arbitrary tuples as simple strings, hence relations as languages.

“A probabilistic model of Ancient Egyptian writing” by Mark-Jan Nederhof and Fahrurrozi Rahman provides a formal model for the transliteration of hieroglyphic writing. Ancient Egyptian writing is particularly complex, because the same hieroglyph can have many different functions: it can have (among other) a semantic content, a phonological content, or just be used to specify the semantic or phonological content of some other hieroglyph (both redundantly or not). The authors approach this extremely complex system by introducing “sign functions”, which go beyond the power of finite-state machines and lay the foundation for “machine transliteration” of Ancient Egyptian writing.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.

<http://creativecommons.org/licenses/by/3.0/>



Chomsky-Schützenberger parsing for weighted multiple context-free languages

Tobias Denking
Faculty of Computer Science
Technische Universität Dresden
Germany

ABSTRACT

We prove a Chomsky-Schützenberger representation theorem for multiple context-free languages weighted over complete commutative strong bimonoids. Using this representation we devise a parsing algorithm for a restricted form of those devices.¹

Keywords:
Chomsky-Schützenberger, parsing, multiple context-free grammars, linear context-free rewriting systems

1

INTRODUCTION

Mildly context-sensitive languages receive much attention in the natural language processing community since they are able to express non-projective constituents (Maier and Søgaard 2008) and non-projective dependencies (Kuhlmann and Satta 2009). Figure 1 shows an example of a non-projective constituency tree. Figure 2 shows an example of a non-projective dependency tree. Non-projectivity is evident from crossings of edges, highlighted by circles. The phenomenon of non-projectivity occurs frequently in natural language corpora, e.g. about 28 percent of all sentences in both the NeGra corpus² and the TIGER corpus (Brants *et al.* 2004) contain non-projective constituents

¹ The CS parser for weighted MCFL (Sections 5 and 6) is original to this work. Sections 2 to 4 are a substantially revised version of Denking (2015).

² <http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/>

Figure 1:
A non-projective
constituency tree (taken
from Maier and Søgaard
2008, Figure 3)

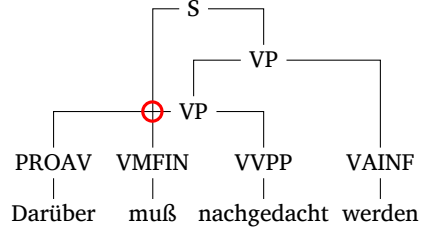
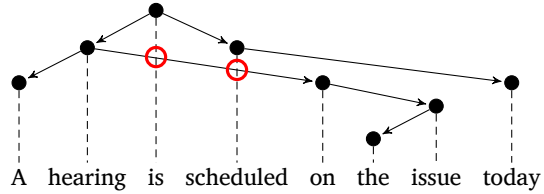


Figure 2:
A non-projective
dependency tree (taken
from Kuhlmann and Satta
2009, Figure 2)



(Maier and Søgaard 2008, Table 6) and about 23 percent of all sentences in the Prague Dependency Treebank³ contain non-projective dependencies (Kuhlmann and Satta 2009, Section 1).

Multiple context-free grammars describe the language classes induced by many mildly context-sensitive grammar formalisms, e.g. head grammars (Seki *et al.* 1991), linear context-free rewriting systems (Seki *et al.* 1991), combinatory categorial grammars (Vijay-Shanker *et al.* 1986; Weir and Joshi 1988), linear indexed grammars (Vijay-Shanker 1988), minimalist grammars (Michaelis 2001b,a), and finite-copying lexical functional grammars (Seki *et al.* 1993).

Parsing, i.e. the annotation of a sentence with syntactic structure, is one of the main concerns of natural language processing. Many parsing approaches are known for multiple context-free grammars, e.g.

- Cocke-Younger-Kasami-style parsing (Seki *et al.* 1991, Procedure MEMBER; and Burden and Ljunglöf 2005, Section 3),
- guided parsing (Barthélemy *et al.* 2001; Burden and Ljunglöf 2005, Section 4; and van Cranenburgh 2012),
- active parsing (Burden and Ljunglöf 2005, Section 5),
- incremental parsing (Villemonte de la Clergerie 2002; Burden and Ljunglöf 2005, Section 6; and Angelov 2009), and
- LR-style parsing (Kallmeyer and Maier 2015).

³https://ufal.mff.cuni.cz/pdt/Corpora/PDT_1.0/

The Chomsky-Schützenberger (CS) representation for context-free languages (Chomsky and Schützenberger 1963, Proposition 2) has been generalised to a variety of unweighted and weighted settings, e.g. context-free languages weighted with commutative semirings (Salomaa and Soittola 1978, Theorem 4.5), tree adjoining languages (Weir 1988, Lemma 3.5.2), multiple context-free languages (Yoshinaka *et al.* 2010, Theorem 3), context-free languages weighted with unital valuation monoids (Droste and Vogler 2013, Theorem 2), yields of simple context-free tree languages (Kanazawa 2014, Theorem 8.3), indexed languages (Duske *et al.* 1979, Theorems 1 and 2; Fratani and Voundy 2015, Theorem 4; and Fratani and Voundy 2016, Theorem 18), and automata with storage weighted with unital valuation monoids (Herrmann and Vogler 2015, Theorem 11).

We give a generalisation to the case of multiple context-free languages weighted with a complete commutative strong bimonoid and apply it to devise a parsing algorithm. Sections 3 to 5 contain the main contributions of this article.

- In Section 3 we provide a CS representation for weighted multiple context-free languages by means of a modular proof that first separates the weights from the given grammar and then employs the result for the unweighted case (using the same overall idea as in Droste and Vogler 2013).
- In Section 4 we give a more algebraic variant of multiple Dyck languages using congruence relations together with a decision algorithm for membership that is strongly related to those congruence relations. Also we show that congruence multiple Dyck languages can be used to develop a CS representation of weighted MCFLs.
- Using the CS representation based on congruence multiple Dyck languages and given a partial order on the weights, we derive a parsing algorithm (Section 5) similar to the one described by Hulden (2011). It employs a regular language and a weight function to generate output that is then filtered by an acceptor for a specific congruence multiple Dyck language in order to obtain the best derivations with respect to the partial order.
- The idea behind the Chomsky-Schützenberger parser presented in Section 5 is similar to already established approaches. Section 6 relates the known approaches to the one presented in this article.

Since our proofs do not require distributivity, we can be slightly more general than complete commutative semirings: We consider complete commutative strong bimonoids.

2

PRELIMINARIES

In this section we recall formalisms used in this article and fix some notation: We denote by \mathbb{N} the set of natural numbers (including zero), and by \mathbb{N}_+ the set $\mathbb{N} \setminus \{0\}$. For every $n \in \mathbb{N}$ we abbreviate $\{1, \dots, n\}$ by $[n]$. Let A be a set. The *power set* of A is denoted by $\mathcal{P}(A)$. Let B be a finite set. A *partition* of B is a set $\mathfrak{P} \subseteq \mathcal{P}(B)$ where the elements of \mathfrak{P} (called *cells*) are non-empty, pairwise disjoint, and $\bigcup_{p \in \mathfrak{P}} p = B$.

Let A and B be sets and $A' \subseteq A$. The *set of functions from A to B* is denoted by $A \rightarrow B$, we still write $f: A \rightarrow B$ rather than $f \in A \rightarrow B$. Let f be a function. The *domain and codomain of f* are denoted by $\text{dom}(f)$ and $\text{codom}(f)$, respectively. The *restriction of f to A'* , denoted by $f|_{A'}$, is a function from A' to B such that $f|_{A'}(a') = f(a')$ for every $a' \in A'$. Let g be a function such that $\text{codom}(f) \subseteq \text{dom}(g)$. We denote the function obtained by applying g *after* f by $g \circ f$. Let F be a set of functions and $B \subseteq \bigcap_{f \in F} \text{dom}(f)$. The set $\{f(B) \mid f \in F\} \subseteq \mathcal{P}(\bigcup_{f \in F} \text{codom}(f))$ is denoted by $F(B)$. Let G and H be sets of functions such that $\bigcap_{g \in G} \text{codom}(g) \subseteq \bigcap_{h \in H} \text{dom}(h)$. The set $\{h \circ g \mid h \in H, g \in G\}$ of functions is denoted by $H \circ G$.

Let A be a set and $\approx \subseteq A \times A$ a binary relation on A . We call \approx an *equivalence relation (on A)* if it is reflexive, symmetric, and transitive. Let $a \in A$ and \approx be an equivalence relation. The *equivalence class of a in \approx* , denoted by $[a]_{\approx}$, is $\{b \in A \mid a \approx b\}$. Let $f: A^k \rightarrow A$ be a function. We say that \approx *respects f* if for every $(a_1, b_1), \dots, (a_k, b_k) \in \approx$ holds $f(a_1, \dots, a_k) \approx f(b_1, \dots, b_k)$. Now let \mathcal{A} be an algebra with domain A . We call \approx a *congruence relation (on \mathcal{A})* if \approx is an equivalence relation on A and respects every operation of \mathcal{A} .

Let $\trianglelefteq \subseteq A \times A$ be a binary relation on A . We call \trianglelefteq a *partial order (on A)* if it is reflexive, antisymmetric, and transitive.

2.1

Sorts

We will use the concept of sorts to formalise restrictions on building terms (or trees), e.g. derivation trees or terms over functions. One can think of sorts as data types in a programming language: Every concrete

value has a sort (type) and every function requires its arguments to be of fixed sorts (types) and returns a value of some fixed sort (type).

Let S be a countable set (of sorts) and $s \in S$. An S -sorted set is a tuple (B, sort) where B is a set and sort is a function from B to S . We denote the preimage of s under sort by B_s and abbreviate (B, sort) by B ; sort will always be clear from the context. Let A be an $(S^* \times S)$ -sorted set. The set of terms over A , denoted by T_A , is the smallest S -sorted set T where $\xi = a(\xi_1, \dots, \xi_k) \in T_s$ if there are $s_1, \dots, s_k, s \in S$ such that $a \in A_{(s_1, \dots, s_k, s)}$ and $\xi_i \in T_{s_i}$ for every $i \in [k]$. Let $\xi = a(\xi_1, \dots, \xi_k) \in T_A$. The set of positions in ξ is defined as $\text{pos}(\xi) = \{\varepsilon\} \cup \{iu \mid i \in [k], u \in \text{pos}(\xi_i)\}$ and for every $\pi \in \text{pos}(\xi)$ the symbol in ξ at position π is defined as $\xi(\pi) = a$ if $\pi = \varepsilon$ and as $\xi(\pi) = \xi_i(u)$ if $\pi = iu$ for some $i \in [k]$ and $u \in \text{pos}(\xi_i)$.

2.2

Weight algebras

A monoid is an algebra $(\mathcal{A}, \cdot, 1)$ where \cdot is associative and 1 is neutral with respect to \cdot . A bimonoid is an algebra $(\mathcal{A}, +, \cdot, 0, 1)$ where $(\mathcal{A}, +, 0)$ and $(\mathcal{A}, \cdot, 1)$ are monoids. We call a bimonoid *strong* if $(\mathcal{A}, +, 0)$ is commutative and for every $a \in \mathcal{A}$ we have $0 \cdot a = 0 = a \cdot 0$. Intuitively, a strong bimonoid is a semiring without distributivity. A strong bimonoid is called *commutative* if $(\mathcal{A}, \cdot, 1)$ is commutative. A commutative strong bimonoid is *complete* if there is an infinitary sum operation \sum that maps every indexed family of elements of \mathcal{A} to \mathcal{A} , extends $+$, and satisfies infinitary associativity and commutativity laws (cf. Droste and Vogler 2013, Section 2), i.e. for every countable set I and every I -indexed family $a: I \rightarrow \mathcal{A}$ it holds that:

- (i) $\sum_{i \in \emptyset} a(i) = 0$;
- (ii) for every $j \in I: \sum_{i \in \{j\}} a(i) = a(j)$;
- (iii) for every $j, k \in I$ with $j \neq k: \sum_{i \in \{j, k\}} a(i) = a(j) + a(k)$; and
- (iv) for every countable set J and family $\mathcal{J}: J \rightarrow \mathcal{P}(I)$ with $I = \bigcup_{j \in J} \mathcal{J}(j)$ and for every $j, j' \in J$ with $j \neq j' \implies \mathcal{J}(j) \cap \mathcal{J}(j') = \emptyset$, we have $\sum_{j \in J} \sum_{i \in \mathcal{J}(j)} a(i) = \sum_{i \in I} a(i)$.

For the rest of this article let $(\mathcal{A}, +, \cdot, 0, 1)$, abbreviated by \mathcal{A} , be a complete commutative strong bimonoid.

Example 2.1. We provide a list of complete commutative strong bimonoids (cf. Droste *et al.* 2010, Example 1) some of which are relevant for natural language processing:

- any complete commutative semiring, e.g.
 - the *Boolean semiring* $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$,
 - the *probability semiring* $\text{Pr} = (\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$,
 - the *Viterbi semiring* $([0, 1], \max, \cdot, 0, 1)$,
 - the *tropical semiring* $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$,
 - the *arctic semiring* $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$,
- any complete lattice, e.g.
 - any non-empty finite lattice $(L, \vee, \wedge, 0, 1)$ where L is a non-empty finite set,
 - the lattice $(\mathcal{P}(A), \cup, \cap, \emptyset, A)$ where A is an arbitrary set,
 - the lattice $(\mathbb{N}, \text{lcm}, \text{gcd}, 1, 0)$,
- the *tropical bimonoid* $(\mathbb{R}_{\geq 0} \cup \{\infty\}, +, \min, 0, \infty)$,
- the *arctic bimonoid* $(\mathbb{R}_{\geq 0} \cup \{-\infty\}, +, \max, 0, -\infty)$, and
- the algebras $\text{Pr}_1 = ([0, 1], \oplus_1, \cdot, 0, 1)$ and $\text{Pr}_2 = ([0, 1], \oplus_2, \cdot, 0, 1)$ where $a \oplus_1 b = a + b - a \cdot b$ and $a \oplus_2 b = \min\{a + b, 1\}$ for every $a, b \in [0, 1]$.

where \mathbb{R} and $\mathbb{R}_{\geq 0}$ denote the set of reals and the set of non-negative reals, respectively; $+$, \cdot , \max , \min denote the usual operations; \vee , \wedge denote disjunction and conjunction, respectively, for the boolean semiring and join and meet, respectively, for any non-empty finite lattice; and lcm and gcd are binary functions that calculate the least common multiple and the greatest common divisor, respectively.

Also, there are some bimonoids that are interesting for natural language processing but are *not* complete commutative strong bimonoids. E.g.

- the *semiring of formal languages* $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ where Σ is an alphabet and \cdot is language concatenation, i.e. $L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$ for every $L_1, L_2 \subseteq \Sigma^*$; and
- the semiring $(\Sigma^* \cup \{\infty\}, \wedge, \cdot, \infty, \varepsilon)$ where Σ is an alphabet, \cdot is concatenation, \wedge calculates the longest common prefix of its arguments, and ∞ is a new element that is neutral with respect to \wedge and annihilating with respect to \cdot (cf. Mohri 2000).

None of the two examples is commutative. □

An \mathcal{A} -weighted language (over Δ) is a function $L: \Delta^* \rightarrow \mathcal{A}$. The support of L , denoted by $\text{supp}(L)$, is $\{w \in \Delta^* \mid L(w) \neq 0\}$. If $|\text{supp}(L)| \leq 1$, we call L a monomial. We write $\mu.w$ for L if $L(w) = \mu$ and for every $w' \in \Delta^* \setminus \{w\}$ we have $L(w') = 0$.

2.3 Recognisable languages

For the many known results concerning finite state automata and regular languages, we will rely on Hopcroft and Ullman (1969) and Hopcroft and Ullman (1979). We nevertheless recall the basic definitions:

Definition 2.2. A finite state automaton, for short: FSA, is a tuple $\mathcal{M} = (Q, \Delta, q_0, F, T)$ where Δ is an alphabet (terminals), Q is a finite set (states) disjoint from Δ , $q_0 \in Q$ (initial state), $F \subseteq Q$ (final states), and $T \subseteq Q \times \Delta^* \times Q$ is a finite set (transitions). \square

A run in \mathcal{M} is a string $\kappa \in (Q \cup \Delta)^*$ where for every substring of κ of the form quq' (for some $q, q' \in Q$ and $u \in \Delta^*$) we have that $(q, u, q') \in T$, the first symbol of κ is q_0 , and the last symbol of κ is in F . If a run κ contains a substring of the form quq' (for some $q, q' \in Q$ and $u \in \Delta^*$), we say that the transition (q, u, q') occurs in κ . The word corresponding to κ is obtained by removing the elements of Q from κ . The language of \mathcal{M} is denoted by $L(\mathcal{M})$. The set of recognisable languages, denoted by REC, is the set of languages L for which there is an FSA \mathcal{M} with $L = L(\mathcal{M})$.

2.4 Weighted string homomorphisms

Let Δ and Γ be alphabets and $g: \Delta \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ (i.e. a function that takes an element of Δ and returns a function that takes an element of Γ^* and returns an element of \mathcal{A}) such that $g(\delta)$ is a monomial for every $\delta \in \Delta$. We define $\widehat{g}: \Delta^* \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ where for every $k \in \mathbb{N}$, $\delta_1, \dots, \delta_k \in \Delta$, and $v \in \Gamma^*$ we have

$$\widehat{g}(\delta_1 \cdots \delta_k)(v) = \sum_{\substack{v_1, \dots, v_k \in \Gamma^* \\ v = v_1 \cdots v_k}} g(\delta_1)(v_1) \cdots g(\delta_k)(v_k).$$

We call \widehat{g} an \mathcal{A} -weighted (string) homomorphism. Since $g(\delta_1), \dots, g(\delta_k)$ are monomials for each $\delta_1, \dots, \delta_k \in \Delta$, there is at most one tuple $(v_1, \dots, v_k) \in (\Gamma^*)^k$ such that $g(\delta_i)(v_i) \neq 0$ for every $i \in [k]$. Hence, there is at most one $v \in \Gamma^*$ with $\widehat{g}(\delta_1 \cdots \delta_k)(v) \neq 0$ (namely $v = v_1 \cdots v_k$). Therefore, $\widehat{g}(u)$ is a monomial for every $u \in \Delta^*$. We call \widehat{g}

alphabetic if there is a function $h: \Delta \rightarrow (\Gamma \cup \{\varepsilon\} \rightarrow \mathcal{A})$ with $\widehat{g} = \widehat{h}$. If $\widehat{g}(u) = \mu.w$ (recall the end of Section 2.2) for $u \in \Delta^*$, then we will sometimes say “ \widehat{g} maps u to w ” (leaving out the weight μ) or “ \widehat{g} weights u with μ ” (leaving out the word w). Now assume that $\mathcal{A} = \mathbb{B}$ and we have $|\text{supp}(g(\delta))| = 1$ for every $\delta \in \Delta$. Then g can be construed as a function from Δ to Γ^* and \widehat{g} can be construed as a function from Δ^* to Γ^* . In this case we call \widehat{g} a *(string) homomorphism*. The sets of all \mathcal{A} -weighted homomorphisms, \mathcal{A} -weighted alphabetic homomorphisms, homomorphisms, and alphabetic homomorphisms are denoted by $\text{HOM}(\mathcal{A})$, $\alpha\text{HOM}(\mathcal{A})$, HOM , and αHOM , respectively.

2.5 Weighted multiple context-free languages

We fix a set $X = \{x_i^j \mid i, j \in \mathbb{N}_+\}$ of *variables*. Let Δ be an alphabet. The set of *composition representations over Δ* is the $(\mathbb{N}^* \times \mathbb{N})$ -sorted set RF_Δ where for every $s_1, \dots, s_\ell, s \in \mathbb{N}$ we define $X_{(s_1 \dots s_\ell, s)} = \{x_i^j \mid i \in [\ell], j \in [s_i]\} \subseteq X$ and $(\text{RF}_\Delta)_{(s_1 \dots s_\ell, s)}$ as the set that contains $[u_1, \dots, u_s]_{(s_1 \dots s_\ell, s)}$ for every $u_1, \dots, u_s \in (\Delta \cup X_{(s_1 \dots s_\ell, s)})^*$. We will often write X_f instead of $X_{(s_1 \dots s_\ell, s)}$. Let $f = [u_1, \dots, u_s]_{(s_1 \dots s_\ell, s)} \in \text{RF}_\Delta$. The *string function of f* , also denoted by f , is the function from $(\Delta^*)^{s_1} \times \dots \times (\Delta^*)^{s_\ell}$ to $(\Delta^*)^s$ such that $f((w_1^1, \dots, w_1^{s_1}), \dots, (w_\ell^1, \dots, w_\ell^{s_\ell})) = (u'_1, \dots, u'_s)$ where (u'_1, \dots, u'_s) is obtained from (u_1, \dots, u_s) by replacing each occurrence of x_i^j by w_i^j for every $i \in [\ell]$ and $j \in [s_i]$. The set of all string functions for some composition representation over Δ is denoted by F_Δ . From here on we no longer distinguish between composition representations and string functions. We define the *rank of f* , denoted by $\text{rank}(f)$, and the *fan-out of f* , denoted by $\text{fan-out}(f)$, as ℓ and s , respectively. Also, we will denote s_i by $\text{fan-out}_i(f)$ for every $i \in [\ell]$. The string function f is called *linear* if in $u_1 \dots u_s$ every element of X_f occurs at most once, f is called *non-deleting* if in $u_1 \dots u_s$ every element of X_f occurs at least once, and f is called *terminal-free* if $u_1, \dots, u_s \in X_f^*$. The subscript is dropped from the string function if its sort is clear from the context.

Note that for every $s' \in \mathbb{N}^* \times \mathbb{N}$, the set of linear terminal-free string functions of sort s' is finite.

Definition 2.3. A *multiple context-free grammar (over Δ)*, for short: MCFG, is a tuple (N, Δ, S, P) where N is a finite \mathbb{N} -sorted set (*non-terminals*), $S \in N_1$ (*initial non-terminal*), and $P \subseteq_{\text{fin}} \{(A, f, A_1 \dots A_\ell) \in N \times F_\Delta \times N^\ell \mid \text{sort}(f) = (\text{sort}(A_1) \dots \text{sort}(A_\ell), \text{sort}(A)), f \text{ is linear}, \ell \in \mathbb{N}\}$

(productions). We construe P as an $(N^* \times N)$ -sorted set where for every $\rho = (A, f, A_1 \cdots A_\ell) \in P$ we have $\text{sort}(\rho) = (A_1 \cdots A_\ell, A)$. \square

Let $G = (N, \Delta, S, P)$ be an MCFG and $w \in \Delta^*$. A production $(A, f, A_1 \cdots A_\ell) \in P$ is usually written as $A \rightarrow f(A_1, \dots, A_\ell)$; it inherits rank, fan-out, and fan-out $_1, \dots, \text{fan-out}_\ell$ from f . Also, $\text{rank}(G) = \max_{\rho \in P} \text{rank}(\rho)$ and $\text{fan-out}(G) = \max_{\rho \in P} \text{fan-out}(\rho)$. MCFGs of fan-out at most k and rank at most r are called (k, r) -MCFGs.

The function $\text{yield}: T_P \rightarrow \bigcup_{s=0}^{\max_{\rho \in P} \text{fan-out}(\rho)} (\Delta^*)^s$ assigns to every tree $d \in T_P$ the tuple obtained by projecting every production in d to the contained function (i.e. the second component) and then evaluating the resulting term over F_Δ .

Let $A \in N$. The set of subderivations in G from A , denoted by $D_G(A)$, is the set of all terms over P with sort A , i.e. $D_G(A) = (T_P)_A$. The set of derivations in G is $D_G = D_G(S)$. Let $w \in \Delta^*$. The set of derivations of w in G is $D_G(w) = \{d \in D_G \mid \text{yield}(d) = w\}$.⁴

The language of G is $L(G) = \{w \in \Delta^* \mid D_G(w) \neq \emptyset\}$. A language L is called *multiple context-free* if there is an MCFG G with $L = L(G)$. The set of multiple context-free languages (for which a (k, r) -MCFG exists) is denoted by MCFL ((k, r) -MCFL, respectively).

The language class (k, r) -MCFL is a substitution-closed full abstract family of languages (Seki *et al.* 1991, Theorem 3.9). Thus (k, r) -MCFL is closed under homomorphisms and under intersection with regular languages.

Definition 2.4. An \mathcal{A} -weighted MCFG (over Δ) is a tuple (N, Δ, S, P, μ) where (N, Δ, S, P) is an MCFG and $\mu: P \rightarrow \mathcal{A} \setminus \{0\}$ (weight assignment). \square

Let $G = (N, \Delta, S, P, \mu)$ be an \mathcal{A} -weighted MCFG and $w \in \Delta^*$. The set of derivations of w in G is the set of derivations of w in (N, Δ, S, P) . G inherits fan-out and rank from (N, Δ, S, P) ; \mathcal{A} -weighted MCFGs of fan-out at most k and rank at most r are called \mathcal{A} -weighted (k, r) -MCFGs. We define a function $\hat{\mu}: D_G \rightarrow \mathcal{A}$ that applies μ at every position of a given derivation and then multiplies the resulting values (in any order, since \cdot is commutative). The \mathcal{A} -weighted language induced by G is the function $\llbracket G \rrbracket: \Delta^* \rightarrow \mathcal{A}$ where for every $w \in \Delta^*$ we have $\llbracket G \rrbracket(w) = \sum_{d \in D_G(w)} \hat{\mu}(d)$.

⁴ We identify the 1-tuple containing a word $w \in \Delta^*$ with the word w itself.

Two (\mathcal{A} -weighted) MCFGs are *equivalent* if they induce the same (\mathcal{A} -weighted) language. An \mathcal{A} -weighted language L is called *multiple context-free* if there is an \mathcal{A} -weighted MCFG G such that $L = \llbracket G \rrbracket$; (k, r) -MCFL(\mathcal{A}) denotes the set of languages induced by multiple context-free \mathcal{A} -weighted grammars of fan-out at most k and rank at most r .

Example 2.5. Consider the Pr_2 -weighted MCFG $G = (N, \Delta, S, P, \mu)$ where $N_1 = \{S\}$, $N_2 = \{A, B\}$, $N = N_1 \cup N_2$, $\Delta = \{a, b, c, d\}$, and P and μ are given by

$$\begin{array}{ll}
 P: \rho_1 = S \rightarrow [x_1^1 x_2^1 x_1^2 x_2^2](A, B) & \mu: \mu(\rho_1) = 1 \\
 \rho_2 = A \rightarrow [ax_1^1, cx_1^2](A) & \mu(\rho_2) = 1/2 \\
 \rho_3 = B \rightarrow [bx_1^1, dx_1^2](B) & \mu(\rho_3) = 1/3 \\
 \rho_4 = A \rightarrow [\varepsilon, \varepsilon](\emptyset) & \mu(\rho_4) = 1/2 \\
 \rho_5 = B \rightarrow [\varepsilon, \varepsilon](\emptyset) & \mu(\rho_5) = 2/3.
 \end{array}$$

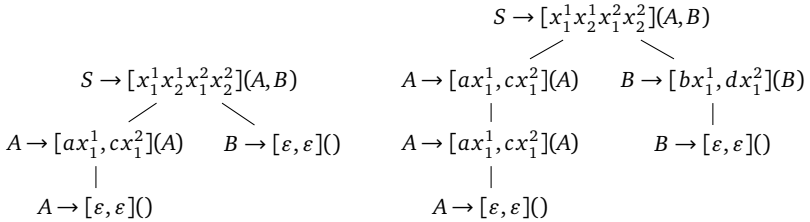
Note that G has fan-out 2 and rank 2. We observe that $\text{supp}(\llbracket G \rrbracket) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$ and for every $m, n \in \mathbb{N}$ we have

$$\begin{aligned}
 \llbracket G \rrbracket(a^m b^n c^m d^n) &= \mu(\rho_1) \cdot (\mu(\rho_2))^m \cdot \mu(\rho_4) \cdot (\mu(\rho_3))^m \cdot \mu(\rho_5) \\
 &= 1/(2^m \cdot 3^{n+1}).
 \end{aligned}$$

The derivation d of $w = ac$ and the derivation \bar{d} of $\bar{w} = aabccd$ in G are shown in Figure 3, their weights are $1/(2^1 \cdot 3^{0+1}) = 1/6$ and $1/(2^2 \cdot 3^{1+1}) = 1/36$, respectively. Since d and \bar{d} are unique derivations for w and \bar{w} , we have $\llbracket G \rrbracket(w) = 1/6$ and $\llbracket G \rrbracket(\bar{w}) = 1/36$. \square

A non-terminal is called *productive in an (\mathcal{A} -weighted) MCFG* if there is at least one subderivation starting from this non-terminal. It is obvious that every (\mathcal{A} -weighted) (k, r) -MCFL can be recognised by an (\mathcal{A} -weighted) (k, r) -MCFG that only has productive non-terminals.

Figure 3:
Derivation d
of ac (left) and
derivation \bar{d}
of $aabccd$
(right) in G
(Example 2.5)



Non-deleting normal form

An (\mathcal{A} -weighted) MCFG is called *non-deleting* if the string function in every production is linear and non-deleting. Non-deleting MCFGs are also called linear context-free rewriting systems (Vijay-Shanker *et al.* 1987) in the literature. Seki *et al.* (1991, Lemma 2.2) proved that for every (k, r) -MCFG there is an equivalent non-deleting (k, r) -MCFG. We generalise this to \mathcal{A} -weighted MCFGs.

Lemma 2.6. For every \mathcal{A} -weighted (k, r) -MCFG there is an equivalent non-deleting \mathcal{A} -weighted (k, r) -MCFG.

Proof idea. We modify the construction for the unweighted case (Seki *et al.* 1991, Lemma 2.2) such that it preserves the structure of derivations. Then a weight assignment can be defined in an obvious manner.

Proof. Let $G = (N, \Delta, S, P, \mu)$ be an \mathcal{A} -weighted (k, r) -MCFG. We construct the (k, r) -MCFG $G' = (N', \Delta, S\langle\emptyset\rangle, P')$ where $N' = \{A\langle\Psi\rangle \mid A \in N, \Psi \subseteq [\text{sort}(A)]\}$, $P' = \{\rho_\Psi \mid \rho \in P, \Psi \subseteq [\text{fan-out}(\rho)]\}$, and $\rho_\Psi = A\langle\Psi\rangle \rightarrow [u_{j_1}, \dots, u_{j_\ell}](A_1\langle\Psi_1\rangle, \dots, A_k\langle\Psi_k\rangle)$ for every $\rho = A \rightarrow [u_1, \dots, u_m](A_1, \dots, A_k) \in P$ and $\Psi \subseteq [\text{sort}(A)]$ such that

- (i) $\{j_1, \dots, j_\ell\} = [\text{sort}(A)] \setminus \Psi$ with $j_1 < \dots < j_\ell$ and
- (ii) $\Psi_i = \{j \subseteq [\text{sort}(A_i)] \mid x_i^j \text{ does not occur in } u_{j_1} \dots u_{j_\ell}\}$ for each $i \in [k]$.

The construction of G' here is a slight modification of the original construction (Lemma 2.2 in Seki *et al.* 1991, step 2 of Procedure 1) where we dropped the restrictions that $\Psi \neq [\text{sort}(A)]$ and $\Psi \neq [\text{fan-out}(\rho)]$ in the definitions of N' and P' , respectively.⁵ Note that for each ρ and Ψ , the sets Ψ_1, \dots, Ψ_k are uniquely defined. Let $g: P' \rightarrow P$ such that $g(\rho_\Psi) = \rho$ and $\widehat{g}: D_{G'} \rightarrow D_G$ be the function obtained by applying g point-wise. We show the following hypothesis by induction on the structure of subderivations:

Induction hypothesis: For every $A \in N$ and $\Psi \subseteq [\text{sort}(A)]$: \widehat{g} is a bijection between $D_{G'}(A\langle\Psi\rangle)$ and $D_G(A)$.

Induction step: Let $d \in D_G(A)$ and $\Psi \subseteq [\text{sort}(A)]$ with $d = \rho(d_1, \dots, d_k)$ for some production $\rho \in P$ and derivations $d_1 \in D_G(A_1), \dots, d_k \in D_G(A_k)$. By construction, $\Psi_1 \subseteq [\text{sort}(A_1)], \dots, \Psi_k \subseteq [\text{sort}(A_k)]$ and therefore

⁵This construction may therefore create productions of fan-out 0.

ρ_Ψ are uniquely defined for every ρ and Ψ . By the induction hypothesis, we know that there are derivations d'_1, \dots, d'_k which are unique for $(d_1, \Psi_1), \dots, (d_k, \Psi_k)$, respectively. Therefore, $d' = \rho(d'_1, \dots, d'_k)$ is unique for d and Ψ . Hence for every Ψ , \widehat{g} is a bijection between $D_{G'}(A\langle\Psi\rangle)$ and $D_G(A)$.

By construction, the new start symbol is $S\langle\emptyset\rangle$; hence for the elements of $D_{G'}$, we set $\Psi = \emptyset$ and by induction hypothesis we obtain that \widehat{g} is bijective. Since \widehat{g} preserves the structure of derivations and is a bijection, we obtain $\widehat{\mu} \circ \widehat{g} = \widehat{\mu} \circ \widehat{g}$. Hence $\llbracket(N', \Delta, S\langle\emptyset\rangle, P', \mu \circ g) \rrbracket = \llbracket G \rrbracket$. Fan-out and rank are not increased by this construction. ■

3 CS CHARACTERISATION FOR WEIGHTED MCFLS

In this section we generalise the CS characterisation of (unweighted) MCFLs (Yoshinaka *et al.* 2010, Theorem 3) to the weighted case. We prove that an \mathcal{A} -weighted MCFL L can be decomposed into an \mathcal{A} -weighted alphabetic homomorphism h , a regular language R and a multiple Dyck language mD such that $L = h(R \cap mD)$.

To show this, we use the proof idea from Droste and Vogler (2013): We separate the weight from our grammar formalism and then use the unweighted CS representation on the unweighted part. The outline of our proof is as follows:

- (i) We separate the weights from L (Lemma 3.3), obtaining an MCFL L' and a weighted alphabetic homomorphism.
- (ii) We use a corollary of the CS representation of (unweighted) MCFLs (Corollary 3.8) to obtain a CS representation of L' .
- (iii) Using the two previous items and Lemma 3.10 for the composition of weighted and unweighted alphabetic homomorphisms, we obtain a CS representation of L (Theorem 3.12).

Figure 4 outlines the proof of Theorem 3.12. The boxes represent sub-diagrams for which the corresponding lemmas prove existence of the arrows and that the sub-diagram commutes.

3.1 Separating the weights

We split a given weighted MCFG G into an unweighted MCFG $G_{\mathbb{B}}$ and a weighted alphabetic homomorphism weights_G such that $\llbracket G \rrbracket(w) = \sum_{u \in L(G_{\mathbb{B}})} \text{weights}_G(u)(w)$ for every $w \in \Delta^*$.

Lemma 3.10

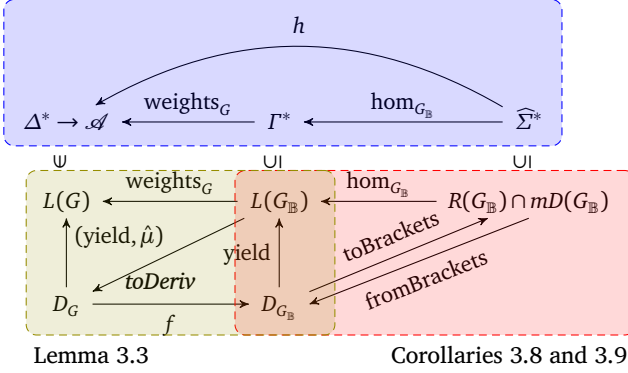


Figure 4:

Outline of the proof of Theorem 3.12

Definition 3.1. Let $G = (N, \Delta, S, P, \mu)$ be a non-deleting \mathcal{A} -weighted k -MCFG. The *unweighted MCFG* for G is the non-deleting k -MCFG $G_{\mathbb{B}} = (N, \Gamma, S, P_{\mathbb{B}})$ where $\Gamma = \Delta \cup \{\rho^i \mid \rho \in P, i \in [\text{fan-out}(\rho)]\}$ and $P_{\mathbb{B}}$ is the smallest set P' such that for every production $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_m) \in P$ there is a production

$$A \rightarrow [\rho^1 u_1, \dots, \rho^s u_s](A_1, \dots, A_m) \in P'. \quad \square$$

Definition 3.2. Let $G = (N, \Delta, S, P, \mu)$ be a non-deleting \mathcal{A} -weighted MCFG. The *weight homomorphism* for G is the \mathcal{A} -weighted alphabetic homomorphism $\text{weights}_G: \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ where $\text{weights}_G(\delta) = 1.\delta$, $\text{weights}_G(\rho^1) = \mu(\rho).\varepsilon$, and $\text{weights}_G(\rho^i) = 1.\varepsilon$ for every $\delta \in \Delta$, $\rho \in P$ and $i \in \{2, \dots, \text{fan-out}(\rho)\}$. \square

$L(G_{\mathbb{B}})$ stands in bijection to D_G via the function *toDeriv* given in Algorithm 1.

Lemma 3.3. $(k, r)\text{-MCFL}(\mathcal{A}) = \alpha\text{HOM}(\mathcal{A})((k, r)\text{-MCFL})$

Proof. (\subseteq) Let $L \in (k, r)\text{-MCFL}(\mathcal{A})$. By Lemma 2.6 there is a non-deleting \mathcal{A} -weighted (k, r) -MCFG $G = (N, \Delta, S, P, \mu)$ such that $\llbracket G \rrbracket = L$. Let $f: P \rightarrow P_{\mathbb{B}}$ where for every $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_m) \in P$ we have $f(\rho) = A \rightarrow [\rho^1 u_1, \dots, \rho^s u_s](A_1, \dots, A_m)$, in other words f represents the construction of rules in $P_{\mathbb{B}}$ from the corresponding rules in P (see Definition 3.1). We extend f to $\hat{f}: D_G \rightarrow D_{G_{\mathbb{B}}}$ by position-wise application, i.e. $\hat{f}(d) = (f(\rho))(\hat{f}(d_1), \dots, \hat{f}(d_k))$ for every subderivation $d = \rho(d_1, \dots, d_k)$ in G ; and we write f instead of \hat{f} . For every

Algorithm 1: **Input:** $w \in L(G_{\mathbb{B}})$
 Function *toDeriv* to calculate for every word in $L(G_{\mathbb{B}})$ the corresponding derivation in D_G , cf. Lemma 3.3

Output: derivation tree $t \in D_G$ corresponding to w (represented as a partial function from \mathbb{N}^* to P)

```

1  function toDeriv( $w$ )
2  let  $t$  be the empty function
3  descend( $t, \varepsilon, 1$ )
4  return  $t$ 
5  end function

6  procedure descend( $t: \mathbb{N}^* \rightarrow P, \pi \in \mathbb{N}^*, j \in \mathbb{N}$ )
7  let  $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_k) \in P$  and  $u$  such that  $\rho^j u = w$ 
8  add the assignment  $\pi \mapsto \rho$  to  $t$ 
9  remove  $\rho^j$  from the beginning of  $w$ 
10 for every symbol  $\delta'$  in  $u_j$  do
11   if  $\delta' \in \Delta$  then
12    remove  $\delta'$  from the beginning of  $w$ 
13   else
14    let  $i, j'$  such that  $x_i^{j'} = \delta'$ 
15    descend( $t, \pi i, j'$ )
16   end if
17 end for
18 end procedure
    
```

$w \in L(G_{\mathbb{B}})$ we can calculate the corresponding derivation t in G (as a function with domain $\text{dom}(t)$ and labelling function t) using *toDeriv* (Algorithm 1), hence $\text{yield} \circ f$ is bijective. We derive for every $w \in \Delta^*$:

$$\begin{aligned}
 L(w) &= \llbracket G \rrbracket(w) \\
 &= \sum_{d \in D_G(w)} \mu(d) \\
 &= \sum_{d \in D_G} (\text{weights}_G \circ \text{yield} \circ f)(d)(w) && \text{(by } \dagger \text{)} \\
 &= \sum_{\substack{d \in D_G, u \in L(G_{\mathbb{B}}) \\ u = (\text{yield} \circ f)(d)}} \text{weights}_G(u)(w) \\
 &= \sum_{u \in L(G_{\mathbb{B}})} \text{weights}_G(u)(w) && (L(G_{\mathbb{B}}) \text{ and } D_G \text{ are in bijection)} \\
 &= \text{weights}_G(L(G_{\mathbb{B}}))(w)
 \end{aligned}$$

For \dagger , one can immediately see from the definitions of f , yield , and weights_G that for every $w \in \Delta^*$ we have $(\text{weights}_G \circ \text{yield} \circ f)(d)(w) =$

$\mu(d)$ if $d \in D_G(w)$ and $(\text{weights}_G \circ \text{yield} \circ f)(d)(w) = 0$ otherwise. Hence $L = \text{weights}_G(L(G_{\mathbb{B}}))$.

(\supseteq) Let $L \in (k, r)$ -MCFL and $h: \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ be an \mathcal{A} -weighted alphabetic homomorphism. By Seki *et al.* (1991, Lemma 2.2) there is a non-deleting (k, r) -MCFG $G = (N, \Gamma, S, P)$ such that $L(G) = L$. We construct the \mathcal{A} -weighted (k, r) -MCFG $G' = (N, \Delta, S, P', \mu)$ as follows: We extend h to $h': (\Gamma \cup X)^* \rightarrow ((\Delta \cup X)^* \rightarrow \mathcal{A})$ where $h'(x) = 1.x$ for every $x \in X$ and $h'(\gamma) = h(\gamma)$ for every $\gamma \in \Gamma$. We define P' as the smallest set such that for every $\rho = A \rightarrow [u_1, \dots, u_s](A_1, \dots, A_m) \in P_{(s_1 \dots s_m, s)}$ and $(u'_1, \dots, u'_s) \in \text{supp}(h'(u_1)) \times \dots \times \text{supp}(h'(u_s))$ we have that P' contains the production $\rho' = A \rightarrow [u'_1, \dots, u'_s](A_1, \dots, A_m)$ and $\mu(\rho') = h'(u_1)(u'_1) \cdot \dots \cdot h'(u_s)(u'_s)$. Since \cdot is commutative and G is non-deleting, we obtain $\llbracket G' \rrbracket = h(L(G))$. ■

By setting $k = 1$ in the above lemma we reobtain the equivalence of 1 and 3 in Theorem 2 of Droste and Vogler (2013) for the case of complete commutative strong bimonoids.

Example 3.4. Recall the Pr_2 -weighted MCFG G from Example 2.5. The set of productions and the weight assignment of G are:

$$\begin{array}{ll}
 P: \rho_1 = S \rightarrow [x_1^1 x_2^1 x_1^2 x_2^2](A, B) & \mu: \mu(\rho_1) = 1 \\
 \rho_2 = A \rightarrow [ax_1^1, cx_1^2](A) & \mu(\rho_2) = 1/2 \\
 \rho_3 = B \rightarrow [bx_1^1, dx_1^2](B) & \mu(\rho_3) = 1/3 \\
 \rho_4 = A \rightarrow [\varepsilon, \varepsilon]() & \mu(\rho_4) = 1/2 \\
 \rho_5 = B \rightarrow [\varepsilon, \varepsilon]() & \mu(\rho_5) = 2/3.
 \end{array}$$

By Definitions 3.1 and 3.2 we obtain the MCFG $G_{\mathbb{B}} = (N, \Gamma, S, P')$ where $\Gamma = \{a, b, c, d, \rho_1^1, \rho_2^1, \rho_2^2, \rho_3^1, \rho_3^2, \rho_4^1, \rho_4^2, \rho_5^1, \rho_5^2\}$ and P' is given by

$$\begin{array}{ll}
 P': \rho'_1 = S \rightarrow [\rho_1^1 x_1^1 x_2^1 x_1^2 x_2^2](A, B) & \\
 \rho'_2 = A \rightarrow [\rho_2^1 ax_1^1, \rho_2^2 cx_1^2](A) & \rho'_4 = A \rightarrow [\rho_4^1, \rho_4^2]() \\
 \rho'_3 = B \rightarrow [\rho_3^1 bx_1^1, \rho_3^2 dx_1^2](B) & \rho'_5 = B \rightarrow [\rho_5^1, \rho_5^2]()
 \end{array}$$

and the \mathcal{A} -weighted alphabetic homomorphism $\text{weights}_G: \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ where weights_G is given for every $\gamma \in \Gamma$ and $\omega \in \Delta \cup \{\varepsilon\}$ by

$$\text{weights}_G(\gamma)(\omega) = \begin{cases} \mu(\rho_i) & \text{if } \gamma = \rho_i^1 \text{ and } \omega = \varepsilon \text{ for } 1 \leq i \leq 5 \\ 1 & \text{if } \gamma = \rho_i^2 \text{ and } \omega = \varepsilon \text{ for } 2 \leq i \leq 5 \\ 1 & \text{if } \gamma \in \Delta \text{ and } \omega = \gamma \\ 0 & \text{otherwise,} \end{cases}$$

Now recall the derivation $d = \rho_1(\rho_2(\rho_4), \rho_5)$ of $w = ac$. Then

$$\begin{aligned} f(d) &= \rho_1'(\rho_2'(\rho_4'), \rho_5') && =: d', \\ g(d') &= \rho_1^1 \rho_2^1 a \rho_4^1 \rho_5^1 \rho_2^2 c \rho_4^2 \rho_5^2 && =: w', \text{ and} \\ \text{weights}_G(w') &= (1 \cdot 1/2 \cdot 1 \cdot 1/2 \cdot 2/3 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \cdot w && = (1/6) \cdot w. \quad \square \end{aligned}$$

3.2 The unweighted CS characterisation

We recall the definition of multiple Dyck languages (Yoshinaka *et al.* 2010, Definition 1):

Definition 3.5. Let Δ be a finite \mathbb{N} -sorted set,⁶ $(\bar{\cdot})$ be a bijection between Δ and some alphabet $\bar{\Delta}$, $k = \max_{\delta \in \Delta} \text{sort}(\delta)$, and $r \geq k$. The *multiple Dyck grammar with respect to Δ and r* is the (k, r) -MCFG $G_\Delta^r = (\{A_1, \dots, A_k\}, \widehat{\Delta}, A_1, P)$ where $\widehat{\Delta} = \{\delta^{[i]}, \bar{\delta}^{[i]} \mid \delta \in \Delta, i \in [\text{sort}(\delta)]\}$, $\text{sort}(A_i) = i$ for every $i \in [k]$, and P is the smallest set such that

- (i) for every linear non-deleting⁷ terminal-free string function $f \in (F_\Delta)_{(s_1 \dots s_\ell, s)}$ with $\ell \in [r]$ and $s_1, \dots, s_\ell, s \in [k]$ we have

$$A_s \rightarrow f(A_{s_1}, \dots, A_{s_\ell}) \in P,$$

- (ii) for every $\delta \in \Delta$ with $\text{sort } s$ we have

$$A_s \rightarrow [\delta^{[1]} x_1^1 \bar{\delta}^{[1]}, \dots, \delta^{[s]} x_1^s \bar{\delta}^{[s]}](A_s) \in P, \text{ and}$$

- (iii) for every $s \in [k]$ we have

$$A_s \rightarrow [u_1, \dots, u_s](A_s) \in P$$

where $u_i \in \{x_i, x_i \delta^{[1]} \bar{\delta}^{[1]}, \delta^{[1]} \bar{\delta}^{[1]} x_i \mid \delta \in \Delta_1\}$ for every $i \in [s]$.

⁶In Yoshinaka *et al.* (2010), \mathbb{N} -sorted sets are called indexed sets and sort is denoted as dim .

⁷We add the restriction “non-deleting” in comparison to the original definition since the proof of Lemma 1 in Yoshinaka *et al.* (2010) only uses non-deleting rules.

The *multiple Dyck language with respect to Δ and r* , denoted by $mD(\Delta, r)$, is $L(G_\Delta^r)$. We call $\max_{\delta \in \Delta} \text{sort}(\delta)$ the *dimension* and r the *rank* of $mD(\Delta, r)$. The *set of multiple Dyck languages of dimension at most k and rank at most r* is denoted by (k, r) -mDYCK. \square

Yoshinaka *et al.* (2010) define (in Section 3.2) a sorted alphabet Δ , a right-linear regular grammar R , and a homomorphism h for some given non-deleting MCFG G that has no rule with two or more identical non-terminals on the right-hand side (this form of G can be assumed without loss of generality). We recall their construction. To fit our notation and highlight the connection to G , we will conceive R as an FSA and call it $\mathcal{M}(G)$; also, h will be called hom_G .

Definition 3.6. Let $G = (N, \Gamma, S, P)$ be an MCFG. The *generator set with respect to G* is the \mathbb{N} -sorted alphabet

$$\Sigma = \{\llbracket \gamma \mid \gamma \in \Gamma \rrbracket \cup \{\llbracket \rho \mid \rho \in P \rrbracket \cup \{\llbracket \rho, i \mid \rho \in P, i \in [\text{rank}(\rho)] \rrbracket\}$$

where $\text{sort}(\llbracket \gamma \rrbracket) = 1$, $\text{sort}(\llbracket \rho \rrbracket) = \text{fan-out}(\rho)$, and $\text{sort}(\llbracket \rho, i \rrbracket) = \text{fan-out}_i(\rho)$ for every $\gamma \in \Gamma$, $\rho \in P$, and $i \in \text{rank}(\rho)$. The *generator alphabet with respect to G* is

$$\widehat{\Sigma} = \{\llbracket u^{[i]} \rrbracket, \llbracket u^{[i]} \rrbracket \mid \llbracket u \rrbracket \in \Sigma, i \in [\text{sort}(\sigma)]\}.$$

For each $u = \gamma_1 \cdots \gamma_m \in \Gamma^*$ (with $\gamma_1, \dots, \gamma_m \in \Gamma$), we will abbreviate $\llbracket \gamma_1^{[1]} \rrbracket \llbracket \gamma_1^{[1]} \rrbracket \dots \llbracket \gamma_m^{[1]} \rrbracket \llbracket \gamma_m^{[1]} \rrbracket$ by \tilde{u} . The *generator automaton with respect to G* is the FSA $\mathcal{M}(G) = (Q, \widehat{\Delta}, S^{[1]}, \{T\}, \tau)$ where $Q = \{A^{[k]} \mid A \in N, k \in [\text{sort}(A)]\} \cup \{T\}$ and τ is the smallest set that contains for every production $\rho = A \rightarrow [v_1, \dots, v_s](B_1, \dots, B_m) \in P$ and each $k \in [s]$ (where v_k is of the form $u_{k,0} x_{i(k,1)}^{j(k,1)} u_{k,1} \cdots x_{i(k,p_k)}^{j(k,p_k)} u_{k,p_k}$ with $u_{k,0}, \dots, u_{k,p_k} \in \Gamma^*$), the transitions

$$\begin{aligned} (A^{[k]}, \llbracket \rho^{[k]} \tilde{u}_{k,0} \rrbracket \llbracket \rho^{[k]} \rrbracket, T) & \quad \text{if } p_k = 0, \\ (A^{[k]}, \llbracket \rho^{[k]} \tilde{u}_{k,0} \llbracket \rho, i(k,1) \rrbracket \rrbracket, B_{i(k,1)}^{[j(k,1)]}) & \quad \text{if } p_k > 0, \\ (T, \llbracket \rho, i(k, \ell-1) \rrbracket \tilde{u}_{k, \ell-1} \llbracket \rho, i(k, \ell) \rrbracket \rrbracket, B_{i(k, \ell)}^{[j(k, \ell)]}) & \quad \text{if } p_k > 0, \text{ for every } \ell \in [p_k], \\ (T, \llbracket \rho, i(k, p_k) \rrbracket \tilde{u}_{k, p_k} \llbracket \rho \rrbracket, T) & \quad \text{if } p_k > 0. \end{aligned}$$

The *generator language with respect to G* is $R(G) = L(\mathcal{M}(G))$. The homomorphism $\text{hom}_G: \widehat{\Delta} \rightarrow \Gamma^*$ is given by

$$\text{hom}_G(\sigma) = \begin{cases} \gamma & \text{if } \sigma = \llbracket \gamma^{[1]} \rrbracket \text{ for some } \gamma \in \Gamma \\ \varepsilon & \text{otherwise.} \end{cases} \quad \square$$

From the four types of transitions in $\mathcal{M}(G)$, it is easy to see that $\mathcal{M}(G)$ is deterministic, i.e. for each given state and input, there is at most one successor state. We will denote $L(D_\Sigma^{\text{rank}(G)})$ (cf. Definition 3.5) by $mD(G)$ to highlight its connection to the MCFG G .

Example 3.7 (Examples 2.5 and 3.4 continued). Figure 5 shows the FSA $\mathcal{M}(G_{\mathbb{B}})$. An edge labelled with a set L of words denotes a set of transitions each reading a word in L . Note that $R(G_{\mathbb{B}})$ is not finite. \square

The following is Theorem 3 of Yoshinaka *et al.* (2010) where “homomorphism” is replaced by “alphabetic homomorphism”.

Corollary 3.8. Let L be a language, $k \in \mathbb{N}$, and $r \in \mathbb{N}_+$. Then the following are equivalent:

- (i) $L \in (k, r)$ -MCFL
- (ii) There are an alphabetic homomorphism h_2 , a regular language R , and a multiple Dyck language mD of at most dimension k and rank r with $L = h_2(R \cap mD)$.

Proof. The construction of the homomorphism in Yoshinaka *et al.* (2010, Section 3.2) already satisfies the definition of an alphabetic homomorphism. \blacksquare

Corollary 3.9. For every MCFG G , there is a bijection between D_G and $R(G) \cap mD(G)$.

Proof. The constructions in Lemmas 1 and 3 in Yoshinaka *et al.* (2010) already hint at the bijection between $R(G) \cap mD(G)$ and D_G , we will merely point out the respective functions $\text{toBrackets}: D_G \rightarrow R(G) \cap mD(G)$ and $\text{fromBrackets}: R(G) \cap mD(G) \rightarrow D_G$ here.

Let Δ be the generator set with respect to G and $r = \text{rank}(G)$. We examine the proof of Lemma 1 in Yoshinaka *et al.* (2010). They construct for every rule $A \rightarrow f(B_1, \dots, B_k)$ in G and all tuples $\bar{\tau}_1, \dots, \bar{\tau}_k$ that are generated by B_1, \dots, B_k in G , respectively, a tuple $\bar{u} = (u_1, \dots, u_m)$ that is generated from A_m in G_Δ^r . For each $i \in [m]$, $\mathcal{M}(G)$ recognises u_i on the way from $A^{[i]}$ to T , and $f(\text{hom}_G(\tau_1), \dots, \text{hom}_G(\tau_k)) = \text{hom}_G(\bar{u})$, where hom_G is applied to tuples component-wise. Now we only look at the initial non-terminal S . Then \bar{u} has only one component and this construction can be conceived as a function $\text{toBrackets}: D_G \rightarrow R(G) \cap mD(G)$ such that $\text{hom}_G \circ \text{toBrackets} = \text{yield}$.

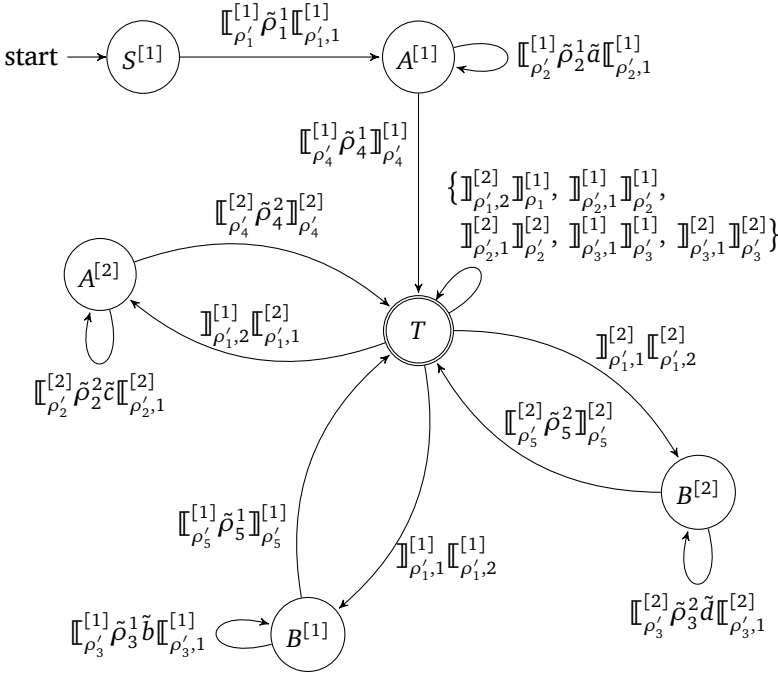


Figure 5:
Automaton
 $\mathcal{M}(G_{\mathbb{B}})$ (cf.
Example 3.7)

In Lemma 3, Yoshinaka *et al.* (2010) give a construction for the opposite direction by recursion on the structure of derivations in G_{Δ} . In a similar way as above, we view this construction as a function $\text{fromBrackets}: R(G) \cap mD(G) \rightarrow D_G$ such that $\text{yield} \circ \text{fromBrackets} = \text{hom}_G$. Then we have $\text{hom}_G \circ \text{toBrackets} \circ \text{fromBrackets} = \text{hom}_G$, and hence $\text{toBrackets} \circ \text{fromBrackets}$ is the identity on $R(G) \cap mD(G)$. ■

3.3

Composing the homomorphisms

Lemma 3.10. $\alpha\text{HOM}(\mathcal{A}) \circ \alpha\text{HOM} = \alpha\text{HOM}(\mathcal{A})$

Proof. (\subseteq) Let $h_1: \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ be an alphabetic \mathcal{A} -weighted homomorphism and $h_2: \Sigma^* \rightarrow \Gamma^*$ be an alphabetic homomorphism. By the definitions of $\alpha\text{HOM}(\mathcal{A})$ and αHOM , there exist $h'_1: \Gamma \rightarrow (\Delta \cup \{\varepsilon\} \rightarrow \mathcal{A})$ and $h'_2: \Sigma \rightarrow \Gamma \cup \{\varepsilon\}$ such that $\widehat{h'_1} = h_1$ and $\widehat{h'_2} = h_2$. Since $h_1(\text{codom}(h'_2)) \subseteq (\Delta \cup \{\varepsilon\} \rightarrow \mathcal{A})$ there is some $h \in \alpha\text{HOM}(\mathcal{A})$ such that $h = h_1 \circ h_2$; hence $h_1 \circ h_2 \in \alpha\text{HOM}(\mathcal{A})$.

(\supseteq) Let $h: \Sigma \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ be an alphabetic \mathcal{A} -weighted homomorphism. Clearly $i: \Sigma^* \rightarrow \Sigma^*$ with $i(w) = w$ for every $w \in \Sigma^*$ is an alphabetic homomorphism. Then we have $h \circ i = h$. \blacksquare

Example 3.11 (Examples 3.4 and 3.7 continued). The homomorphism $h: (\Sigma \cup \bar{\Sigma})^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ obtained from $\text{weights}_G: \Gamma^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ and $\text{hom}_{G_{\mathbb{B}}}: (\Sigma \cup \bar{\Sigma})^* \rightarrow \Gamma^*$ by the construction for \subseteq in Lemma 3.10 is given for every $\sigma \in \Sigma$ and $\omega \in \Delta \cup \{\varepsilon\}$ by

$$h(\sigma)(\omega) = \begin{cases} \mu(\rho_i) & \text{if } \sigma = \llbracket \rho_i \rrbracket^{\llbracket 1 \rrbracket} \text{ and } \omega = \varepsilon \text{ for some } i \in [5] \\ 1 & \text{if } \sigma \notin \{ \llbracket \rho_i \rrbracket^{\llbracket 1 \rrbracket} \mid i \in [5] \} \cup \{ \llbracket \delta \rrbracket^{\llbracket 1 \rrbracket} \mid \delta \in \Delta \} \text{ and } \omega = \varepsilon \\ 1 & \text{if } \sigma = \llbracket \delta \rrbracket^{\llbracket 1 \rrbracket} \text{ and } \omega = \delta \text{ for some } \delta \in \Delta \\ 0 & \text{otherwise.} \end{cases} \quad \square$$

3.4 The weighted CS characterisation

Theorem 3.12. Let L be an \mathcal{A} -weighted language over Σ , $k \in \mathbb{N}$, and $r \in \mathbb{N}_+$. The following are equivalent:

- (i) $L \in (k, r)$ -MCFL(\mathcal{A})
- (ii) there are an \mathcal{A} -weighted alphabetic homomorphism h , a regular language R , and an multiple Dyck language mD of dimension at most k and rank r with $L = h(R \cap mD)$.

Proof. (i) \Rightarrow (ii) There are some $L' \in (k, r)$ -MCFL, $h, h_1 \in \alpha\text{HOM}(\mathcal{A})$, $h_2 \in \alpha\text{HOM}$, $mD \in k$ -mDYCK $_c$, and $R \in \text{REC}$ such that

$$\begin{aligned} L &= h_1(L') && \text{(by Lemma 3.3)} \\ &= h_1(h_2(R \cap mD)) && \text{(by Corollary 3.8)} \\ &= h(R \cap mD) && \text{(by Lemma 3.10)} \end{aligned}$$

(ii) \Rightarrow (i) We use Definition 3.5 and Lemma 3.3, and the closure of (k, r) -MCFG under intersection with regular languages and application of homomorphisms. \blacksquare

Corollary 3.13. For every \mathcal{A} -weighted MCFG G , there is a bijection between D_G and $R(G_{\mathbb{B}}) \cap mD(G_{\mathbb{B}})$.

Proof. There are bijections between D_G and $L(G_{\mathbb{B}})$ by claims in the proof of Lemma 3.3, between $L(G_{\mathbb{B}})$ and $D_{G_{\mathbb{B}}}$ by claims in the proof of Lemma 3.3, and between $D_{G_{\mathbb{B}}}$ and $R(G_{\mathbb{B}}) \cap mD(G_{\mathbb{B}})$ by Corollary 3.9. \blacksquare

4 CONGRUENCE MULTIPLE DYCK LANGUAGES

According to Kanazawa (2014, Section 1) there is no definition of multiple Dyck languages using a cancellation law. The congruence multiple Dyck languages (Definition 4.2) close this gap. Even though congruence multiple Dyck languages turn out to be quite different from the multiple Dyck languages by Yoshinaka *et al.* (2010) (see Proposition 4.5 and Observation 4.6), we argue that they are still useful since they allow a CS representation (Theorem 4.11) and they can be utilised more efficiently for CS parsing than multiple Dyck languages (see Section 5.5).

For the rest of this section let Σ be an alphabet. Also let $\overline{\Sigma}$ be a set (disjoint from Σ) and (\cdot) be a bijection between Σ and $\overline{\Sigma}$. Intuitively Σ and $\overline{\Sigma}$ are sets of opening and closing parentheses and (\cdot) matches an opening to its closing parenthesis.

We define \equiv_{Σ} as the smallest congruence relation on the free monoid $(\Sigma \cup \overline{\Sigma})^*$ where for every $\sigma \in \Sigma$ the cancellation rule $\sigma\overline{\sigma} \equiv_{\Sigma} \varepsilon$ holds. The *Dyck language with respect to Σ* , denoted by $D(\Sigma)$, is $[\varepsilon]_{\equiv_{\Sigma}}$. The *set of Dyck languages* is denoted by DYCK.

Example 4.1. Let $\Sigma = \{(\langle, [, \llbracket\}$. We abbreviate $\bar{(\langle}$, $\bar{\langle}$, $\bar{[}$, and $\bar{\llbracket}$ by \rangle , \rangle , \rrbracket , and \rrbracket , respectively. Then we have for example $\llbracket(\rangle)\langle\rangle \equiv_{\Sigma} \llbracket\rrbracket\langle\rangle \equiv_{\Sigma} \llbracket\rrbracket \equiv_{\Sigma} \varepsilon$ and $(\langle)\rrbracket\langle\rangle \equiv_{\Sigma} (\langle)\rrbracket(\rangle) \equiv_{\Sigma} (\langle)\rrbracket \not\equiv_{\Sigma} \varepsilon$. \square

Let \mathfrak{P} be a partition of Σ . We define $\equiv_{\Sigma, \mathfrak{P}}$ as the smallest congruence relation on the free monoid $(\Sigma \cup \overline{\Sigma})^*$ such that if $v_1 \cdots v_{\ell} \equiv_{\Sigma, \mathfrak{P}} \varepsilon$ with $v_1, \dots, v_{\ell} \in D(\Sigma)$, then the *cancellation rule*

$$u_0\sigma_1v_1\overline{\sigma_1}u_1 \cdots \sigma_{\ell}v_{\ell}\overline{\sigma_{\ell}}u_{\ell} \equiv_{\Sigma, \mathfrak{P}} u_0 \cdots u_{\ell}$$

holds for every $\{\sigma_1, \dots, \sigma_{\ell}\} \in \mathfrak{P}$ and $u_0, \dots, u_{\ell} \in D(\Sigma)$. Intuitively, every cell of \mathfrak{P} denotes a set of *linked* opening parentheses, i.e. parentheses that must be consumed simultaneously by $\equiv_{\Sigma, \mathfrak{P}}$.

Definition 4.2. The *congruence multiple Dyck language with respect to Σ and \mathfrak{P}* , denoted by $mD_{\mathfrak{C}}(\Sigma, \mathfrak{P})$, is $[\varepsilon]_{\equiv_{\Sigma, \mathfrak{P}}}$. \square

Example 4.3. Let $\Sigma = \{(\langle, [, \llbracket\}$ and $\mathfrak{P} = \{p_1, p_2\}$ where $p_1 = \{(\langle, \rangle\}$ and $p_2 = \{[, \rrbracket\}$. We abbreviate $\bar{(\langle}$, $\bar{\langle}$, $\bar{[}$, and $\bar{\llbracket}$ by \rangle , \rangle , \rrbracket , and \rrbracket , respectively. Then, using the cancellation rule, we have for example $\llbracket(\rangle)\langle\rangle \equiv_{\Sigma, \mathfrak{P}} \varepsilon$ since $p_2 = \{[, \rrbracket\} \in \mathfrak{P}$, $(\rangle) \equiv_{\Sigma, \mathfrak{P}} \varepsilon$, and $u_0 = u_1 = u_2 = \varepsilon$.

But $\llbracket () \rrbracket \langle \llbracket () \rrbracket \rangle \notin_{\Sigma, \mathfrak{P}} \varepsilon$ since when instantiating the cancellation rule with any of the two cells of \mathfrak{P} , we can not reduce $\llbracket () \rrbracket \langle \llbracket () \rrbracket \rangle$:

- (i) If we choose $\{\sigma_1, \sigma_2\} = \{\llbracket, \rrbracket\}$ then we would need to set $u_1 = \langle$ and $u_2 = \rangle$, but they are not in $D(\Sigma)$, also $() \notin_{\Sigma, \mathfrak{P}} \varepsilon$;
- (ii) If we choose $\{\sigma_1, \sigma_2\} = \{\langle, \rangle\}$ then we would need to set $u_0 = \llbracket$ and $u_1 = \rrbracket$, but they are not in $D(\Sigma)$, also $\llbracket \rrbracket \notin_{\Sigma, \mathfrak{P}} \varepsilon$.

Hence $\llbracket () \rrbracket \langle \rangle, () \langle \rangle \in mD_c(\Sigma, \mathfrak{P})$ and $\llbracket () \rrbracket \langle \llbracket () \rrbracket \rangle \notin mD_c(\Sigma, \mathfrak{P})$. \square

Observation 4.4. From the definition of $\equiv_{\Sigma, \mathfrak{P}}$ it is easy to see that for every $u_1, \dots, u_k \in D(\Sigma)$ and $v_1, \dots, v_\ell \in D(\Sigma)$ we have that $u_1 \cdots u_k, v_1 \cdots v_\ell \in mD_c(\Sigma, \mathfrak{P})$ implies that every permutation of $u_1, \dots, u_k, v_1, \dots, v_\ell$ is in $mD_c(\Sigma, \mathfrak{P})$. \blacksquare

The *dimension* of $mD_c(\Sigma, \mathfrak{P})$ is $\max_{p \in \mathfrak{P}} |p|$. The *set of congruence multiple Dyck languages (of at most dimension k)* is denoted by $mDYCK_c$ (k - $mDYCK_c$, respectively).

Proposition 4.5. For each $mD \in (k, r)$ - $mDYCK$ there is an $mD_c \in k$ - $mDYCK_c$ such that $mD \subseteq mD_c$.

Proof idea. We construct a congruence multiple Dyck language mD_c of dimension at most k such that, if a tuple (w_1, \dots, w_m) can be generated in G_Δ^r from non-terminal A_m , then $w_1, \dots, w_m \in D(\Sigma)$ and $w_1 \cdots w_m \in mD_c$. We prove this by induction on the structure of derivations in G_Δ^r .

Proof. Let $mD \in (k, r)$ - $mDYCK$. Then there is an \mathbb{N} -sorted set Δ such that $mD = mD(\Delta, r)$ and $\max_{\delta \in \Delta} \text{sort}(\delta) \leq k$. We define $p_\delta = \{\delta^{[i]} \mid i \in [\text{sort}(\delta)]\}$ for every $\delta \in \Delta$, $\Sigma = \bigcup_{\delta \in \Delta} p_\delta$, and $\mathfrak{P} = \{p_\delta \mid \delta \in \Delta\}$. Clearly $\max_{p \in \mathfrak{P}} |p| \leq k$. Thus $mD_c(\Sigma, \mathfrak{P}) \in k$ - $mDYCK_c$. Let $\text{Tup}(G_\Delta^r, A)$ denote the set of tuples generated in G_Δ^r when starting with non-terminal A where A is not necessarily initial. In the following we show that for every $m \in [\max_{\delta \in \Delta} \text{sort}(\delta)]$ and $w_1, \dots, w_m \in (\Sigma \cup \bar{\Sigma})^*$:

$$(w_1, \dots, w_m) \in \text{Tup}(G_\Delta^r, A_m) \implies w_1 \cdots w_m \in mD_c(\Sigma, \mathfrak{P}) \quad (*)$$

$$\wedge w_1, \dots, w_m \in D(\Sigma).$$

It follows from the definitions of Tup and G_Δ^r that $(w_1, \dots, w_m) \in \text{Tup}(G_\Delta^r, A_m)$ implies that there are a rule $A_m \rightarrow f(A_{m_1}, \dots, A_{m_\ell})$ in G_Δ^r and tuples $\tilde{u}_i = (u_i^1, \dots, u_i^{m_i}) \in \text{Tup}(G_\Delta^r, A_{m_i})$ for every $i \in [\ell]$ such

that $f(\vec{u}_1, \dots, \vec{u}_\ell) = (w_1, \dots, w_m)$. By applying the induction hypothesis ℓ times, we also have that $u_1^1, \dots, u_1^{m_1}, \dots, u_\ell^1, \dots, u_\ell^{m_\ell} \in D(\Sigma)$ and $u_1^1 \cdots u_1^{m_1}, \dots, u_\ell^1 \cdots u_\ell^{m_\ell} \in mD_c(\Sigma, \mathfrak{P})$. We distinguish three cases (each corresponding to one type of rule in G_Δ^r):

- (i) f is linear, non-deleting, and terminal-free. Then we have for every $i \in [m]$ that $w_i \in \{u_1^1, \dots, u_1^{m_1}, \dots, u_\ell^1, \dots, u_\ell^{m_\ell}\}^*$ and therefore also $w_i \in D(\Sigma)$. Furthermore, by applying Observation 4.4 ($\ell - 1$) times, we have that $w_1 \cdots w_m \in mD_c(\Sigma, \mathfrak{P})$.
- (ii) $f = [\delta^{[1]}x_1^1\bar{\delta}^{[1]}, \dots, \delta^{[m]}x_1^m\bar{\delta}^{[m]}]$; then $\ell = 1$, $m_1 = m$, and for every $i \in [m]$ we have $w_i = \delta^{[i]}u_1^i\bar{\delta}^{[i]}$ and since $u_1^i \in D(\Sigma)$, also $w_i \in D(\Sigma)$. Furthermore, $w_1 \cdots w_m = \delta^{[1]}u_1^1\bar{\delta}^{[1]} \dots \delta^{[m]}u_1^m\bar{\delta}^{[m]} \in mD_c(\Sigma, \mathfrak{P})$ due to the cancellation rule.
- (iii) $f = [u_1, \dots, u_m]$ where $u_i \in \{x_i^1, x_i^1\delta^{[1]}\bar{\delta}^{[1]}, \delta^{[1]}\bar{\delta}^{[1]}x_i^1 \mid \delta \in \Delta_1\}$ for every $i \in [m]$; then $w_i \in \{u_i^1, u_i^1\delta^{[1]}\bar{\delta}^{[1]}, \delta^{[1]}\bar{\delta}^{[1]}u_i^1 \mid \delta \in \Delta_1\}$ for every $i \in [m]$, $\ell = 1$, and $m_1 = m$. Since \equiv_Σ is a congruence relation (in particular, \equiv_Σ respects composition), we have that $w_1, \dots, w_m \in D(\Sigma)$. By applying Observation 4.4 m times, we have that $w_1 \cdots w_m \in mD_c(\Sigma, \mathfrak{P})$. ■

From the above proof we can easily see that the rank r of a multiple Dyck language can not be taken into account by a congruence multiple Dyck language. This leads us to the next observation.

Observation 4.6. Let mD be a multiple Dyck language and mD_c be a congruence multiple Dyck language. Then $mD \neq mD_c$. ■

Similar to multiple Dyck languages, the congruence multiple Dyck languages cover the Dyck languages if we set the dimension to 1. Also they form a hierarchy with increasing dimension.

Proposition 4.7. $DYCK = 1\text{-}mDYCK_c \subsetneq 2\text{-}mDYCK_c \subsetneq \dots$

Proof. We have the equality since the dimension of some partition \mathfrak{P} of Σ is 1 if and only if $\mathfrak{P} = \{\{\sigma\} \mid \sigma \in \Sigma\}$. Then we have $\equiv_\Sigma = \equiv_{\Sigma, \mathfrak{P}}$ and thus $D(\Sigma) = mD_c(\Sigma, \mathfrak{P})$. Hence $DYCK = 1\text{-}mDYCK_c$. We get “ \subsetneq ” from the definition of $k\text{-}mDYCK_c$. ■

4.1 Membership in a congruence multiple Dyck language

We give an algorithm to decide membership in a congruence multiple Dyck language (Algorithm 2). It is closely related to the cancellation

Algorithm 2: **Input:** Σ , \mathfrak{P} , and $w \in (\Sigma \cup \overline{\Sigma})^*$
 Function **Output:** True if $w \in mD_c(\Sigma, \mathfrak{P})$, False otherwise
isMember to decide membership in $mD_c(\Sigma, \mathfrak{P})$

```

1  function isMember( $\Sigma, \mathfrak{P}, w$ )
2  if  $w \notin D(\Sigma)$  then return False end if
3  let  $(\sigma_1 u_1 \bar{\sigma}_1, \dots, \sigma_\ell u_\ell \bar{\sigma}_\ell) = \text{split}(\Sigma, w)$  such that  $\sigma_1, \dots, \sigma_\ell \in \Sigma$ 
4  let  $\mathcal{I} = \emptyset$ 
5  for each  $I = \{i_1, \dots, i_k\} \subseteq [\ell]$  with  $\{\sigma_{i_1}, \dots, \sigma_{i_k}\} \in \mathfrak{P}$  do
6    if isMember( $\Sigma, \mathfrak{P}, u_{i_1} \cdots u_{i_k}$ ) then
7      add  $I$  as an element to  $\mathcal{I}$ 
8    end if
9  end for
10 for each  $J \subseteq \mathcal{I}$  do
11   if  $J$  is a partition of  $[\ell]$  then return True end if
12 end for
13 return False
14 end function

```

rule and thus provides an algorithmic view on congruence multiple Dyck languages. Although the algorithm is at least exponential in a polynomial of the length of the input word, it becomes quadratic if we only accept input words of a specific form. The parsing algorithm presented in Section 5 will only consider words of that form.

Algorithm 2 works roughly as follows: It is a recursive algorithm. In every call of *isMember*, it checks if the given word can be reduced to ε by applications of the cancellation rule. For substrings $\sigma_1 v_1 \bar{\sigma}_1, \dots, \sigma_\ell v_\ell \bar{\sigma}_\ell$ to be cancelled, the string $v_1 \cdots v_\ell$ must be an equivalence multiple Dyck word; this is checked with a recursive call to *isMember*. Note that it suffices to only apply the cancellation rule from left to right. To account for all possible applications of the cancellation rule, *isMember* must consider all decompositions of the input string into Dyck words. For this purpose, recall *split* (from Section 4) that splits a given Dyck word into shortest non-empty Dyck words.

Outline of *isMember*

In the following, all line numbers refer to Algorithm 2. We first check if w is in $D(\Sigma)$, e.g. with the context-free grammar in (7.6) in Salomaa (1973). If w is not in $D(\Sigma)$, it is also not in $mD_c(\Sigma, \mathfrak{P})$ and we return False. Otherwise, we split w into shortest non-empty

Dyck words (on line 3), i.e. we compute the tuple (u_1, \dots, u_ℓ) such that $u_1, \dots, u_\ell \in D(\Sigma) \setminus \{\varepsilon\}$, $u_1 \cdots u_\ell = u$, and for every $i \in [\ell]$ there are no $u'_i, v'_i \in D(\Sigma) \setminus \{\varepsilon\}$ with $u'_i v'_i = u_i$. We denote (u_1, \dots, u_ℓ) by $split(u)$. Note that $split(u)$ can be calculated in time and space linear in $|u|$ with the help of a pushdown transducer (Aho and Ullman 1972, Section 3.1.4): We initially write “(” on the output tape. Whenever we read an element of Σ , we write that element on the output tape and push it on the stack. Whenever we read an element of $\bar{\Sigma}$, we write that element on the output tape and pop it from the stack. Upon reaching the bottom of the stack, we write a “,” on the output tape. Finally, we write “)” on the output tape. Then the inscription of the output tape is (u_1, \dots, u_ℓ) . Since each of those shortest non-empty Dyck words has the form $\sigma u \bar{\sigma}$ for some $\sigma \in \Sigma$ and $u \in (\Sigma \cup \bar{\Sigma})^*$, we write $(\sigma_1 u_1 \bar{\sigma}_1, \dots, \sigma_\ell u_\ell \bar{\sigma}_\ell)$ for the left-hand side of the assignment on line 3. On lines 4 to 9 we calculate the set \mathcal{S} of sets of indices $I = \{i_1, \dots, i_k\}$ such that the outer parentheses of the substrings $\sigma_{i_1} u_{i_1} \bar{\sigma}_{i_1}, \dots, \sigma_{i_k} u_{i_k} \bar{\sigma}_{i_k}$ of w can be reduced with one step of the cancellation rule. This reduction is possible if there exists an appropriate cell in \mathfrak{P} (checked on line 5) and if $u_{i_1} \cdots u_{i_k}$ is in $mD_c(\Sigma, \mathfrak{P})$ (checked on line 6). Therefore, at the end of line 9, each element of \mathcal{S} represents one possible application of the cancellation rule. In order for $\equiv_{\Sigma, \mathfrak{P}}$ to reduce w to ε , each component of $(\sigma_1 u_1 \bar{\sigma}_1, \dots, \sigma_\ell u_\ell \bar{\sigma}_\ell)$ needs to be reduced (exactly once) by an application of the cancellation rule. This is equivalent to a subset of \mathcal{S} being a partition of $[\ell]$ (lines 10 to 12). If no such subset exists, then w can not be reduced to ε and we return False on line 13.

Example 4.8 (Example 4.3 continued). Table 1 shows a run of Algorithm 2 on the word $[(\)][\langle \rangle]$ where we report return values and a subset of the variable assignment, when necessary, at the line ending. The recursive calls to *isMember* are indented. Table 2 shows the run of Algorithm 2 on the word $[(\)][\]][\][\langle \rangle]$. \square

In light of the close link between Algorithm 2 and the relation $\equiv_{\Sigma, \mathfrak{P}}$ we omit the proof of correctness.

Proof of termination for Algorithm 2. If $w = \varepsilon$, then $\ell = 0$, the loop on lines 5 to 9 has no I 's to consider, the loop on lines 10 to 13 has only $J = \emptyset$ to consider, which is a partition of $[\ell] = \emptyset$, and hence the algorithm terminates on line 11. If $w \notin D(\Sigma)$, the algorithm terminates

<p>Run of Algorithm 2 on the word $\llbracket () \rrbracket \llbracket \langle \rangle \rrbracket$, cf. Examples 4.3 and 4.8.</p>	<p>Table 1: $isMember(\Sigma, \mathfrak{P}, \llbracket () \rrbracket \llbracket \langle \rangle \rrbracket)$ line 3: $\ell = 2, \sigma_1 = \llbracket, \sigma_2 = \llbracket, u_1 = (), u_2 = \langle \rangle$ line 5: $\mathcal{S} = \emptyset, k = 2, i_1 = 1, i_2 = 2$ line 6: $isMember(\Sigma, \mathfrak{P}, () \langle \rangle)$ line 3: $\ell = 2, \sigma_1 = \langle, \sigma_2 = \langle, u_1 = \varepsilon = u_2$ line 5: $\mathcal{S} = \emptyset, k = 2, i_1 = 1, i_2 = 2$ line 6: $isMember(\Sigma, \mathfrak{P}, \varepsilon)$ line 3: $\ell = 0$ line 10: $J = \emptyset$ line 11: return True line 7: $\mathcal{S} = \{\{1, 2\}\}$ line 10: $J = \emptyset$ line 10: $J = \{\{1, 2\}\}$ line 11: return True line 7: $\mathcal{S} = \{\{1, 2\}\}$ line 10: $J = \emptyset$ line 10: $J = \{\{1, 2\}\}$ line 11: return True</p>
--	---

on line 2. The loop on lines 5 to 9 considers only finitely many values I . Thus there are only finitely many calls to $isMember$ on line 6 for each recursion. In the call of $isMember$, the length of the third argument $u_{i_1} \cdots u_{i_k}$ is strictly smaller than the length of w . Therefore, after a finite number of recursions, the fourth argument passed to $isMember$ becomes the empty word and the algorithm terminates. ■

Properties of $isMember$

Algorithm 2 is at least exponential in a polynomial of the length of the input word. This is due to the cardinality of \mathcal{S} and the **for**-loop on lines 10 to 12. Let κ be the number of different cells $p \in \mathfrak{P}$ that occur in $\sigma_1 \cdots \sigma_\ell$, and let each symbol occurs at most r times. Both κ and r have upper bound ℓ . Let m be the dimension of \mathfrak{P} . Then there are at most $\kappa \cdot r^{m-1} \leq \ell^m$ values of I considered in the **for**-loop on lines 5 to 9. Since $\ell < |w|$, we execute this **for**-loop at most $|w|^\ell$ times. Hence, \mathcal{S} has cardinality at most $|w|^\ell$. Therefore, the **for**-loop on lines 10 to 12 considers $2^{|\mathcal{S}|} \leq 2^{|w|^\ell}$ different values of J in the worst case.

Let us now turn to the modification of $isMember$ we will use in Section 5. Let $u \in D(\Sigma)$ and $(\sigma_1 u'_1 \bar{\sigma}_1, \dots, \sigma_\ell u'_\ell \bar{\sigma}_\ell) = split(u)$. For every $\sigma \in \Sigma$, we define $occ_\sigma u = |\{i \in [\ell] \mid \sigma_i = \sigma\}|$. Furthermore for every $p \in \mathfrak{P}$, we define $occ_p u = \max\{occ_\sigma u \mid \sigma \in p\}$ and we define

Table 2: Run of Algorithm 2 on the word $[(\][(\][(\][(\]$.

```

isMember( $\Sigma, \mathfrak{P}, [( \ ][( \ ][( \ ][( \ ]$ )
line 3:  $\ell = 4, \sigma_1 = [ = \sigma_3, \sigma_2 = [ = \sigma_4, u_1 = (, u_2 = \varepsilon = u_3, u_4 = \langle$ )
line 5:  $\mathcal{S} = \emptyset, k = 2, i_1 = 1, i_2 = 2$ 
line 6: isMember( $\Sigma, \mathfrak{P}, ()$ )
    line 3:  $\ell = 1, \sigma_1 = (, u_1 = \varepsilon$ 
    line 9:  $\mathcal{S} = \emptyset$ 
    line 10:  $J = \emptyset$ 
    line 13: return False
line 8:  $\mathcal{S} = \emptyset$ 
line 5:  $k = 2, i_1 = 1, i_2 = 4$ 
line 6: isMember( $\Sigma, \mathfrak{P}, ()\langle$ )
    line 3:  $\ell = 2, \sigma_1 = (, \sigma_2 = \langle, u_1 = \varepsilon = u_2$ 
    line 5:  $\mathcal{S} = \emptyset, k = 2, i_1 = 1, i_2 = 2$ 
    line 6: isMember( $\Sigma, \mathfrak{P}, \varepsilon$ )
        line 3:  $\ell = 0$ 
        line 9:  $\mathcal{S} = \emptyset$ 
        line 10:  $J = \emptyset$ 
        line 11: return True
    line 7:  $\mathcal{S} = \{\{1, 2\}\}$ 
    line 10:  $J = \emptyset$ 
    line 10:  $J = \{\{1, 2\}\}$ 
    line 13: return True
line 7:  $\mathcal{S} = \{\{1, 4\}\}$ 
line 5:  $k = 2, i_1 = 2, i_2 = 3$ 
line 6: isMember( $\Sigma, \mathfrak{P}, \varepsilon$ )
    line 3:  $\ell = 0$ 
    line 10:  $\mathcal{S} = \emptyset, J = \emptyset$ 
    line 11: return True
line 7:  $\mathcal{S} = \{\{1, 4\}, \{2, 3\}\}$ 
line 5:  $k = 2, i_1 = 3, i_2 = 4$ 
line 6: isMember( $\Sigma, \mathfrak{P}, \langle$ )
    line 3:  $\ell = 1, \sigma_1 = \langle, u_1 = \varepsilon$ 
    line 10:  $\mathcal{S} = \emptyset, J = \emptyset$ 
    line 13: return False
line 8:  $\mathcal{S} = \{\{1, 4\}, \{2, 3\}\}$ 
line 10:  $J = \emptyset$ 
line 10:  $J = \{\{1, 4\}\}$ 
line 10:  $J = \{\{2, 3\}\}$ 
line 10:  $J = \{\{1, 4\}, \{2, 3\}\}$ 
line 11: return True

```

$\text{occ}_{\mathfrak{P}} u = \sum_{p \in \mathfrak{P}} \text{occ}_p u$. We call a word $w \in (\Sigma \cup \Sigma)^*$ \mathfrak{P} -simple if $w \notin D(\Sigma)$; or $\text{occ}_p w \leq 1$ for each cell $p \in \{p' \in \mathfrak{P} \mid |p'| \geq 2\}$ and $v_1 \cdots v_\ell$ is \mathfrak{P} -simple whenever there are $u_0, \dots, u_\ell \in D(\Sigma)$, $v_1, \dots, v_\ell \in D(\Sigma)$, and $p = \{\sigma_1, \dots, \sigma_\ell\} \in \mathfrak{P}$ with $w = u_0 \sigma_1 v_1 \bar{\sigma}_1 u_1 \cdots \sigma_\ell v_\ell \bar{\sigma}_\ell u_\ell$. In other words, w is \mathfrak{P} -simple if, whenever the cancellation rule can be applied to w to cancel an occurrence of the cell p (where p has more than one element), then there is only one such occurrence and the string $v_1 \cdots v_\ell$ (from the definition of the cancellation rule) is also \mathfrak{P} -simple.

In order for *isMember* to recognise w only if it is \mathfrak{P} -simple, we check between lines 1 and 2 in the algorithm whether $\text{occ}_p w \leq 1$ for each cell $p \in \{p' \in \mathfrak{P} \mid |p'| \geq 2\}$. If this is the case, we continue, otherwise we return False. Let us call the function obtained in this manner *isMember'*. Note that the check between lines 1 and 2 can be done in time linear in the input word. Then the I 's that Algorithm 2 considers in the **for**-loop on lines 5 to 9 are pairwise disjoint. This means that each u_i (for $i \in [\ell]$) occurs in at most one recursive call on line 6. Then the elements of \mathcal{S} are always pairwise disjoint and we only need to consider $J = \mathcal{S}$ in the **for**-loop on lines 10 to 12. We can decide in time $\mathcal{O}(\ell)$ whether \mathcal{S} is a partition of $[\ell]$. Lines 2 and 3 can be computed in time $\mathcal{O}(|w|)$. Since $\ell < |w|$, we know that for each call of *isMember*, we have to invest time linear in the length of the third argument. The maximum depth of recursion is $|w|/2$ because the third argument in the call on line 6 has at most length $|w| - 2$. For every recursion depth, the sum of the lengths of all third arguments is at most $|w|$ because u_i (for $i \in [\ell]$) occurs in at most one recursive call on line 6. Therefore *isMember'* $(\Sigma, \mathfrak{P}, w)$ can be calculated in time $\mathcal{O}(|w|^2)$.

4.2 A CS representation using congruence multiple Dyck languages

Definition 4.9. Let $G = (N, \Gamma, S, P)$ be an MCFG. The *congruence multiple Dyck language with respect to G* , denoted by $mD_c(G)$, is $mD_c(\Sigma, \mathfrak{P})$ where \mathfrak{P} is the smallest set such that

- $p_\gamma = \{\llbracket \gamma \rrbracket\} \in \mathfrak{P}$ for every $\gamma \in \Gamma$,
- $p_\rho = \{\llbracket \rho \rrbracket^j \mid j \in [\text{fan-out}(\rho)]\} \in \mathfrak{P}$ for every $\rho \in P$, and
- $p_{\rho,i} = \{\llbracket \rho \rrbracket_{\rho,i}^j \mid j \in [\text{fan-out}_i(\rho)]\} \in \mathfrak{P}$ for every $\rho \in P$ and $i \in [\text{rank}(\rho)]$,

and $\Sigma = \bigcup_{p \in \mathfrak{P}} p$. □

Lemma 4.10. $R(G) \cap mD(G) = R(G) \cap mD_c(G)$ for each MCFG G .

Proof. It follows from Definitions 3.5 and 4.9 and Proposition 4.5 that $mD(G) \subseteq mD_c(G)$ and hence $R(G) \cap mD(G) \subseteq R(G) \cap mD_c(G)$. It remains to be shown that $R(G) \cap mD_c(G) \subseteq mD(G)$. Let $G = (N, \Gamma, S, P)$ and Σ and Δ be defined as in Definitions 3.5 and 4.9.

We prove the following statement by induction on the length of $w_1 \cdots w_\ell$: If $B \in N$ and $w_1, \dots, w_\ell \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_c$ and w_κ is recognised along a path from $B^{[\kappa]}$ to T in $\mathcal{M}(G)$ for every $\kappa \in [\ell]$, then (w_1, \dots, w_ℓ) can be generated by A_ℓ in $G_\Delta^{\text{rank}(G)}$.

By setting $\ell = 1$ and $B = S$, this statement implies our claim. Now let $B \in N$ and $w_1, \dots, w_k \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_c$ and w_κ is recognised along a path from $B^{[\kappa]}$ to T in $\mathcal{M}(G)$ for every $\kappa \in [\ell]$. By the definitions of $\mathcal{M}(G)$ and $mD_c(G)$, we know that there is some production $\rho = B \rightarrow f(B_1, \dots, B_k) \in P$ with

$$f = [u_{1,0} x_{i(1,1)}^{j(1,1)} u_{1,1} \cdots x_{i(1,p_1)}^{j(1,p_1)} u_{1,p_1}, \dots, u_{\ell,0} x_{i(\ell,1)}^{j(\ell,1)} u_{\ell,1} \cdots x_{i(\ell,p_\ell)}^{j(\ell,p_\ell)} u_{\ell,p_\ell}]$$

such that for every $\kappa \in [\ell]$ either

- (i) $w_\kappa = \llbracket_\rho^{[\kappa]} \tilde{u}_{\kappa,0} \rrbracket_\rho^{[\kappa]}$ or
- (ii) $w_\kappa = \llbracket_\rho^{[\kappa]} \tilde{u}_{\kappa,0} \llbracket_{\rho,i(\kappa,1)}^{[j(\kappa,1)]} v_{i(\kappa,1)}^{j(\kappa,1)} \rrbracket_{\rho,i(\kappa,1)}^{[j(\kappa,1)]} \tilde{u}_{\kappa,1} \cdots \llbracket_{\rho,i(\kappa,p_\kappa)}^{[j(\kappa,p_\kappa)]} v_{i(\kappa,p_\kappa)}^{j(\kappa,p_\kappa)} \rrbracket_{\rho,i(\kappa,p_\kappa)}^{[j(\kappa,p_\kappa)]} \tilde{u}_{\kappa,p_\kappa} \rrbracket_\rho^{[\kappa]}$,

and $v_i^j \in D(\Sigma)$ is recognised along a path from $B_i^{[j]}$ to T in $\mathcal{M}(G)$ for every $i \in [k]$ and $j \in [\text{sort}(B_i)]$, and $v_i^1 \cdots v_i^{\text{sort}(B_i)} \in mD_c(G)$ for every $i \in [k]$. Then by induction hypothesis $(v_i^1, \dots, v_i^{\text{sort}(B_i)})$ can be generated from $A_{\text{sort}(B_i)}$ in $G_\Delta^{\text{rank}(G)}$ for every $i \in [k]$. Using productions of types (ii) and (iii) (cf. Definition 3.5), $A_{\text{sort}(B_1)}, \dots, A_{\text{sort}(B_k)}$ can generate tuples that together have exactly the components

$$\begin{aligned} & \tilde{u}_{1,0} \llbracket_{\rho,i(1,1)}^{[j(1,1)]} v_{i(1,1)}^{j(1,1)} \rrbracket_{\rho,i(1,1)}^{[j(1,1)]} \tilde{u}_{1,1}, \llbracket_{\rho,i(1,2)}^{[j(1,2)]} v_{i(1,2)}^{j(1,2)} \rrbracket_{\rho,i(1,2)}^{[j(1,2)]} \tilde{u}_{1,2}, \dots, \\ & \llbracket_{\rho,i(1,p_1)}^{[j(1,p_1)]} v_{i(1,p_1)}^{j(1,p_1)} \rrbracket_{\rho,i(1,p_1)}^{[j(1,p_1)]} \tilde{u}_{1,p_1}, \dots, \\ & \tilde{u}_{\ell,0} \llbracket_{\rho,i(\ell,1)}^{[j(\ell,1)]} v_{i(\ell,1)}^{j(\ell,1)} \rrbracket_{\rho,i(\ell,1)}^{[j(\ell,1)]} \tilde{u}_{\ell,1}, \llbracket_{\rho,i(\ell,2)}^{[j(\ell,2)]} v_{i(\ell,2)}^{j(\ell,2)} \rrbracket_{\rho,i(\ell,2)}^{[j(\ell,2)]} \tilde{u}_{\ell,2}, \dots, \\ & \llbracket_{\rho,i(\ell,p_\ell)}^{[j(\ell,p_\ell)]} v_{i(\ell,p_\ell)}^{j(\ell,p_\ell)} \rrbracket_{\rho,i(\ell,p_\ell)}^{[j(\ell,p_\ell)]} \tilde{u}_{\ell,p_\ell}. \end{aligned}$$

Set $w'_1, \dots, w'_\ell \in D(\Sigma)$ such that $w_\kappa = \llbracket_\rho^{[\kappa]} w'_\kappa \rrbracket_\rho^{[\kappa]}$ for every $\kappa \in [\ell]$. Then we can derive (w'_1, \dots, w'_ℓ) from A_ℓ in $G_\Delta^{\text{rank}(G)}$ by first using a production

of type (i) with rank exactly $\text{rank}(\rho)$, and for each $\kappa \in [\ell]$ where w_κ has the form $\llbracket \tilde{u}_{\kappa,0} \rrbracket_\rho^{[\kappa]}$ productions of type (iii). Using a production of type (ii), we finally obtain (w_1, \dots, w_ℓ) from A_ℓ in $G_\Delta^{\text{rank}(G)}$. ■

Theorem 4.11. For every multiple context-free languages L of fan-out at most k , there exist a weighted homomorphism h , a regular language R , and a congruence multiple Dyck language mD_c of dimension at most k such that $L = h(R \cap mD_c)$.

Proof. This follows immediately from Lemma 4.10 and Theorem 3.12 when taking $h = \text{weights}_G \circ \text{hom}_{G_\mathbb{B}}$, $R = R(G_\mathbb{B})$, and $mD_c = mD_c(G_\mathbb{B})$. ■

5 n -BEST PARSING FOR WEIGHTED MCFGs USING A CHOMSKY-SCHÜTZENBERGER REPRESENTATION

In this section we describe an n -best parsing algorithm (cf. Huang and Chiang 2005; Büchse *et al.* 2010) for a subset of weighted MCFGs, i.e. we want to find the best parses of a given word in a weighted MCFG. In our case, “parse” refers to a derivation in the weighted MCFG. We formalise our intuition of when a derivation is “better” than another derivation by means of a partial order \preceq on the weights. That is, if we have two derivations d_1 and d_2 with weights μ_1 and μ_2 , respectively, we call d_1 “not worse than” d_2 if $\mu_1 \preceq \mu_2$. Note that we think of weights as *costs* here, i.e. better derivations will get a smaller weight with respect to \preceq . We define the *irreflexive relation with respect to* \preceq as $\triangleleft = \preceq \setminus \{(a, a) \mid a \in A\} \subseteq A \times A$. We say that \mathcal{A} *respects* \preceq if \cdot has the following three properties:

- (i) it is *(strictly) increasing*, i.e. $a \triangleleft a \cdot b$ for every $a, b \in \mathcal{A} \setminus \{0, 1\}$,
- (ii) it has *arbitrarily large powers*, i.e. for every $a, b \in \mathcal{A} \setminus \{0, 1\}$ there is a $k \in \mathbb{N}$ with $b \preceq a^k$ where $a^k = a^{k-1} \cdot a$ for $k \geq 1$ and $a^0 = 1$, and
- (iii) it is *monotone*, i.e. $a \preceq b$ implies $a \cdot c \preceq b \cdot c$ for every $a, b, c \in \mathcal{A}$.

For the rest of this section let argmin^\preceq be a function that assigns for every family $f: B \rightarrow \mathcal{A}$ a value $\bar{b} \in B$ such that there is no $b' \in B$ with $f(b') \triangleleft \bar{b}$; we write $\text{argmin}_{b \in B}^\preceq (f(b))$ for \bar{b} .

Note that we only need multiplication to obtain the weight of a derivation and therefore the sum operation of our complete commutative strong bimonoid becomes irrelevant within this section.

5.1 n -best parsing

We will take the n best parses from the possibly infinite sequence of derivations for some word in an MCFG. We therefore need a notion of infinite strings.

Definition 5.1. Let B be a set. The set of infinite strings over B , denoted by B^ω , is the set of partial functions $u: \mathbb{N} \setminus \{0\} \rightarrow B$ where the fact that $u(n)$ is defined implies that $u(n-1)$ is defined as well, for every $n > 1$. \square

Every element u of B^* can be construed as an element of B^ω where $\text{dom}(u)$ is finite. Let $u \in B^*$ and $v \in B^\omega$. The concatenation of u and v , denoted by $u \cdot^\omega v$, is given by

$$(u \cdot^\omega v)(n) = \begin{cases} u(n) & \text{if } n \leq |u| \\ v(n - |u|) & \text{otherwise.} \end{cases}$$

To work with infinite strings, we define the following functions:
`map` applies a function to every element in an infinite list.

$$\begin{aligned} \text{map}: (B \rightarrow C) &\rightarrow (B^\omega \rightarrow C^\omega) \\ \text{map}(f)(bu) &= f(b) \cdot^\omega \text{map}(f)(u) && \text{(if } b \in B) \\ \text{map}(f)(\varepsilon) &= \varepsilon \end{aligned}$$

`take` returns a finite prefix of a given infinite list.

$$\begin{aligned} \text{take}: \mathbb{N} &\rightarrow (B^\omega \rightarrow B^*) \\ \text{take}(n)(bu) &= b \cdot^\omega \text{take}(n-1)(u') && \text{(if } n > 0 \text{ and } b \in B) \\ \text{take}(n)(u) &= \varepsilon && \text{(if } n = 0 \text{ or } u = \varepsilon) \end{aligned}$$

`filter` removes elements that are not in a given set from an infinite list.

$$\begin{aligned} \text{filter}: \mathcal{P}(B) &\rightarrow (B^\omega \rightarrow B^\omega) \\ \text{filter}(B')(b'u) &= b' \cdot^\omega \text{filter}(B')(u) && \text{(if } b' \in B') \\ \text{filter}(B')(bu) &= \text{filter}(B')(u) && \text{(if } b \notin B') \\ \text{filter}(B')(\varepsilon) &= \varepsilon \end{aligned}$$

sort returns an infinite list that contains each element of a given set exactly once in an order that respects \trianglelefteq .

$$\begin{aligned} \text{sort}: (B \rightarrow \mathcal{A}) \times \mathcal{P}(\mathcal{A} \times \mathcal{A}) &\rightarrow (\mathcal{P}(B) \rightarrow B^\omega) \\ \text{sort}(f, \trianglelefteq)(B') &= \text{argmin}_{b \in B'}^{\trianglelefteq}(f(b)) \\ &\cdot^\omega \text{sort}(f, \trianglelefteq)(B' \setminus \text{argmin}_{b \in B'}^{\trianglelefteq}(f(b))) \quad (\text{if } B' \neq \emptyset) \\ \text{sort}(f, \trianglelefteq)(\emptyset) &= \varepsilon \end{aligned}$$

Definition 5.2. Let B be a set, $f: B \rightarrow \mathcal{A}$, $n \in \mathbb{N}$, and \trianglelefteq be a partial order on \mathcal{A} . We define the n -best function with respect to f and \trianglelefteq as a function $n\text{-best}(f, \trianglelefteq): \mathcal{P}(B) \rightarrow \mathcal{P}(B^*)$ where for every $B' \subseteq B$ we have that $(b_1, \dots, b_k) \in n\text{-best}(f, \trianglelefteq)(B')$ if the following conditions hold

- (i) $k = \min\{n, |B'|\}$,
- (ii) $b_1, \dots, b_k \in B'$ are pairwise different,
- (iii) $f(b_1) \trianglelefteq f(b_2) \trianglelefteq \dots \trianglelefteq f(b_k)$, and
- (iv) there is no $b \in B' \setminus \{b_1, \dots, b_k\}$ with $f(b) \triangleleft f(b_k)$. □

Note that $|n\text{-best}(f, \trianglelefteq)(B')| = 1$ if $n \leq |B'|$.

Definition 5.3. The parsing problem for \mathcal{A} -weighted MCFG is:

- given** an \mathcal{A} -weighted MCFG $G = (N, \Sigma, S, P, \mu)$, a partial order \trianglelefteq on \mathcal{A} , a word $w \in \Sigma^*$, and an integer $n \in \mathbb{N}$,
- output** an element of $n\text{-best}(\mu, \trianglelefteq)(D_G(w))$. □

The following observation is apparent from the definitions of sort, take, and n -best.

Observation 5.4. $\text{take}(n) \circ \text{sort}(f, \trianglelefteq)(B') \in n\text{-best}(f, \trianglelefteq)(B')$ for every set B , subset $B' \subseteq B$, function $f: B \rightarrow \mathcal{B}$, $n \in \mathbb{N}$, and partial order \trianglelefteq on \mathcal{B} . ■

5.2 Specification of the CS parser

Given an \mathcal{A} -weighted MCFG G , we construct the regular language $R(G_{\mathbb{B}})$, the \mathcal{A} -weighted homomorphism $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$, and the congruence multiple Dyck language $mD_c(G_{\mathbb{B}})$. Then $L(G) = h(R(G_{\mathbb{B}}) \cap mD_c(G_{\mathbb{B}}))$. In a more procedural view on this representation of G , we might say that one obtains words (with weights) in $L(G)$ by (i) generating words in $R(G_{\mathbb{B}})$ (called *candidates*), (ii) discarding the candidates that are not in $mD_c(G_{\mathbb{B}})$, and (iii) applying h to the remaining candidates.

Recall that there is a bijection between D_G and $R(G_{\mathbb{B}}) \cap mD_c(G_{\mathbb{B}})$ (Corollary 3.13). Now let w be the word we want to parse. Furthermore, let $R_{h,w}$ be the set of words that h maps to w . Then there is a bijection between $R_{h,w} \cap R(G_{\mathbb{B}}) \cap mD_c(G_{\mathbb{B}})$ and $D_G(w)$. Our strategy to compute the n best derivations is to enumerate the words in $R_{h,w} \cap R(G_{\mathbb{B}})$ in the order given by the weights defined by the homomorphism h and then checking whether each word is in $mD_c(G_{\mathbb{B}})$ until we have found n words. However, this approach will have to be refined to ensure termination of the parsing algorithm.

First we show that the set $R_{h,w}$ of words mapped by h to w is regular.

Definition 5.5. Let $h: \Sigma^* \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ be an \mathcal{A} -weighted alphabetic homomorphism and $\gamma_1, \dots, \gamma_n \in \Gamma$. Furthermore, let $\Sigma_\varepsilon = \{\sigma \in \Sigma \mid h(\sigma) = \mu.\varepsilon \text{ for some } \mu \in \mathcal{A}\}$, and for every $\gamma \in \Gamma$, define $\sigma_\gamma \in \Sigma$ such that $h(\sigma_\gamma) = \mu.\gamma$ for some $\mu \in \mathcal{A}$. The domain language of h with respect to $\gamma_1 \cdots \gamma_n$, is the language

$$R_{h,\gamma_1 \cdots \gamma_n} = \{u_0 \sigma_{\gamma_1} u_1 \cdots \sigma_{\gamma_n} u_n \mid u_0, \dots, u_n \in \Sigma_\varepsilon^*\}. \quad \square$$

Lemma 5.6. $R_{h,w}$ is a regular language for every \mathcal{A} -weighted alphabetic homomorphism $h: \Sigma^* \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ and word $w \in \Gamma^*$.

Proof. Since $\{\sigma_{\gamma_1}\}, \dots, \{\sigma_{\gamma_n}\}$, and Σ_ε are finite sets, $R_{h,w} = \Sigma_\varepsilon^* \cdot \{\sigma_{\gamma_1}\} \cdot \Sigma_\varepsilon^* \cdot \dots \cdot \{\sigma_{\gamma_n}\} \cdot \Sigma_\varepsilon^*$, and because finite sets are recognisable (Hopcroft and Ullman 1969, Theorem 3.7) and recognisable languages are closed under language concatenation (Hopcroft and Ullman 1969, Theorem 3.8) and Kleene-star (Hopcroft and Ullman 1969, Theorem 3.9), we have that $R_{h,w}$ is regular. \blacksquare

Example 5.7 (Examples 3.7 and 3.11 continued). Figure 6 shows a deterministic FSA $\mathcal{M}_{h,ac}$ that recognises $R_{h,ac}$. \square

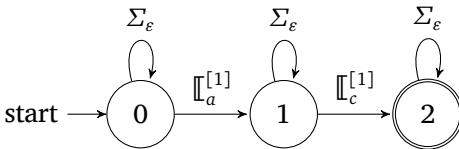


Figure 6:
Deterministic FSA $\mathcal{M}_{h,ac}$ that recognises $R_{h,ac}$
(cf. Example 5.7)

Definition 5.8. Let $\mathcal{M} = (Q, \Delta, q_i, F, T)$ be an FSA and $h: \Delta^* \rightarrow (\Gamma^* \rightarrow \mathcal{A})$ be an \mathcal{A} -weighted homomorphism. The set of harmful loops in \mathcal{M} with respect to h is the set of all runs $q_0 u_1 q_1 \cdots u_k q_k$ in \mathcal{M} where q_1, \dots, q_k are pairwise disjoint elements of Q , $q_0 = q_k$, $u_1, \dots, u_k \in \Delta^*$, and $h(u_1) = \dots = h(u_k) = 1.\varepsilon$. \square

When examining Example 3.11 and Figure 5, we see that there are seven harmful loops in $\mathcal{M}(G_{\mathbb{B}})$ with respect to h : the five self-loops of T , the loop between T and $A^{[2]}$, and the loop between T and $B^{[2]}$. Harmful loops are problematic for the termination of a parsing algorithm since they cause an infinite set of candidates that are not distinguishable by their image under h . When generating the sequence of candidates for our parsing algorithm, such a situation creates the contingency of never producing a candidate that is accepted by $mD_c(G_{\mathbb{B}})$ even if one exists. To allow our parsing algorithm to solve the above problem, we will

- (i) only admit a restricted form of weighted MCFGs and
- (ii) require each value in the domain of \mathcal{A} (except 0 and 1) to be viewed as a product of arbitrarily many smaller values from the domain of \mathcal{A} .

Restricted weighted MCFG

Definition 5.9. An \mathcal{A} -weighted MCFG $G = (N, \Delta, S, P, \mu)$ is called *restricted* if there do not exist a subderivation d in G and a position $\pi = n_1 \cdots n_k \in \text{pos}(d)$ (where $n_1, \dots, n_k \in \mathbb{N}$) such that $d(\varepsilon) = d(\pi)$, $d(\varepsilon), d(n_1), d(n_1 n_2), \dots, d(n_1 \cdots n_k)$ are pairwise different, and $\mu(d(\varepsilon)) = \mu(d(n_1)) = \mu(d(n_1 n_2)) = \dots = \mu(d(n_1 \cdots n_k)) = 1$. \square

Restricted weighted MCFG are strictly less powerful than (unrestricted) weighted MCFG, as the next example shows.

Example 5.10. Let us consider an arbitrary \mathbb{B} -weighted MCFG G and let m be the number of rules in G . Assume that $L(G)$ is not finite. Then there are subderivations in G of arbitrary height. It is clear that every subderivation d in G with a height greater than $m + 1$ must have a position $\pi \in \text{pos}(d)$ such that $d(\varepsilon) = d(\pi)$. Then, since G has weights from \mathbb{B} , we know that 1 is assigned to every production in G and thus G is not restricted. \square

Restricted weighted MCFGs are still useful in practice, as the following two observations show.

Definition 5.11. An \mathcal{A} -weighted MCFL is called *proper* if \mathcal{A} is the probability semiring, the Viterbi semiring, or one of the algebras Pr_1 or Pr_2 (cf. Example 2.1) and for each non-terminal A the sum (using the usual addition in \mathbb{R}) of the weights of all productions with left-hand side A is 1. \square

Observation 5.12. Every proper weighted MCFG is restricted.

Proof. Assume that G is proper but not restricted. Then there is a subderivation d in G and a position $\pi \in \text{pos}(d)$ such that the weights of all productions along the path from the root to position π in d are 1 and $d(\varepsilon) = d(\pi) = \rho$. All productions along the path from the root to position π are unique for their respective left-hand side non-terminals since G is probabilistic. This means that every subderivation d' starting from ρ has the position π and $\rho = d'(\varepsilon) = d'(\pi) = d(\varepsilon) = d(\pi)$. But then $\{\varepsilon, \pi, \pi\pi, \pi\pi\pi, \dots\} \subseteq \text{pos}(d)$ and hence d is not a (finite) term, which contradicts our definition of a subderivation. \blacksquare

If we extract a weighted MCFG from a corpus and assign the weights by maximum-likelihood estimation (as for example in Kallmeyer and Maier 2013, p. 107), then we will get a weighted MCFG that is proper and therefore restricted.

The next observation allows us to enrich the weight structure of a \mathbb{B} -weighted MCFG to make it suitable for CS-parsing.

Observation 5.13. For every \mathbb{B} -weighted MCFG G , there is a restricted weighted MCFG G' such that $\text{supp}(L(G)) = \text{supp}(L(G'))$.

Proof. This can be done by assigning to every $w \in \text{supp}(G)$, the size (i.e. number of productions) of its smallest derivation. To achieve that, we choose the tropical semiring as weight algebra for G' , use the productions from G , and give every production the weight 1. Then no production in G' has the semiring-1 (which is 0 for the tropical semiring) as its weight and G' is restricted. \blacksquare

Factorisable weight structures

Definition 5.14. Let \mathcal{A} be a complete commutative strong bimonoid and \leq be a partial order on \mathcal{A} . We say that \mathcal{A} is \leq -factorisable if for

every $a \in \mathcal{A} \setminus \{0, 1\}$ and natural number $k \geq 2$, there are $a_1, \dots, a_k \in \mathcal{A} \setminus \{0, 1\}$ such that $a_1 \triangleleft a, \dots, a_k \triangleleft a$ and $a_1 \cdot \dots \cdot a_k = a$. We then call $a_1 \cdot \dots \cdot a_k$ a (\trianglelefteq, k) -factorisation of a . \square

Some of the complete commutative strong bimonoids mentioned in Example 2.1 are \trianglelefteq -factorisable for some suitable partial order \trianglelefteq , as Table 3 shows. The examples where multiplication is idempotent, however, have no suitable partial order since $a \cdot a = a$ contradicts \cdot being increasing (in particular $a \triangleleft a \cdot a$).

Table 3: List of some complete commutative strong bimonoids \mathcal{A} from Example 2.1 together with a partial order \trianglelefteq that \mathcal{A} respects and a (\trianglelefteq, k) -factorisation.	example algebra ($\mathcal{A}, +, \cdot, 0, 1$)	partial order \trianglelefteq	(\trianglelefteq, k) -factorisation of $a \in \mathcal{A} \setminus \{0, 1\}$
	Viterbi semiring ($[0, 1], \max, \cdot, 0, 1$)	\geq	$\underbrace{\sqrt[k]{a} \cdot \dots \cdot \sqrt[k]{a}}_{k \text{ times}}$
	tropical semiring ($\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +, \infty, 0$)	\leq	$\underbrace{a/k + \dots + a/k}_{k \text{ times}}$
	arctic semiring ($\mathbb{R}_{\geq 0} \cup \{-\infty\}, \max, +, -\infty, 0$)	\leq	$\underbrace{a/k + \dots + a/k}_{k \text{ times}}$
	$\text{Pr}_1 = ([0, 1], \oplus_1, \cdot, 0, 1)$ and $\text{Pr}_2 = ([0, 1], \oplus_2, \cdot, 0, 1)$	\geq	$\underbrace{\sqrt[k]{a} \cdot \dots \cdot \sqrt[k]{a}}_{k \text{ times}}$

For the probability semiring $\text{Pr} = (\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ monotonicity and increasingness contradict each other. To show this we first assume that Pr would respect some partial order \trianglelefteq . From $1/2 \cdot 1/2 = 1/4$ and $2 \cdot 2 = 4$ follows $1/2 \triangleleft 1/4$ and $2 \triangleleft 4$ since \cdot is increasing. By monotonicity then follows $2 \cdot 1/2 \trianglelefteq 4 \cdot 1/2 \trianglelefteq 4 \cdot 1/4$ and hence $1 \trianglelefteq 2 \trianglelefteq 1$. But \trianglelefteq is a partial order and thus antisymmetric.

5.4 A CS parsing algorithm

For the remainder of this article let \mathcal{A} be a complete commutative strong bimonoid and \trianglelefteq be a partial order on \mathcal{A} such that \mathcal{A} respects \trianglelefteq and is \trianglelefteq -factorisable. Also, we will require our weighted MCFGs to be restricted.

To solve the problem stated in the beginning of Section 5.3, we define a modified generator language and a modified weight function to replace R and h from the CS-theorem (cf. Theorem 3.12).

The modified generator language

Consider an MCFG G with non-terminals N . Then, intuitively, $\mathcal{M}(G)$ has two kinds of states:

- (i) For every non-terminal $A \in N$ and every $j \in [\text{fan-out}(A)]$ there is a state $A^{[j]}$ that signifies that the automaton *is about to process* the j -th component of a string tuple generated by G with A .
- (ii) There is a state T that signifies that the automaton *just finished processing* some component of a string tuple generated by G with some non-terminal.

We split state T from $\mathcal{M}(G)$ up to formalise the following intuition:

- (ii') For every non-terminal $A \in N$ and every $j \in [\text{fan-out}(A)]$ there is a state $\bar{A}^{[j]}$ that signifies that the automaton *just finished processing* the j -th component of a string tuple generated by G with A .

Definition 5.15. Let $G = (N, \Delta, S, R)$ be an MCFG. The *modified generator automaton with respect to G* , denoted by $\mathcal{M}'(G)$, is the finite state automaton $(Q \cup \bar{Q}, \Sigma, S^{[1]}, \{\bar{S}^{[1]}\}, \tau)$ where Σ is the generator alphabet with respect to G , $Q = \{A^{[j]} \mid A \in N, j \in [\text{fan-out}(A)]\}$, $\bar{Q} = \{\bar{A}^{[j]} \mid A \in N, j \in [\text{fan-out}(A)]\}$, and τ is the smallest set such that for every rule

$$\rho = A \rightarrow [u_1^0 y_1^1 u_1^1 \cdots y_1^{n_1} u_1^{n_1}, \dots, u_s^0 y_s^1 u_s^1 \cdots y_s^{n_s} u_s^{n_s}](B_1, \dots, B_m)$$

in R we have that

- (i) $(A^{[j]}, \llbracket_{\rho}^{[j]} \tilde{u}_j^0 \rrbracket_{\rho}^{[j]}, \bar{A}^{[j]}) \in \tau$ for every $j \in [s]$ with $n_j = 0$,
- (ii) $(A^{[j]}, \llbracket_{\rho}^{[j]} \tilde{u}_j^0 \rrbracket_{\rho, i}^{[\ell]}, B_i^{[\ell]}) \in \tau$ for every $i \in [m]$, $j \in [s]$, and $\ell \in [\text{fan-out}(B_i)]$ with $n_j \neq 0$ and $y_j^1 = x_i^{\ell}$,
- (iii) $(\bar{B}_i^{[\ell]}, \llbracket_{\rho, i}^{[\ell]} \tilde{u}_j^{\kappa-1} \rrbracket_{\rho, i'}^{[\ell]}, B_{i'}^{[\ell]}) \in \tau$ for every $i, i' \in [m]$, $j \in [s]$, $\kappa \in [n_j]$, $\ell \in [\text{fan-out}(B_i)]$, $\ell' \in [\text{fan-out}(B_{i'})]$ with $n_j \neq 0$, $y_j^{\kappa-1} = x_i^{\ell}$, $y_j^{\kappa} = x_{i'}^{\ell'}$, and
- (iv) $(\bar{B}_i^{[\ell]}, \llbracket_{\rho, i}^{[\ell]} \tilde{u}_j^{n_j} \rrbracket_{\rho}^{[j]}, \bar{A}^{[j]}) \in \tau$ for every $i \in [m]$, $j \in [s]$, and $\ell \in [\text{fan-out}(B_i)]$ with $n_j \neq 0$ and $y_j^{n_j} = x_i^{\ell}$.

The *modified generator language with respect to G* is $R'(G) = L(\mathcal{M}'(G))$. \square

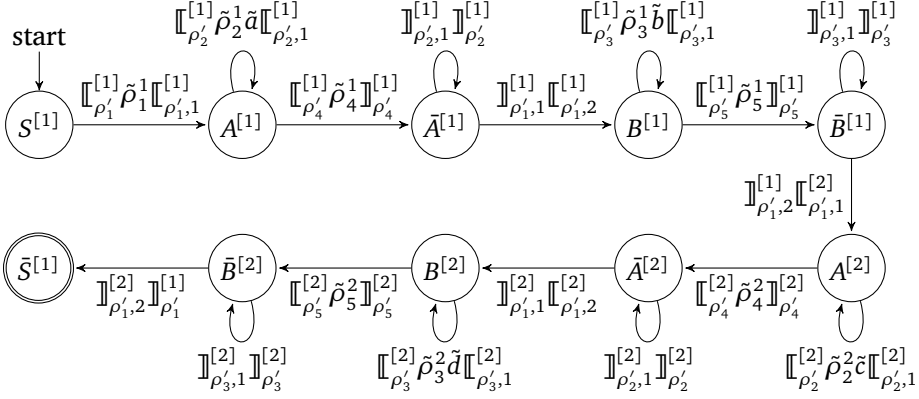


Figure 7: Modified generator automaton $\mathcal{M}'(G)$ with respect to G from Example 3.4

Lemma 5.16. $R(G) \cap mD_c(G) = R'(G) \cap mD_c(G)$ for every MCFG G .

Proof. Let $G = (N, \Delta, S, P)$ and Σ be the generator alphabet with respect to G .

(\supseteq) For this we show that $R(G) \supseteq R'(G)$. Let $q_0 u_1 q_1 \cdots u_m q_m$ be a successful run in $\mathcal{M}'(G)$. We define the string $v = t(q_0) u_1 t(q_1) \cdots u_m t(q_m)$ where

$$t(q) = \begin{cases} q & \text{if } q = A^{[j]} \in Q \text{ for some } A \in N \text{ and } j \in \text{fan-out}(A) \\ T & \text{otherwise.} \end{cases}$$

Clearly for every transition (q, u, q') in $\mathcal{M}'(G)$, there is a transition $(t(q), u, t(q'))$ in $\mathcal{M}(G)$. Since $\bar{S}^{[1]}$ is the final state in $\mathcal{M}'(G)$ and $t(\bar{S}^{[1]})$ is a final state in $\mathcal{M}(G)$, we have that v is a successful run in $\mathcal{M}(G)$.

(\subseteq) Since $R(G) \cap mD_c(G)$ is in bijection with D_G (Corollary 3.9), it suffices to show that $\text{yield}(D_G) \supseteq \text{hom}_G(R'(G))$. We prove the following for every $A \in N$ and $d \in D_G(A)$ by induction on d :

Let $\text{yield}(d) = (u_1, \dots, u_m)$. There are $v_1, \dots, v_m \in \Sigma^*$ such that $\mathcal{M}'(G)$ recognises v_i from $A^{[i]}$ to $\bar{A}^{[i]}$ and $\text{hom}_G(v_i) = u_i$ for every $i \in [m]$.

This statement implies the claim.

Induction base: Let $d = \rho = A \rightarrow [u_1, \dots, u_m] \in P$. Then for every $i \in [m]$, there is a transition $(A^{[i]}, \llbracket_{\rho}^{[i]} \tilde{u}_i \rrbracket_{\rho}^{[i]}, \bar{A}^{[i]})$ in $\mathcal{M}'(G)$. Clearly, $\text{hom}_G(\llbracket_{\rho}^{[i]} \tilde{u}_i \rrbracket_{\rho}^{[i]}) = u_i$ and $\mathcal{M}'(G)$ recognises $\llbracket_{\rho}^{[i]} \tilde{u}_i \rrbracket_{\rho}^{[i]}$ from $A^{[i]}$ to $\bar{A}^{[i]}$.

Step: Let $d = \rho(d_1, \dots, d_k)$ with $\rho = A \rightarrow [u_1, \dots, u_m](B_1, \dots, B_k)$ and $m_i = \text{fan-out}(B_i)$ for every $i \in [k]$. By induction hypothesis there are $v_1^i, \dots, v_{m_i}^i$ for every $i \in [k]$ such that $(\text{hom}_G(v_1^i), \dots, \text{hom}_G(v_{m_i}^i)) = \text{yield}(d_i)$ and $\mathcal{M}'(G)$ recognises v_j^i from $B_i^{[j]}$ to $\bar{B}_i^{[j]}$ for every $j \in [m]$. By the existence of ρ in P and Definition 5.15, we can construct runs in $\mathcal{M}'(G)$ from $A^{[1]}$ to $\bar{A}^{[1]}$, ..., $A^{[m]}$ to $\bar{A}^{[m]}$, recognising v_1, \dots, v_m , respectively such that $(\text{hom}_G(v_1), \dots, \text{hom}_G(v_m)) = \text{yield}(d)$. ■

The modified weight function

Examining $\mathcal{M}'(G_{\mathbb{B}})$ from Figure 7 together with h from Example 3.11 we notice that there are still four harmful loops: the self-loops of $A^{[1]}$, $B^{[1]}$, $A^{[2]}$, and $B^{[2]}$. Therefore, we will define a function $\text{wt}_{G, \trianglelefteq}$ from strings of parentheses to \mathcal{A} that assigns a weight different from 1 to the labels along those loops, but still computes the same weights as h on the subset $mD_c(G_{\mathbb{B}})$. Intuitively, we take the weight attached to a symbol $\llbracket_{\rho}^{[1]}$ (for some rule ρ of fan-out s) and distribute it along the $2 \cdot s$ symbols $\llbracket_{\rho}^{[1]}, \rrbracket_{\rho}^{[1]}, \dots, \llbracket_{\rho}^{[s]}, \rrbracket_{\rho}^{[s]}$ using the fact that \mathcal{A} is \trianglelefteq -factorisable.

Definition 5.17. Let \mathcal{A} be a complete commutative strong bimonoid, \trianglelefteq be a partial order on \mathcal{A} , \mathcal{A} be \trianglelefteq -factorisable, and $G = (N, \Delta, S, P, \mu)$ be an \mathcal{A} -weighted k -MCFG. Furthermore, let Σ be the generator alphabet with respect to G . For every $\rho = A \rightarrow [u_1, \dots, u_s](B_1, \dots, B_k) \in P$, we set $\rho' = A \rightarrow [\rho^1 u_1, \dots, \rho^s u_s](B_1, \dots, B_k)$ and fix values $a_1^{\rho}, \dots, a_{2 \cdot s}^{\rho} \in \mathcal{A} \setminus \{0\}$ such that

- if $\mu(\rho) \neq 1$ then $a_1^{\rho}, \dots, a_{2 \cdot s}^{\rho}$ are not 1, are smaller or equal to $\mu(\rho)$ with respect to \trianglelefteq , and $a_1^{\rho} \cdot \dots \cdot a_{2 \cdot s}^{\rho} = \mu(\rho)$; or
- if $\mu(\rho) = 1$ then $a_1^{\rho} = \dots = a_{2 \cdot s}^{\rho} = 1$.

The *weight function with respect to G and \trianglelefteq* is the function $\text{wt}_{G, \trianglelefteq}: \Sigma^* \rightarrow \mathcal{A}$ defined for every $u_1, \dots, u_k \in \Sigma$ by

$$\text{wt}_{G, \trianglelefteq}(u_1 \cdots u_k) = \text{wt}'_{G, \trianglelefteq}(u_1) \cdot \dots \cdot \text{wt}'_{G, \trianglelefteq}(u_k)$$

where $\text{wt}'_{G, \trianglelefteq}: \Sigma \rightarrow \mathcal{A}$ is given by

$$\text{wt}'_{G, \trianglelefteq}(\sigma) = \begin{cases} a_{2 \cdot i-1}^{\rho} & \text{if } \sigma \text{ is of the form } \llbracket_{\rho}^{[i]} \\ a_{2 \cdot i}^{\rho} & \text{if } \sigma \text{ is of the form } \rrbracket_{\rho}^{[i]} \\ 1 & \text{otherwise.} \end{cases} \quad \square$$

Example 5.18 (Example 2.5 continued). First, we calculate factorisations of the weights in G as shown in Table 3 for Pr_2 :

$$\begin{aligned} \text{for } \rho_2 \text{ and } \rho_4: & \quad 1/2 = \sqrt[4]{1/2} \cdot \sqrt[4]{1/2} \cdot \sqrt[4]{1/2} \cdot \sqrt[4]{1/2} \\ \text{for } \rho_3: & \quad 1/3 = \sqrt[4]{1/3} \cdot \sqrt[4]{1/3} \cdot \sqrt[4]{1/3} \cdot \sqrt[4]{1/3} \\ \text{for } \rho_5: & \quad 2/3 = \sqrt[4]{2/3} \cdot \sqrt[4]{2/3} \cdot \sqrt[4]{2/3} \cdot \sqrt[4]{2/3} \end{aligned}$$

Then $\text{wt}_{G, \preceq}$ is given as follows:

$$\text{wt}_{G, \preceq}(\sigma) = \begin{cases} \sqrt[4]{1/2} & \text{if } \sigma \in \{[\rho'_j, \mathbb{I}^{[j]}_{\rho'_j}] \mid \rho'_j \in \{\rho'_2, \rho'_4\}, j \in [2]\}, \\ \sqrt[4]{1/3} & \text{if } \sigma \in \{[\rho'_j, \mathbb{I}^{[j]}_{\rho'_j}] \mid j \in [2]\}, \\ \sqrt[4]{2/3} & \text{if } \sigma \in \{[\rho'_j, \mathbb{I}^{[j]}_{\rho'_j}] \mid j \in [2]\}, \\ 1 & \text{otherwise.} \end{cases} \quad \square$$

Now we examine the FSA $\mathcal{M}'(G_{\mathbb{B}})$ from Figure 7 again, but this time we use the weight function $\text{wt}_{G, \preceq}$ from Example 5.18 instead of the weights given by the homomorphism h in Example 3.11. For this let $h': (\Sigma \cup \bar{\Sigma})^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ be the \mathcal{A} -weighted homomorphism defined by $h'(\sigma) = \text{wt}_{G, \preceq}(\sigma) \cdot u_\sigma$ where $h(\sigma) = \mu_\sigma \cdot u_\sigma$ for every $\sigma \in \Sigma \cup \bar{\Sigma}$. Then there are no more harmful loops in $\mathcal{M}'(G_{\mathbb{B}})$ with respect to h' .

Lemma 5.19. Let $G = (N, \Delta, S, P, \mu)$ be a restricted \mathcal{A} -weighted MCFG that only has productive non-terminals and \preceq be a partial order on \mathcal{A} such that \mathcal{A} is \preceq -factorisable. Furthermore, let $h': (\Sigma \cup \bar{\Sigma})^* \rightarrow (\Delta^* \rightarrow \mathcal{A})$ be the \mathcal{A} -weighted homomorphism defined by $h'(\sigma) = \text{wt}_{G, \preceq}(\sigma) \cdot u_\sigma$ where $(\text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}})(\sigma) = \mu_\sigma \cdot u_\sigma$ for every $\sigma \in \Sigma \cup \bar{\Sigma}$. Then $\mathcal{M}'(G_{\mathbb{B}})$ has no harmful loops with respect to h' .

Proof. We show the claim by contradiction. For this assume that $q_0 u_1 q_1 \cdots u_k q_k$ is a harmful loop in $\mathcal{M}'(G_{\mathbb{B}})$ with respect to h' where q_0, \dots, q_k are states and $u_1, \dots, u_k \in (\Sigma \cup \bar{\Sigma})^*$. Then $q_0 = q_k$ and $\text{wt}_{G, \preceq}(u_1) = \dots = \text{wt}_{G, \preceq}(u_k) = 1$. We now distinguish two cases:

(Case 1) Let $q_0 = q_k \in Q$ and $I = \{i_0, \dots, i_m\}$ be the maximal subset of $\{0, \dots, k\}$ (with $i_0 < \dots < i_m$) such that

- (i) $i_0 = 0$ and $i_m = k$,
- (ii) for every $i \in I$ we have that q_i is of the form $B_i^{[j_i]}$ for some $B_i \in N$ and $j_i \in [\text{fan-out}(B_i)]$, and

- (iii) for every $\kappa \in [m]$ we have that B_{i_κ} occurs on the right-hand side of $\rho_{i_{\kappa-1}}$ where $\llbracket \rho_{i_{\kappa-1}} \rrbracket^{[j_{\kappa-1}]}$ is read in the transition that leaves $q_{i_{\kappa-1}}$.

Since every non-terminal in G is productive there is a derivation d and a position $\pi = n_1 \cdots n_m$ in d such that $n_1, \dots, n_m \in \mathbb{N}$ and $d(\varepsilon) = \rho_{i_1}$, $d(n_1) = \rho_{i_2}, \dots, d(n_1 \cdots n_{m-1}) = \rho_{i_m}$, and $d(n_1 \cdots n_m) = \rho_{i_1}$. For every $i \in I$ we know that $\mu(\rho_i) = 1$ since $\text{wt}_{G, \leq}(\llbracket \rho_i \rrbracket^{[j_i]}) = 1$ and by Definition 5.17. This contradicts G being restricted.

(Case 2) Let $q_0 = q_k \in \bar{Q}$ and $I = \{i_0, \dots, i_m\}$ be the maximal subset of $\{0, \dots, k\}$ (with $i_m < \dots < i_0$) such that

- (i) $i_m = 0$ and $i_0 = k$,
- (ii) for every $i \in I$ we have that q_i is of the form $B_i \bar{B}_i^{[j_i]}$ for some $B_i \in N$ and $j_i \in [\text{fan-out}(B_i)]$, and
- (iii) for every $\kappa \in [m]$ we have that B_{i_κ} occurs on the right-hand side of $\rho_{i_{\kappa-1}}$ where $\llbracket \rho_{i_{\kappa-1}} \rrbracket^{[j_{\kappa-1}]}$ is read in the transition that reaches $q_{i_{\kappa-1}}$.

Since every non-terminal in G is productive there is a derivation d and a position $\pi = n_1 \cdots n_m$ in d such that $n_1, \dots, n_m \in \mathbb{N}$ and $d(\varepsilon) = \rho_{i_m}$, $d(n_1) = \rho_{i_{m-1}}, \dots, d(n_1 \cdots n_{m-1}) = \rho_{i_1}$, and $d(n_1 \cdots n_m) = \rho_{i_m}$. For every $i \in I$ we know that $\mu(\rho_i) = 1$ since $\text{wt}_{G, \leq}(\llbracket \rho_i \rrbracket^{[j_i]}) = 1$ and by Definition 5.17. This contradicts G being restricted. ■

Lemma 5.20. Let $G = (N, \Delta, S, P, \mu)$ be an \mathcal{A} -weighted MCFG and \leq a partial order on \mathcal{A} . Then $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$ assigns the same weight to each word in $mD_c(G_{\mathbb{B}}) \cap R'(G_{\mathbb{B}})$ as $\text{wt}_{G, \leq}$.

Proof. Let $\rho \in P$ be an arbitrary production, $m = \text{fan-out}(\rho)$, and $\text{wt}_{G, \leq}(\llbracket \rho \rrbracket^{[1]}) = a_{2, i-1}$ and $\text{wt}_{G, \leq}(\llbracket \rho \rrbracket^{[1]}) = a_{2, i}$ for every $i \in [m]$. By the definition of $R'(G_{\mathbb{B}})$, we know that symbols of the forms $\llbracket \rho \rrbracket^{[i]}$, $\llbracket \rho \rrbracket^{[1]}$, and $\llbracket \rho \rrbracket^{[1]}$ occur only as a sequence $\llbracket \rho \rrbracket^{[i]} \llbracket \rho \rrbracket^{[1]} \llbracket \rho \rrbracket^{[1]}$ (see the construction in Lemma 3.3 and items (i) and (ii) in Definition 5.15). By the definition of the cancellation rule, we also know that for every symbol $\llbracket \rho \rrbracket^{[i]}$ there must occur corresponding symbols $\llbracket \rho \rrbracket^{[1]}, \dots, \llbracket \rho \rrbracket^{[i-1]}, \llbracket \rho \rrbracket^{[i+1]}, \dots, \llbracket \rho \rrbracket^{[m]}$ and $\llbracket \rho \rrbracket^{[1]}, \dots, \llbracket \rho \rrbracket^{[m]}$ in $mD_c(G_{\mathbb{B}})$. Thus in the set $mD_c(G_{\mathbb{B}}) \cap R'(G_{\mathbb{B}})$ we have that $\llbracket \rho \rrbracket^{[1]}$ occurs iff all the corresponding symbols $\llbracket \rho \rrbracket^{[2]}, \dots, \llbracket \rho \rrbracket^{[m]}$ and $\llbracket \rho \rrbracket^{[1]}, \dots, \llbracket \rho \rrbracket^{[m]}$ occur.

Then by the construction of $\text{wt}_{G,\trianglelefteq}$ it follows that

$$\underbrace{\text{wt}_{G,\trianglelefteq}(\llbracket \rho^{[1]} \rrbracket)}_{=a_1} \cdot \underbrace{\text{wt}_{G,\trianglelefteq}(\llbracket \rho^1 \rrbracket \llbracket \rho^1 \rrbracket)}_{=1} \cdot \underbrace{\text{wt}_{G,\trianglelefteq}(\llbracket \rho^{[1]} \rrbracket)}_{=a_2} \cdot \dots \cdot \underbrace{\text{wt}_{G,\trianglelefteq}(\llbracket \rho^{[m]} \rrbracket)}_{=a_{2m-1}} \cdot \underbrace{\text{wt}_{G,\trianglelefteq}(\llbracket \rho^{[m]} \rrbracket)}_{=a_{2m}}$$

is $\mu(\rho)$ and thus exactly the weight assigned to those symbols by h . ■

The parser

Definition 5.21. Let G be an \mathcal{A} -weighted MCFG over Δ , $n \in \mathbb{N}$, and \trianglelefteq be a partial order on \mathcal{A} . The n -best CS parser with respect to G and \trianglelefteq , denoted by $\text{CS-parse}(G, n, \trianglelefteq)$, is a function from Δ^* to D_G^* that assigns for every $w \in \Delta^*$ the value

$$\begin{aligned} & \text{take}(n) \circ \text{map}(\text{toDeriv} \circ \text{hom}_{G_{\mathbb{B}}}) \circ \text{filter}(mD_c(G_{\mathbb{B}})) \\ & \quad \circ \text{sort}(\text{wt}_{G,\trianglelefteq}, \trianglelefteq)(R'(G_{\mathbb{B}}) \cap R_{h,w}) \end{aligned}$$

where $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$. □

Theorem 5.22. $\text{CS-parse}(G, n, \trianglelefteq)(w) \in n\text{-best}(\hat{\mu}, \trianglelefteq)(D_G(w))$ for every \mathcal{A} -weighted MCFG $G = (N, \Delta, S, P, \mu)$, $n \in \mathbb{N}$, partial order \trianglelefteq that respects \mathcal{A} , and word $w \in \Delta^*$.

Proof. Let $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$. We prove the claim by verifying the four conditions from Definition 5.2.

(i) This follows from the definition of sort and the bijection between $R(G_{\mathbb{B}}) \cap mD_c(G_{\mathbb{B}})$ and D_G (Corollary 3.13).

(ii) d_1, \dots, d_k are pairwise different since u_1, \dots, u_k are pairwise different and $R(G_{\mathbb{B}}) \cap mD_c(G_{\mathbb{B}})$ and D_G stand in bijection (Corollary 3.13).

(iii) Let $(d_1, \dots, d_k) = \text{CS-parse}(G, n, \trianglelefteq)(w)$ and (u_1, \dots, u_k) be a prefix of $\text{filter}(mD_c(G_{\mathbb{B}})) \circ \text{sort}(\text{wt}_{G,\trianglelefteq}, \trianglelefteq)(R(G_{\mathbb{B}}) \cap R_{h,w})$. Due to the definition of filter , there must be a natural number m such that (v_1, \dots, v_m) is a prefix of $\text{sort}(\text{wt}_{G,\trianglelefteq}, \trianglelefteq)(R(G_{\mathbb{B}}) \cap R_{h,w})$ for some $v_1, \dots, v_m \in (\Sigma \cup \bar{\Sigma})^*$ and u_1, \dots, u_k occur in that order in v_1, \dots, v_m . By the definition of sort , we have that $\text{wt}_{G,\trianglelefteq}(v_1) \trianglelefteq \dots \trianglelefteq \text{wt}_{G,\trianglelefteq}(v_m)$. Since u_1, \dots, u_k occur in that order in v_1, \dots, v_m , it follows that $\text{wt}_{G,\trianglelefteq}(u_1) \trianglelefteq \dots \trianglelefteq \text{wt}_{G,\trianglelefteq}(u_k)$. By Lemma 5.20 we obtain $\hat{\mu} \circ \text{toDeriv} \circ \text{hom}_{G_{\mathbb{B}}}(u_1) \trianglelefteq \dots \trianglelefteq \hat{\mu} \circ \text{toDeriv} \circ \text{hom}_{G_{\mathbb{B}}}(u_k)$ and by the definition of map we get $\hat{\mu}(d_1) \trianglelefteq \dots \trianglelefteq \hat{\mu}(d_k)$.

(iv) There is no $d \in D_G(w)$ with $\hat{\mu}(d) \triangleleft \hat{\mu}(d_k)$ since, by definition of sort , there is no $u \in R(G_{\mathbb{B}}) \cap mD_c(G_{\mathbb{B}})$ with $\text{wt}_{G,\trianglelefteq}(u) \triangleleft \text{wt}_{G,\trianglelefteq}(u_k)$. ■

In addition to just implementing $\text{CS-parse}(G, n, \preceq)$, we add a threshold θ for the weight of the found derivations. Then our algorithm will only consider the candidates whose weight is less than or equal to θ . Without such a threshold, the parsing algorithm would not terminate if the intersection of $R'(G_{\mathbb{B}})$ and $R_{h,w}$ had an infinite language and w had less than n derivations in G . On the other hand, if w has a derivation in G with a weight above the threshold, our algorithm will not find it, however large we choose n . Thus the algorithm only approximates $\text{CS-parse}(G, n, \preceq)$.

Input: a restricted \mathcal{A} -weighted MCFG $G = (N, \Delta, S, P, \mu)$, a number $n \in \mathbb{N}$, a partial order \preceq on \mathcal{A} where \preceq respects \mathcal{A} and \mathcal{A} is \preceq -factorisable, a threshold $\theta \in \mathcal{A}$ with $0 \preceq \theta$, and a word $w \in \Delta^*$
Output: the subsequence of derivations d in $\text{CS-parse}(G, n, \preceq)(w)$ with $\mu(d) \preceq \theta$

Algorithm 3:
Approximation
of CS-parse using
a threshold θ

```

1  function CS-Parse( $G, n, \preceq, \theta, w$ )
2    let  $mD_c(\Sigma, \mathfrak{P}) = mD_c(G_{\mathbb{B}})$ 
3    let  $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$ 
4    let  $\text{parses} = \varepsilon$ 
5    while  $\text{hasNextCandidate} \wedge |\text{parses}| < n$  do
6      let  $u = \text{nextCandidate}$ 
7      if  $\text{isMember}(\Sigma, \mathfrak{P}, u)$  then
8        append  $\text{toDeriv}(u)$  to the end of  $\text{parses}$ 
9      end if
10   end while
11   return  $\text{parses}$ 
12 end function

13 procedure  $\text{hasNextCandidate}$ 
14   return whether some  $u \in R'(G_{\mathbb{B}}) \cap R_{h,w}$  with  $\text{wt}_{G, \preceq}(u) \preceq \theta$  was
      not yet considered
15 end procedure

16 procedure  $\text{nextCandidate}$ 
17   from the elements of  $R'(G_{\mathbb{B}}) \cap R_{h,w}$  previously not considered,
      return an element whose image under  $\text{wt}_{G, \preceq}$  is smallest with
      respect to  $\preceq$ 
18 end procedure

```

To achieve that our algorithm terminates, we require for our input that G is restricted and that \mathcal{A} is \trianglelefteq -factorisable. Let us take a closer look at how Algorithm 3 works. Since $\text{sort}(\text{wt}_{G,\trianglelefteq}, \trianglelefteq)$ returns an infinite list, we have a procedure *nextCandidate* that computes only the next element in that list, *hasNextCandidate* returns whether there is such a next element. We utilise those two procedures to only compute the prefix of $\text{sort}(\text{wt}_{G,\trianglelefteq}, \trianglelefteq)(R'(G_{\mathbb{B}}) \cap R_{h,w})$ we need to obtain the first n parses. Since there are deterministic FSA to recognise both $R'(G_{\mathbb{B}})$ and $R_{h,w}$, the computation of $\text{sort}(\text{wt}_{G,\trianglelefteq}, \trianglelefteq)(R'(G_{\mathbb{B}}) \cap R_{h,w})$ amounts to enumerating the paths in a weighted labelled graph ordered by their weights⁸ and then replacing each path by the unique word recognised along it, where we take the graph with its labels from a deterministic FSA recognising $R'(G_{\mathbb{B}}) \cap R_{h,w}$ and the weights from $\text{wt}_{G,\trianglelefteq}$.

Proof of termination for Algorithm 3. Let $h = \text{weights}_G \circ \text{hom}_{G_{\mathbb{B}}}$ and M be the set of all weights assigned by $\text{wt}_{G,\trianglelefteq}$ to each of the words read in the transitions along one pass of every loop in the product of $\mathcal{M}'(G_{\mathbb{B}})$ and $\mathcal{M}_{h,w}$. Then M contains neither 0 nor 1 because $\mathcal{M}'(G_{\mathbb{B}})$ has no harmful loops. Since \cdot has arbitrarily large powers, we know that for every element $a \in M$, there is a natural number k such that $\theta \trianglelefteq a^k$. Let \hat{k} be the maximum of all such k 's. There are only finitely many runs in the product of $\mathcal{M}'(G_{\mathbb{B}})$ and $\mathcal{M}_{h,w}$ that contain every loop less than \hat{k} times. Hence *hasNextCandidate* is false after a finite number of iterations of the **while**-loop (lines 6 to 11), and Algorithm 3 terminates. ■

Example 5.23 (Examples 2.5, 3.7, and 5.7 continued). Consider the word $w = ac$. The product of $\mathcal{M}'(G_{\mathbb{B}})$ and $\mathcal{M}_{h,w}$ together with $\text{wt}_{G,\trianglelefteq}$ is shown in Figure 8 (for the product construction see Hopcroft and Ullman 1979, after Theorem 3.3). It suffices to consider at most 8 candidates to find the (only) derivation of w in G , as is shown in Table 4. The candidates themselves are not shown, instead we see their weight, their corresponding path in the graphical representation of the product of $\mathcal{M}'(G_{\mathbb{B}})$ and $\mathcal{M}_{h,w}$, and whether they are in $mD_c(G_{\mathbb{B}})$. Candidate 7 is exactly $\text{toBrackets}(\rho'_1(\rho'_2(\rho'_4), \rho'_5))$. □

⁸This is described for example in Hoffman and Pavley 1959 for graphs with edge weights from the real numbers. However, their algorithm works also for complete commutative strong bimonoids \mathcal{A} with a partial order on \mathcal{A} that respects \mathcal{A} .

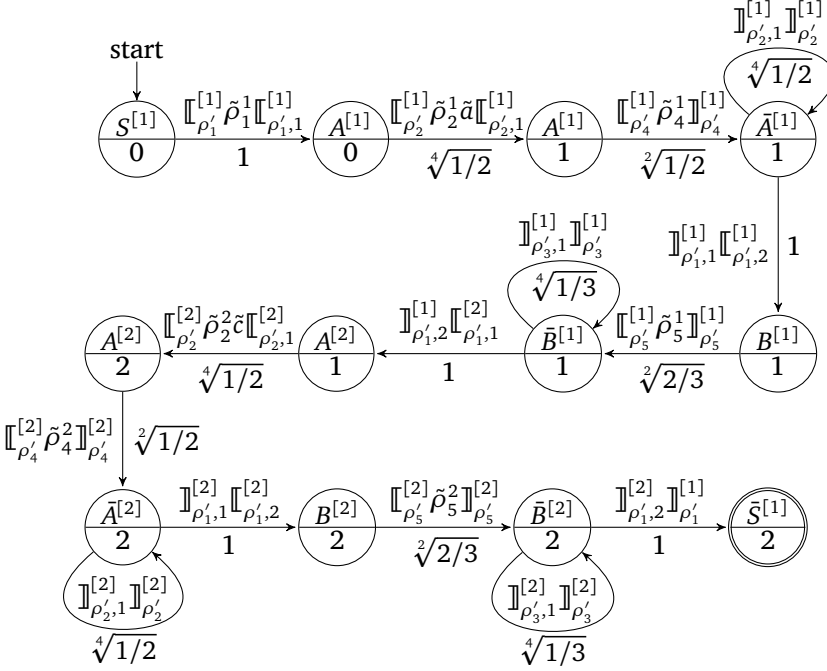


Figure 8: Product of $\mathcal{M}'(G_{\mathbb{B}})$ and $\mathcal{M}_{h,w}$ with the weight assigned by $\text{wt}_{G_{\leq}}$ to words read in each transition

Table 4: First eight paths (sorted by their image under $\text{wt}_{G_{\leq}}$) in the product of $\mathcal{M}'(G_{\mathbb{B}})$ and $\mathcal{M}_{h,w}$ and whether the corresponding candidate u_i is in $mD_c(G_{\mathbb{B}})$

i	$\text{wt}_{G_{\leq}}(u_i)$	path corresponding to u_i	$u_i \in mD_c(G_{\mathbb{B}})$?
1	$\frac{1}{3\sqrt{2}}$	without using any loops	no
2	$\frac{1}{3\sqrt[4]{8}}$	use the loop of $(\bar{A}^{[1]}, 1)$	no
3	$\frac{1}{3\sqrt[4]{8}}$	use the loop of $(\bar{A}^{[2]}, 2)$	no
4	$\frac{1}{3\sqrt[4]{12}}$	use the loop of $(\bar{B}^{[1]}, 1)$	no
5	$\frac{1}{3\sqrt[4]{12}}$	use the loop of $(\bar{B}^{[2]}, 2)$	no
6	$\frac{1}{6}$	use the loop of $(\bar{A}^{[1]}, 1)$ twice	no
7	$\frac{1}{6}$	use the loops of $(\bar{A}^{[1]}, 1)$ and $(\bar{A}^{[2]}, 2)$	yes
8	$\frac{1}{6}$	use the loop of $(\bar{A}^{[2]}, 2)$ twice	no

From the proof of termination of Algorithm 3 we can gather that the complexity of the algorithm depends on how many candidates are considered. The upper bound for the number of considered candidates is determined by n , θ , and the number \widehat{k} . In particular, it does not depend on the input word w . Therefore we cannot expect to get a meaningful time complexity for Algorithm 3.

Instead we will only determine the time complexity of the evaluation of *isMember* on line 7.

Lemma 5.24. Let G be an MCFG and $mD_c(\Sigma, \mathfrak{P}) = mD_c(G)$. The every element of $R'(G) \cap mD_c$ is \mathfrak{P} -simple.

Proof. Let $G = (N, \Gamma, S, P)$ and Σ and \mathfrak{P} be defined as in Definition 4.9.

We prove the following statement by induction on the length of $w_1 \cdots w_\ell$: If $B \in N$ and $w_1, \dots, w_\ell \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_c$ and w_κ is recognised along a path from $B^{[\kappa]}$ to $\overline{B}^{[\kappa]}$ in $\mathcal{M}'(G)$ for every $\kappa \in [\ell]$, then $w_1 \cdots w_\ell$ is \mathfrak{P} -simple.

By setting $\ell = 1$ and $B = S$, this statement implies our claim. Now let $B \in N$ and $w_1, \dots, w_\ell \in D(\Sigma)$ such that $w_1 \cdots w_\ell \in mD_c$ and w_κ is recognised along a path from $B^{[\kappa]}$ to $\overline{B}^{[\kappa]}$ in $\mathcal{M}'(G)$ for every $\kappa \in [\ell]$. By the definitions of $\mathcal{M}'(G)$ and $mD_c(G)$, we know that there is some production $\rho = B \rightarrow f(B_1, \dots, B_k) \in P$ with

$$f = [u_{1,0}x_{i(1,1)}^{j(1,1)}u_{1,1} \cdots x_{i(1,p_1)}^{j(1,p_1)}u_{1,p_1}, \dots, u_{\ell,0}x_{i(\ell,1)}^{j(\ell,1)}u_{\ell,1} \cdots x_{i(\ell,p_\ell)}^{j(\ell,p_\ell)}u_{\ell,p_\ell}]$$

such that for every $\kappa \in [\ell]$ either

- (i) $w_\kappa = \llbracket \rho^{[\kappa]} \tilde{u}_{\kappa,0} \rrbracket \rho^{[\kappa]}$ or
- (ii) $w_\kappa = \llbracket \rho^{[\kappa]} \tilde{u}_{\kappa,0} \quad \llbracket \rho^{[j(\kappa,1)]} \quad v_{i(\kappa,1)}^{j(\kappa,1)} \quad \llbracket \rho^{[j(\kappa,1)]} \quad \tilde{u}_{\kappa,1} \quad \dots$
 $\quad \quad \quad \llbracket \rho^{[j(\kappa,p_\kappa)]} \quad v_{i(\kappa,p_\kappa)}^{j(\kappa,p_\kappa)} \quad \llbracket \rho^{[j(\kappa,p_\kappa)]} \quad \tilde{u}_{\kappa,p_\kappa} \quad \llbracket \rho^{[\kappa]} \rrbracket,$

and $v_i^j \in D(\Sigma)$ is recognised along a path from $B_i^{[j]}$ to $\overline{B}_i^{[j]}$ in $\mathcal{M}'(G)$ for every $i \in [k]$ and $j \in [\text{sort}(B_i)]$, and $v_i^1 \cdots v_i^{\text{sort}(B_i)} \in mD_c(G)$ for every $i \in [k]$. Set $w'_1, \dots, w'_\ell \in D(\Sigma)$ such that $w_\kappa = \llbracket \rho^{[\kappa]} w'_\kappa \rrbracket \rho^{[\kappa]}$ for each $\kappa \in [\ell]$. Then by induction hypothesis the string $v_i^1 \cdots v_i^{\text{sort}(B_i)}$ is \mathfrak{P} -simple for every $i \in [k]$. By analysing the form of w_κ for each $\kappa \in [\ell]$, we observe that $\text{occ}_p w'_1 \cdots w'_\ell \leq 1$ and $\text{occ}_p w_1 \cdots w_\ell \leq 1$ for each cell $p \in \{p' \in \mathfrak{P} \mid |p'| \geq 2\}$. Hence, $w_1 \cdots w_\ell$ is \mathfrak{P} -simple. ■

Let G be an MCFG and $mD_c(\Sigma, \mathfrak{P}) = mD_c(G)$. Since every element of $R'(G) \cap mD_c(G)$ is \mathfrak{P} -simple, we can use *isMember'* on line 7 instead of *isMember*. Then line 7 can be done in time quadratic in the length of the candidate u .

6 RELATED PARSING APPROACHES

An established approach to speed up the parsing of MCFGs for practical applications is to use a formalism with lower parsing complexity than MCFGs to guide the exploration of the search space. In the following, we will focus on four such approaches.

The parsers in Barthélemy *et al.* (2001); Burden and Ljunglöf (2005); van Cranenburgh (2012) work as follows: Suppose that we want to parse a given word w with a grammar G of a formalism A . We first construct a grammar (or automaton) G' in a formalism B that has a lower parsing complexity than A . This can be done offline. Then, we parse w with G' . Lastly, we parse w with G , but while doing so, we consult the parses of w in G' to *guide* the exploration of the search space (of possible parses). The three papers differ in their choice of formalisms for G and G' , and in their use of the parses of w in G' while parsing w in G :

- (i) Barthélemy *et al.* (2001) have a positive range concatenation grammar (short: PRCG) (Boullier 1998) of arbitrary arity for G and use a PRCG of arity 1 for G' . They extract from the parse forest F of w in G' a so-called *guiding structure* and query this structure while parsing w in G . The guiding structure can range from a set of instantiated clauses that occur in F to F itself. In their experiments they used as a guiding structure the function that assigns for each instantiated clause the number of its occurrences in F .
- (ii) Burden and Ljunglöf (2005, Section 4) have a linear context-free rewriting system (short: LCFRS) (Vijay-Shanker *et al.* 1987) for G and a context-free grammar for G' . They use deductive parsing. The *parse chart* C' of w in G' is created. While creating the parse chart of w in G , only items are created that are consistent with the items in C' . The algorithm is therefore an instance of *coarse-to-fine parsing* (Charniak *et al.* 2006).
- (iii) Van Cranenburgh (2012) has a probabilistic LCFRS (of arbitrary fan-out) for G and a probabilistic LCFRS of fan-out 1 for G' . As

Burden and Ljunglöf (2005), he uses deductive parsing: First, a parse chart C' of w in G' is created. Then the probabilities of G' are used to restrict C' to the n best parses, obtaining a new parse chart \widehat{C} , this step is called *pruning*. A value of $n = 50$ was used in the experiments. Then, while creating the parse chart of w in G , only items are created that are consistent with the items in \widehat{C} . The algorithm is an instance of coarse-to-fine parsing.

Kallmeyer and Maier (2015) present a different approach:

- (iv) They construct an FSA G' as the predict/resume-closure of *thread automaton thread stores* (Villemonde de la Clergerie 2002) where the corresponding thread automaton is constructed from the given LCFRS G . The addresses in the thread stores are represented by regular expressions to keep the set of states of G' finite. Then, a *parse table* is read off of G' . As opposed to Items (i) to (iii), w is *not* parsed with G' . Instead, while parsing w with G using a shift-reduce parser, the parse table is consulted directly at each shift or reduce operation to determine the successor state. Their algorithm is an instance of LR-parsing.

With the Chomsky-Schützenberger parsing presented in this article, we construct from the given weighted LCFRS G *three* devices (instead of just one): the deterministic FSA $\mathcal{M}'(G_{\mathbb{B}})$ together with the weight assignment $\text{wt}_{G, \leq}$, the congruence multiple Dyck language $mD_c(G_{\mathbb{B}})$, and the alphabetic homomorphism $\text{hom}_{G_{\mathbb{B}}}$. For the given word w we construct a deterministic FSA, let us call it \mathcal{M} , that recognises $\text{hom}_{G_{\mathbb{B}}}^{-1}(w) \cap L(\mathcal{M}'_{G_{\mathbb{B}}})$. Constructing \mathcal{M} is an additional pre-processing step in comparison to Items (i) to (iv). In contrast to Item (iii), we do not use the weight assignment $\text{wt}_{G, \leq}$ for pruning. We instead use it to enumerate the elements of $L(\mathcal{M})$ in increasing order of their costs. Finally, we filter the list of those elements with $mD_c(G_{\mathbb{B}})$. Note that w is never actually parsed with G as in Items (i) to (iv).

We obtained a weighted version of the Chomsky-Schützenberger characterisation of MCFLs for complete commutative strong bimonoids (Theorem 3.12) by separating the weights from the weighted MCFG and using Yoshinaka *et al.* (2010, Theorem 3) for the unweighted part.

We defined a variant of multiple Dyck languages that uses congruence relations (Definition 4.2), gave an algorithm to decide whether a word is in a given congruence multiple Dyck language in Algorithm 2, and derived a CS representation using congruence multiple Dyck languages.

Following the idea of Hulden (2011), we used this CS representation for weighted MCFL to describe a parsing algorithm (Algorithm 3) that approximates n -best parsing for restricted weighted MCFGs with a weight structure that is partially ordered and factorisable.

The utility of Algorithm 3 can *not* be judged based on theoretical considerations alone. The author therefore plans to implement it and evaluate it empirically.

ACKNOWLEDGEMENTS

The author is very grateful to the anonymous reviewers, as well as Mark-Jan Nederhof and Toni Dietze for their insightful comments, which helped improve this article.

REFERENCES

- Alfred Vaino AHO and Jeffrey D. ULLMAN (1972), *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN 0-13-914556-7, <http://dl.acm.org/citation.cfm?id=SERIES11430.578789>.
- Krasimir ANGELOV (2009), Incremental parsing with parallel multiple context-free grammars, in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 69–76, Association for Computational Linguistics, <http://dl.acm.org/citation.cfm?id=1609074>.
- François BARTHÉLEMY, Pierre BOULLIER, Philippe DESCHAMP, and Éric DE LA CLERGERIE (2001), Guided parsing of range concatenation languages, in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics - ACL '01*, Association for Computational Linguistics (ACL), doi:10.3115/1073012.1073019.
- Pierre BOULLIER (1998), A generalization of mildly context-sensitive formalisms, in *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 4)*, pp. 17–20, Citeseer.
- Sabine BRANTS, Stefanie DIPPER, Peter EISENBERG, Silvia HANSEN-SCHIRRA, Esther KÖNIG, Wolfgang LEZIUS, Christian ROHRER, George SMITH, and Hans

- USZKOREIT (2004), TIGER: Linguistic Interpretation of a German Corpus, *Research on Language and Computation*, 2(4):597–620, doi:10.1007/s11168-004-7431-3.
- Håkan BURDEN and Peter LJUNGLÖF (2005), Parsing Linear Context-free Rewriting Systems, in Harry BUNT, Robert MALOUF, and Alon LAVIE, editors, *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pp. 11–17, Association for Computational Linguistics, Stroudsburg, PA, USA, <http://dl.acm.org/citation.cfm?id=1654494.1654496>.
- Matthias BÜCHSE, Daniel GEISLER, Torsten STÜBER, and Heiko VOGLER (2010), n-Best Parsing Revisited, in Frank DREWES and Marco KUHLMANN, editors, *Proceedings of the 2010 Workshop on Applications of Tree Automata in Natural Language Processing*, pp. 46–54, Association for Computational Linguistics, <http://www.aclweb.org/anthology/W10-2506>.
- Eugene CHARNIAK, Michael POZAR, Theresa VU, Mark JOHNSON, Micha ELSNER, Joseph AUSTERWEIL, David ELLIS, Isaac HAXTON, Catherine HILL, R. SHRIVATHS, and Jeremy MOORE (2006), Multilevel coarse-to-fine PCFG parsing, in *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics -*, Association for Computational Linguistics (ACL), doi:10.3115/1220835.1220857.
- Noam CHOMSKY and Marcel Paul SCHÜTZENBERGER (1963), The algebraic theory of context-free languages, pp. 118–161, doi:10.1016/S0049-237X(09)70104-1.
- Tobias DENKINGER (2015), A Chomsky-Schützenberger representation for weighted multiple context-free languages, in *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing (FSM/NLP 2015)*, <http://aclweb.org/anthology/W15-4803>.
- Manfred DROSTE, Torsten STÜBER, and Heiko VOGLER (2010), Weighted finite automata over strong bimonoids, *Information Sciences*, 180(1):156–166, doi:10.1016/j.ins.2009.09.003.
- Manfred DROSTE and Heiko VOGLER (2013), The Chomsky-Schützenberger Theorem for Quantitative Context-Free Languages, in Marie-Pierre BÉAL and Olivier CARTON, editors, *Developments in Language Theory*, volume 7907 of *Lecture Notes in Computer Science*, pp. 203–214, Springer Berlin Heidelberg, ISBN 978-3-642-38770-8, doi:10.1007/978-3-642-38771-5_19.
- Jürgen DUSKE, Rainer PARCHMANN, and Johann SPECHT (1979), A Homomorphic Characterization of Indexed Languages, 15(4):187–195.
- Séverine FRATANI and El Makki VOUNDY (2015), Context-free characterization of Indexed Languages, abs/1409.6112, <http://arxiv.org/abs/1409.6112>.
- Séverine FRATANI and El Makki VOUNDY (2016), Homomorphic Characterizations of Indexed Languages, in Adrian-Horia DEDIU, Jan

JANOŮŠEK, Carlos MARTÍN-VIDE, and Bianca TRUTHE, editors, *Proceedings of the 10th International Conference on Language and Automata Theory and Applications (LATA 2015)*, pp. 359–370, Springer Science + Business Media, doi:10.1007/978-3-319-30000-9_28.

Luisa HERRMANN and Heiko VOGLER (2015), A Chomsky-Schützenberger Theorem for Weighted Automata with Storage, in Andreas MALETTI, editor, *Proceedings of the 6th International Conference on Algebraic Informatics (CAI 2015)*, volume 9270, pp. 90–102, Springer International Publishing, ISBN 978-3-319-23021-4, doi:10.1007/978-3-319-23021-4_11.

Walter HOFFMAN and Richard PAVLEY (1959), A Method for the Solution of the Nth Best Path Problem, 6(4):506–514, doi:10.1145/320998.321004.

John Edward HOPCROFT and Jeffrey David ULLMAN (1969), *Formal languages and their relation to automata*, Addison-Wesley Longman Publishing Co., Inc.

John Edward HOPCROFT and Jeffrey David ULLMAN (1979), *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1st edition.

Liang HUANG and David CHIANG (2005), Better k-best Parsing, in *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pp. 53–64, Association for Computational Linguistics, Stroudsburg, PA, USA, <http://dl.acm.org/citation.cfm?id=1654494.1654500>.

Mans HULDEN (2011), Parsing CFGs and PCFGs with a Chomsky-Schützenberger Representation, in Zygmunt VETULANI, editor, *Human Language Technology. Challenges for Computer Science and Linguistics*, volume 6562 of *Lecture Notes in Computer Science*, pp. 151–160, Springer Berlin Heidelberg, ISBN 978-3-642-20094-6, doi:10.1007/978-3-642-20095-3_14.

Laura KALLMEYER and Wolfgang MAIER (2013), Data-driven parsing using probabilistic linear context-free rewriting systems, *Computational Linguistics*, 39(1):87–119, doi:10.1162/COLI_a_00136.

Laura KALLMEYER and Wolfgang MAIER (2015), LR Parsing for LCFRS, in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics (ACL), doi:10.3115/v1/n15-1134.

Makoto KANAZAWA (2014), Multidimensional trees and a Chomsky-Schützenberger-Weir representation theorem for simple context-free tree grammars, ISSN 1465-363X, doi:10.1093/logcom/exu043.

Marco KUHLMANN and Giorgio SATTA (2009), Treebank Grammar Techniques for Non-projective Dependency Parsing, in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pp. 478–486, Association for Computational Linguistics, Stroudsburg, PA, USA, <http://dl.acm.org/citation.cfm?id=1609067.1609120>.

Wolfgang MAIER and Anders SØGAARD (2008), Treebanks and mild context-sensitivity, in *Proceedings of Formal Grammar*, p. 61, <http://web.stanford.edu/group/cslipublications/cslipublications/FG/2008/maier.pdf>.

Jens MICHAELIS (2001a), Derivational Minimalism Is Mildly Context-Sensitive, in Michael MOORTGAT, editor, *Logical Aspects of Computational Linguistics*, volume 2014 of *Lecture Notes in Computer Science*, pp. 179–198, Springer Berlin Heidelberg, ISBN 978-3-540-42251-8, doi:10.1007/3-540-45738-0_11.

Jens MICHAELIS (2001b), Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars, in Philippe GROOTE, Glyn MORRILL, and Christian RETORÉ, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Computer Science*, pp. 228–244, Springer Berlin Heidelberg, ISBN 978-3-540-42273-0, doi:10.1007/3-540-48199-0_14.

Mehryar MOHRI (2000), Minimization algorithms for sequential transducers, 234(1–2):177–201, ISSN 0304-3975, doi:10.1016/s0304-3975(98)00115-7.

Arto SALOMAA (1973), *Formal languages*, Academic Press, ISBN 978-0126157505.

Arto SALOMAA and Matti SOITTOLA (1978), *Automata-Theoretic Aspects of Formal Power Series*, Springer New York, ISBN 978-1-4612-6266-4, doi:10.1007/978-1-4612-6264-0.

Hiroyuki SEKI, Takashi MATSUMURA, Mamoru FUJII, and Tadao KASAMI (1991), On multiple context-free grammars, 88(2):191–229, ISSN 0304-3975, doi:10.1016/0304-3975(91)90374-B.

Hiroyuki SEKI, Ryuichi NAKANISHI, Yuichi KAJI, Sachiko ANDO, and Tadao KASAMI (1993), Parallel Multiple Context-free Grammars, Finite-state Translation Systems, and Polynomial-time Recognizable Subclasses of Lexical-functional Grammars, in *Proceedings of the 31st Annual Meeting on Association for Computational Linguistics*, ACL '93, pp. 130–139, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/981574.981592.

Andreas VAN CRANENBURGH (2012), Efficient Parsing with Linear Context-free Rewriting Systems, in Walter DAELEMANS, editor, *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pp. 460–470, Association for Computational Linguistics, Stroudsburg, PA, USA, ISBN 978-1-937284-19-0, <http://dl.acm.org/citation.cfm?id=2380816.2380873>.

Krishnamurti VIJAY-SHANKER (1988), *A study of tree adjoining grammars*, Ph.D. thesis, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.401.1695&rep=rep1&type=pdf>.

Krishnamurti VIJAY-SHANKER, David Jeremy WEIR, and Aravind K. JOSHI (1986), Tree adjoining and head wrapping, in *Proceedings of the 11th Conference*

CS-parsing for weighted MCFL

on *Computational Linguistics*, pp. 202–207, Association for Computational Linguistics, doi:10.3115/991365.991425.

Krishnamurti VIJAY-SHANKER, David Jeremy WEIR, and Aravind K. JOSHI (1987), Characterizing Structural Descriptions Produced by Various Grammatical Formalisms, in *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, ACL '87, pp. 104–111, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/981175.981190.

Éric VILLEMONTÉ DE LA CLERGERIE (2002), Parsing Mildly Context-Sensitive Languages with Thread Automata, in *Proceedings of the 19th International Conference on Computational Linguistics (COLING '02)*, volume 1, pp. 1–7, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/1072228.1072256.

David Jeremy WEIR (1988), *Characterizing mildly context-sensitive grammar formalisms*, Ph.D. thesis, <http://repository.upenn.edu/dissertations/AAI8908403>.

David Jeremy WEIR and Aravind K. JOSHI (1988), Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems, in *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pp. 278–285, Association for Computational Linguistics, doi:10.3115/982023.982057.

Ryo YOSHINAKA, Yuichi KAJI, and Hiroyuki SEKI (2010), Chomsky-Schützenberger-type characterization of multiple context-free languages, in Adrian-Horia DEDIU, Henning FERNAU, and Carlos MARTÍN-VIDE, editors, *Language and Automata Theory and Applications*, pp. 596–607, Springer, ISBN 978-3-642-13088-5, doi:10.1007/978-3-642-13089-2_50.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.

<http://creativecommons.org/licenses/by/3.0/>



Relative clauses as a benchmark for Minimalist parsing

Thomas Graf, James Monette, and Chong Zhang
Department of Linguistics, Stony Brook University, USA

ABSTRACT

Minimalist grammars have been used recently in a series of papers to explain well-known contrasts in human sentence processing in terms of subtle structural differences. These proposals combine a top-down parser with complexity metrics that relate parsing difficulty to memory usage. So far, though, there has been no large-scale exploration of the space of viable metrics. Building on this earlier work, we compare the ability of 1,600 metrics to derive several processing effects observed with relative clauses, many of which have been proven difficult to unify. We show that among those 1,600 candidates, a few metrics (and only a few) can provide a unified account of all these contrasts. This is a welcome result for two reasons: First, it provides a novel account of extensively studied psycholinguistic data. Second, it significantly limits the number of viable metrics that may be applied to other phenomena, thus reducing theoretical indeterminacy.

Keywords:
Parsing, sentence processing, Minimalist grammars, memory usage, relative clauses, promotion analysis, English, East Asian

1

INTRODUCTION

It is beyond doubt that the structural properties of a sentence influence how easily said sentence is processed by humans. For example, a sentence with multiple levels of center embedding is harder to parse than its counterpart with right embedding, and English subject relative clauses are processed more quickly than object relative clauses. There is large disagreement, however, on what the relevant structural properties are. This paper continues a recent series of investigations (Kobele *et al.* 2013; Graf and Marcinek 2014; Graf *et al.* 2015; Gerth

2015) that approach this question by combining Stabler's (2013) top-down parser for Minimalist grammars (MGs) with structurally rich analyses from Minimalist syntax, the most recent version of Chomsky's transformational grammar framework.

The works above are part of longer tradition applying computational formalisms to human sentence processing (Joshi 1990; Rambow and Joshi 1995; Steedman 2001; Hale 2011; Yun *et al.* 2014, among others). Common to all of them is a tripartite structure consisting of I) an articulated theory of syntax that has sufficient empirical coverage to be applicable to a wide range of constructions, II) a sound and complete parser for the syntactic formalism, and III) a complexity metric that acts as linking theory to derive psycholinguistic predictions from the previous two components. The allure of this approach is that all components are rigorously specified and mathematically worked out to a degree that allows for very fine-grained and detailed processing predictions. Not only does this provide insightful explanation of certain processing phenomena, it also makes it possible to distinguish between competing syntactic proposals based on their psycholinguistic predictions.

The decomposition into three distinct modules is intuitive and elegant, but it also highlights a worrying underspecification issue. With three components that by necessity have to make very detailed assumptions, it is to be expected that a large number of different combinations all replicate the same behavioral data. For instance, a syntactician whose analysis makes the wrong processing predictions may insist that the problem is not with the analysis but with the parser or the complexity metric. For the previous MG modeling work, the issue has already arisen with respect to the choice of complexity metrics, with Graf and Marcinek (2014) and Graf *et al.* (2015) arguing for slightly different metrics than Kobele *et al.* (2013) and Gerth (2015). The specificity of computational models – one of their biggest virtues – thus runs the risk of combinatorial explosion and empirical indeterminacy, which would severely weaken their appeal.

In order to address this issue, we define a total of 1,600 complexity metrics and evaluate whether they can account for the processing contrasts with relative clauses that were originally discussed in Kobele *et al.* (2013), Graf and Marcinek (2014), and Graf *et al.* (2016). We also use two different analyses of relative clauses from Minimal-

ist syntax (promotion and *wh*) to determine whether the set of empirically viable metrics is still sufficiently structure-sensitive to distinguish between the accounts. Our findings show that the issues of indeterminacy and combinatorial explosion are much less severe in practice than one might expect – a handful of data points is sufficient to significantly reduce the space of empirically viable parsing models. Furthermore, this reduced set contains some very simple metrics that are capable of explaining a wide range of processing contrasts in an intuitively pleasing fashion. At the same time, the set of viable metrics varies with the posited analysis in an interesting way, which suggests that more data points will eventually allow us to rule out specific syntactic proposals.

The paper proceeds as follows. The next section introduces MGs (2.1) and explains how the behavior of Stabler’s (2013) top-down parser for MGs can be represented at an abstract level with index/outdex annotated derivation trees. Section 3 then defines 1,600 complexity metrics that operate over these annotated derivation trees. This large number is obtained from just three basic metrics that are subsequently parameterized along various axes. In Section 4, we establish the empirical viability of only a few of these 1,600 metrics for the processing of relative clauses in English, Chinese, Korean, and Japanese. The paper concludes with a discussion of conceptual and methodological aspects of our finding.

2 MINIMALIST GRAMMARS FOR PROCESSING

The mathematical backbone of this paper is provided by MGs (Stabler 1997, 2011) and the top-down MG parser proposed by Stabler (2013). MGs were chosen because they present a rare combination of traits. On the one hand they incorporate ideas from Chomskyan syntax that have found wide adoption in syntactic processing. MGs are also flexible enough to implement even unusual proposals such as Late Merge (Lebeaux 1988; Takahashi and Hulseley 2009) and test their predictions. On the other hand they can be regarded as a simple variant of context-free grammars, which have been studied extensively in the computational parsing literature (Shieber *et al.* 1995; Sikkell 1997). MGs thus act as mathematical glue between formal parsing theory, psycholinguistics, and large areas of contemporary syntax.

After a brief introduction to MGs in Section 2.1, we discuss the central role of derivation trees (2.2) and how Kobele *et al.*'s (2013) system of annotating derivation trees acts as a high-level abstraction of Stabler's top-down parser (2.3). This provides us with a unified representational format that simultaneously describes the structure of a sentence and relevant parts of its parse history. When combined with the complexity metrics in Section 3, this simple system is sufficient to obtain concrete processing predictions.

2.1 *Non-technical introduction to Minimalist grammars*

MGs take inspiration from the most recent iteration of transformational grammar, known as *Minimalism* or *Minimalist syntax* (Chomsky 1995b, 2001). Like all iterations of transformational grammar, MGs furnish a mechanism for encoding basic head-argument relations which are then manipulated by a movement operation to produce the actual syntactic structure. In the case of MGs, these two modes of structure building are called *Merge* and *Move*, respectively. What differentiates MGs from standard Minimalism is their fully explicit feature calculus, which regulates when each operation has to be applied. This makes MGs a lexicalized formalism similar to CCG, LFG or HPSG in the sense that each grammar G is just a finite list of feature-annotated lexical items (LIs) – a structure is generated by G iff it can be assembled from LIs of G according to their feature specifications.

For the purposes of this paper, an intuitive understanding of MGs is sufficient, so we do not give a complete, rigorous definition here. The interested reader is referred to Stabler (2011), Graf (2013, Chapters 1 & 2), and Gerth (2015, Section 4.1) for detailed yet accessible introductions. Suppose that we want to generate the sentence *John, the girl likes*. While there are in principle infinitely many distinct ways to do so with MGs, only a few, marginally different ones are also entertained in Minimalist syntax. The most common analysis posits the syntactic structure shown in Figure 1, where dashed lines have been added to indicate which positions certain phrases were moved from. The basic idea is that the sentence starts out as *the girl likes John* and is subsequently transformed into *John, the girl likes* via *Move*. There are, however, several independently motivated factors that complicate this simple picture.

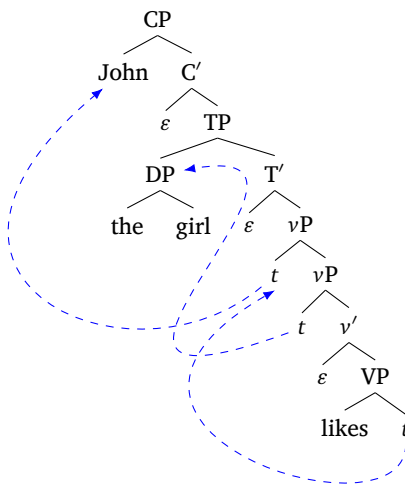


Figure 1:
MG phrase structure tree for
John, the girl likes; dashed arrows
indicate movement

- Instead of the usual X'-structure, a more succinct Bare Phrase Structure tree (Chomsky 1995a) is assumed. The two are almost identical except that Bare Phrase Structure trees omit all unary branches (wherefore they are strictly binary branching). So an X'-structure like $[_{DP} [_{D'} [_{D} \textit{John}]]]$ reduces to simply *John*.
- A phrase can have multiple specifiers but only one complement. Heads are always linearized to the left of their complement and to the right of their specifiers (Kayne 1994).
- Sentences are allowed to contain unpronounced LIs, which are denoted ϵ .
- VPs are split into VP and ν P (read “little VP”) following ideas first formulated in Larson (1988). The ν P phrase serves many purposes in the literature, but its relevance for this paper is limited to its role as the base position for subjects.
- Subjects in English (and all other languages discussed in this paper) move from the lowest specifier of ν P to the canonical subject position, i.e. the lowest specifier of TP.
- For the sake of exposition, Figure 1 also incorporates the assumption that a phrase moving out of ν P must intermittently land in a specifier of ν P (Chomsky 2001). We omit this in the remainder of the paper as it has no effect on our results and thus would only add irrelevant complexity.

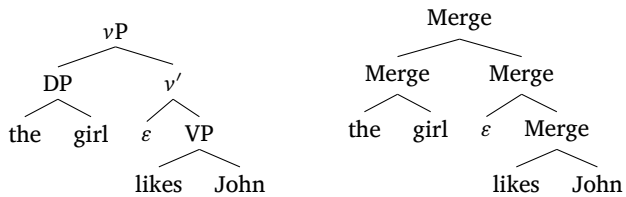
While many of these assumptions are not widely shared outside of Minimalist syntax and add a certain degree of verbosity, they rest on decades of syntactic research. Since a major goal of the MG parsing project is to determine to what extent different syntactic assumptions can affect processing predictions, we adapt these Minimalist analyses as faithfully as possible to MGs.

Let us then look more closely at how the phrase structure tree in Figure 1 is assembled by an MG. MGs combine LIs into trees via the structure-building operations *Merge* and *Move* based on the features carried by those LIs. We start out by applying Merge to *likes* and *John*, which marks *John* as an argument of *likes*. In order for this Merge step to be licensed, *likes* must have a feature that indicates that the verb takes a DP as its complement, whereas *John* must have the matching category feature D. The verb is then selected by the unpronounced head of *vP*, which also requires a DP as its specifier. In this particular case, the DP is obtained by merging *the* with *girl*. As before, all these instances of Merge must be licensed by suitable feature specifications on the LIs. We do not write out these features here as they will play no role in this paper beyond the fact that an MG parser needs to keep track of all features.

At this point we have assembled the tree depicted in Figure 2. This figure also shows the corresponding *derivation tree*, which records the structure-building steps taken to build the phrase structure tree. In a derivation tree, all leaves are lexical items, and all interior nodes are labeled by Merge and Move depending on which operation takes place at that point. The two trees in Figure 2 differ only in their interior node labels, but they will diverge more significantly once Move enters the picture.

So far each step has added new material to the tree via Merge. But now something different happens: the object DP *John* is displaced to a specifier of *vP* via Move. When exactly Move may take place and

Figure 2:
An intermediate state of
the assembly of *John, the
girl likes*; all feature
specifications are omitted



which phrase it may displace is once again controlled by the feature calculus. In the case at hand, the vP -head must have a feature f that requires some phrase to move into a vP specifier. Similarly, *John* must have a feature that requires it to undergo f -movement. The result of Move is shown in Figure 3. Note that the phrase structure tree and the derivation tree now have different geometries, with *John* in a vP specifier in the former but still in its base position in the latter. Consequently, reading the leaves in a derivation tree from left to right thus may not produce the actual word order of the sentence, which will play an important role during the discussion of MG parsing in the next two sections.

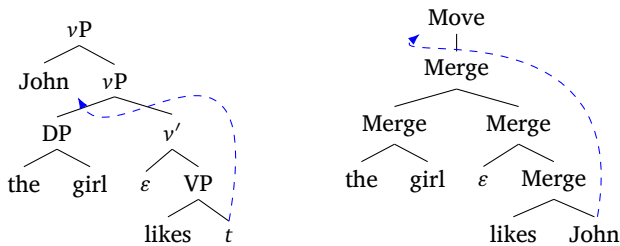
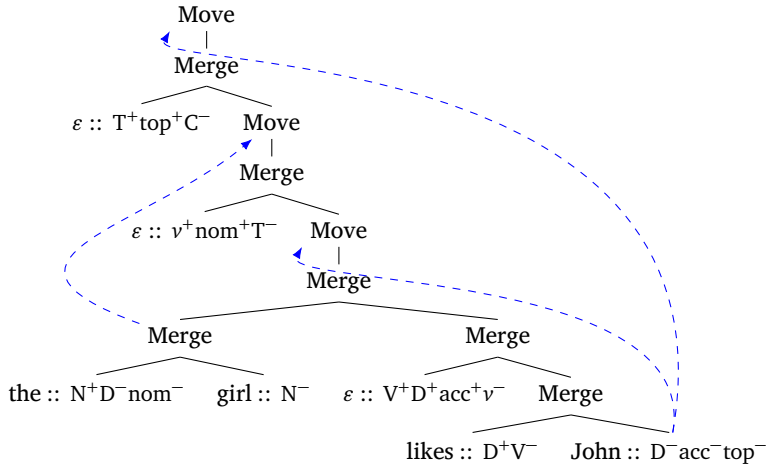


Figure 3:
Phrase structure tree and
derivation tree after the
first movement step;
dashed arrows are not part
of the trees

The reader may also wonder why Move is represented as a unary branching node even though the operation seems to involve two arguments, a target position and the subtree that is to be displaced. The answer is that Move is a deterministic operation in MGs. The target position is always added at the root of the current tree, and from the feature specifications of LIs one can always infer which particular subtree is to be displaced. There simply is no need to explicitly specify the arguments of Move. However, in the derivation trees in this paper we omit the feature specifications for the sake of brevity and instead use dashed arrows to indicate what moves where.

Strictly speaking we could stop here as the current phrase structure tree already has *John, the girl likes* as its string yield. However, the tree is still incomplete according to Minimalist syntax and thus the derivation continues. After merging the tree with an unpronounced T-head, the subject *the girl* moves from its base position inside vP to the canonical subject position in the specifier of TP. Then the TP is merged with an unpronounced C-head, and *John* is topicalized by moving it to a CP specifier. By assumption, a tree is well-formed iff its root is

Figure 4:
MG derivation
tree for *John, the
girl likes* with
explicit feature
specifications for
all LIs



labeled CP and all feature requirements have been satisfied. Since this is the case for this tree, the derivation can stop here. The full derivation is given in Figure 4 – to give the reader an idea of what the MG feature calculus looks like, we also list the feature specifications in this example.

2.2 The central role of derivation trees

Since derivation trees provide a record of how a given phrase structure tree is to be assembled, they implicitly contain all the information encoded in the latter. In itself this is a rather unremarkable fact, but in the MG community a trend has developed in the last 10 years to treat derivation trees as the primary data structure of MGs (Kobele *et al.* 2007; Kobele 2011, 2015; Graf 2011, 2012a,b, 2013; Hunter 2011). That is to say, MGs are no longer viewed as generators of phrase structure trees or strings but rather as a generator of derivation trees. A suitable graph transduction then transforms the derivation trees into the desired output structure – strings, phrase structure trees, logical forms, dependency graphs, and so on. Similar ideas have been explored in a more general setting under the label of two-step approaches (Morawietz 2003; Mönnich 2006), interpreted regular tree grammars (Koller and Kuhlmann 2011), and Abstract Categorical Grammar (de Groote 2001). This view of MGs has many technical advantages, but it also provides a unique perspective on parsing: if one assumes that the structures to be produced by an MG parser are derivation trees rather than

phrase structure trees, MG parsing turns out to be closely related to parsing of context-free grammars (CFGs).

It has been known for a while now that an MG's set of well-formed derivation trees forms a regular tree language (Michaelis 2001; Kobele *et al.* 2007; Salvati 2011; Graf 2012a).¹ Regular tree languages, in turn, can be directly linked to CFGs. For any CFG G , let $D(G)$ be the set of its derivation trees. Furthermore, a projection $\pi : \Sigma \rightarrow \Omega$ is a total function from alphabet Σ to alphabet Ω . Projections can be extended to trees in a point-wise fashion such that the image of tree t under π is the result of replacing each label in t by its image under π . A famous theorem by Thatcher (1967) states that a tree language L is regular iff there is a CFG G and projection π such that $L = \pi(D(G))$. In other words, every regular tree language can be generated by a CFG if one is willing to refine the node labels. For MGs, the refinement involves replacing all instances of Merge and Move with tuples of feature specifications. The details are of no particular interest here (see Michaelis 2001, Kobele *et al.* 2007 and section 2.1.1 of Graf 2013). The crucial point is that MGs have a close link to CFGs via their derivation trees, and this link can be exploited in parsing.

An MG parser can co-opt CFG parsing techniques as long as it has mechanisms to deal with the properties that separate MGs from CFGs. The use of Merge and Move as interior node labels instead of more fine-grained labels is rather trivial in this respect. The true challenge lies in the fact that the left-to-right order of leaves in the MG derivation tree does not correspond to the linear order in the string. The latter can be deduced from the former only if one keeps track of the structural alterations brought about by Move, which requires some ingenuity. At any rate, MGs can be regarded as CFGs with a more complex mapping

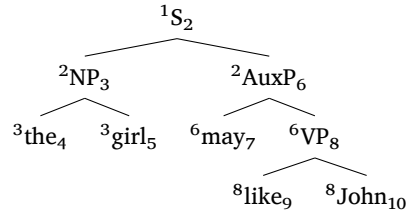
¹ This fact is not restricted to standard MGs as presented in the previous section. MGs have been extended and modified with numerous devices from the syntacticians' toolbox. Even the most truncated list includes adjunction (Frey and Gärtner 2002; Fowlie 2013; Hunter 2015a), new movement types (Kobele 2006; Stabler 2006; Gärtner and Michaelis 2010; Graf 2012b), and a variety of constraints (Gärtner and Michaelis 2007; Graf 2013). But these revised versions still preserve the regularity of the derivation tree language. The complexity of the string yield mapping is affected by new movement types, but stays within the restricted class of transductions that are definable in first-order logic with equality and proper dominance (Graf 2012b).

from trees to strings, and MG parsers are CFG parsers that have been augmented with a mechanism to handle this increased complexity.²

2.3 Encoding parses with tree annotations

Consider a standard recursive descent parser for CFGs, i.e. a parser that operates top-down, depth-first, and left-to-right. Following Kobele *et al.* (2013), the order in which a parser builds a given tree t for input string i can be represented by a specific annotation of the nodes of t as in Figure 5.

Figure 5:
The annotations of the tree indicate in what order it is built by a recursive descent parser



Intuitively, the annotation indicates for each node in the tree when it is first conjectured by the parser and at what point it is considered completed and flushed from memory. So at the very first step, the parser conjectures S , which is expanded in step 2 to NP and $AuxP$. Assuming that NP and $AuxP$ will eventually yield the desired string, S can be marked as done and removed from memory at step 2. After that the parser works on NP (because it operates left-to-right), adding *the* and *girl*. So those two are first conjectured at step 3 while NP is removed from memory at the same time. As the parser is depth-first, it now proceeds to work on *the* and *girl* rather than $AuxP$. First *the* is scanned at step 4. This means that the parser reads in the first word of

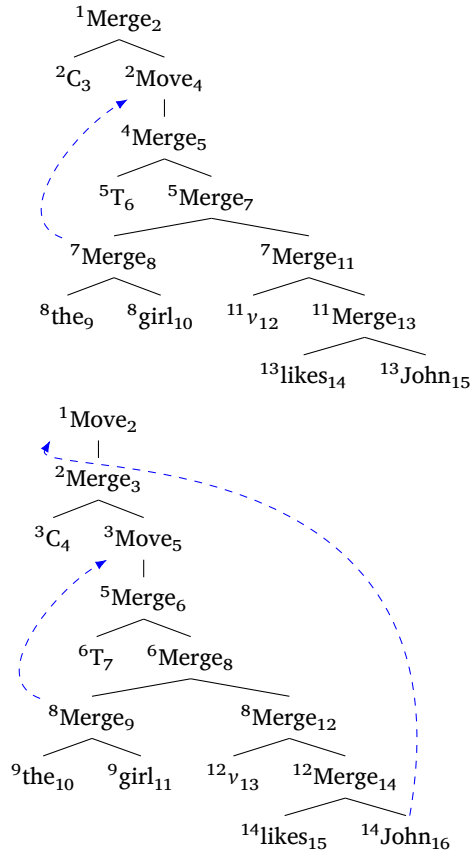
²The connection between MGs and CFGs does not emerge with phrase structure trees. MGs are known to be weakly equivalent to MCFGs (Harkema 2001; Michaelis 2001), from which it follows immediately that there are MGs whose set of well-formed phrase structure trees is supra-regular. But supra-regular tree languages cannot be made context-free via a simple relabeling as is the case for regular tree languages, wherefore CFG parsing techniques are not easily extended to Minimalist phrase structure trees. The technical gulf between phrase structure trees on the one hand and derivation trees on the other is significant, and it holds only because derivation trees need not directly encode the linear order of leaves in the string.

the input and verifies that it is indeed *the*. If so, the parser then scans *girl* at step 5, checking it against the second word in the string. Assuming that scanning succeeded, the parser then returns to AuxP, which it has held in memory since step 2. Now at step 6 it finally gets to flush AuxP from memory and replace it by *may* and VP. The remainder of the parse is straight-forward.

This is of course a highly abstracted view of the actual work done by a parser. For one thing, a parser operates with parse items rather than trees or tree nodes, and how such parse items are organized in memory depends on a lot on the specifics of the algorithm (for example, a chart-based parser would never remove an already constructed parse item from memory). More importantly, the problem of which rewrite rules must be chosen to derive the correct string is completely ignored. So this way of annotating trees is no substitute for a proper, rigorously defined parser. Crucially, though, these abstractions are immaterial for this paper's approach to modeling human sentence processing – the tree annotation is a sufficiently close representation of a parser's behavior to enable the kind of processing predictions we are interested in.

Now consider how a standard recursive descent parser would operate over an MG derivation tree. Consider first the derivation tree for *the girl likes John*, depicted in Figure 6. For the sake of clarity, we indicate unpronounced LIs by their category (C, T, ν). As can be gleaned from the figure, the parser scans the leaf nodes in this derivation in the following order: C T *the girl* ν *likes John*. But the actual order in the input string is C *the girl* T ν *likes John*, with *the girl* preceding T rather than following it. In this particular case the slight difference does not matter because T is the empty string, so “T *the girl*” and “*the girl* T” are exactly the same string. However, this example already shows that a standard recursive descent parser is not guaranteed to scan the leaf nodes of an MG derivation in the order in which they appear in the input string. Problems arise whenever Move actually alters the precedence relations between leaf nodes, as is the case with *John, the girl likes* (also shown in Figure 6). The recursive descent parser will reach *the* and try to scan it. It subsequently aborts the parse because the scanned leaf node does not match the first word in the input, *John*. We see, then, that a CFG recursive descent parser does not operate correctly over MG derivation trees despite them being context-free.

Figure 6:
Standard recursive descent works for
MG derivation trees only if Move does
not alter the precedence relations in
the string



The problem with the CFG recursive descent parser is its assumption that the left-to-right order in trees reflects the left-to-right order in the derived string. The core insight of Stabler (2013) (building on Main-guy 2010) is that the left-to-right order can instead be inferred from the MG feature calculus. At the level of abstraction used in this paper, the answer is even simpler. Given two sibling nodes m and n in an MG derivation, m is left of n iff m reflexively dominates a leaf node l such that every leaf node reflexively dominated by n is somewhere to the right of l in the derived string (where reflexive dominance is the reflexive, transitive closure of the mother-of relation). According to this definition, the recursive descent parser will choose a right branch instead of a left one whenever the right branch contains a mover and this mover appears to the left of all the material in the left branch.

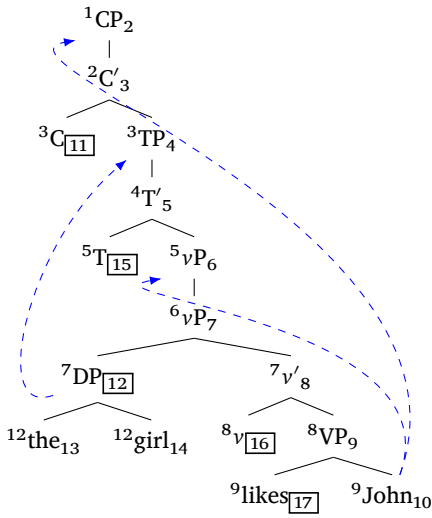


Figure 7:
Modified recursive descent operates correctly
over MG derivation trees

Figure 7 shows that this modified kind of recursive descent scans the leaf nodes in the correct order: *John C the girl T v likes*. To further increase the readability of derivation trees, this and all later figures replace the labels Merge and Move by the corresponding X'-labels in the phrase structure tree.

The annotations for MG derivation trees can be computed in a purely tree-geometric fashion (Graf *et al.* 2015). From here on out, we will refer to a node's superscript as its *index* and its subscript as its *outdex*. The terminology is intended to highlight that the index represents the step at which the parser first conjectures a node whereas the outdex records the point at which it has finished working on the node. Index and outdex thus provide information about the parser's memory usage. The greater the difference between the two, the longer an item has to be stored in memory. Since memory usage plays a central role in deriving processing predictions from these annotations, any outdex that is larger than the corresponding index by a non-trivial amount will be henceforth highlighted by a box (more on this in Section 3.2).

Definition 1 Let *s[urface]-precedence* be the relation that holds between nodes m_d and n_d in a derivation tree iff their counterparts m_p and n_p in the corresponding phrase structure tree stand in the precedence relation. If m_d undergoes movement during the derivation, its counterpart m_p is the final landing site rather than its base position.

Given a well-formed Minimalist derivation tree t , its index/outdex annotation is computed as follows:

1. Every node of t has exactly one index and exactly one outdex.
2. The index of the root is 1. For every other node, its index is identical to the outdex of its mother.
3. If nodes n and n' are distinct nodes with index i , and n reflexively dominates a node that is not s -preceded by any node reflexively dominated by n' , then n has outdex $i + 1$.
4. Otherwise, the outdex of node n with index i is $\max(i+1, j+1)$, where $j \geq 0$ is greatest among the outdices of all nodes that s -precede n but are not reflexively dominated by n .

Throughout the rest of the paper we use these annotated derivation trees as abstract representations of the behavior of Stabler's (2013) recursive descent parser for MGs. This greatly simplifies the discussion by substituting easily interpreted derivation trees with indices and outdices for the complex mechanics of the parser. But it means that the difficulty of finding this derivation tree in the first place is completely ignored. The most demanding task of parsing – searching through a large space of structures in the search for the correct one – is taken out of the equation. This simplification is shared among all recent work that use Stabler's MG parser to model human processing (Kobele *et al.* 2013; Graf and Marcinek 2014; Graf *et al.* 2015; Gerth 2015). In the words of Graf *et al.* (2015, p.3):

While psychologically implausible, this idealization is meant to stake out a specific research goal: processing effects must be explained purely in terms of the syntactic complexity of the involved structures, rather than the difficulty of finding these structures in a large space of alternatives. More pointedly, we assume that parsing difficulty *modulo* non-determinism is sufficient to account for the processing phenomena under discussion.

The aim of these MG processing models, then, is to see how much of human sentence processing can be explained by considering only the order of how the parts of the correct derivation are built. This does not deny that ambiguity has a large role to play, e.g. in garden path

sentences, but it is taken out of the equation in order to determine the relevance of isolated structural factors. A simpler model has the advantage of being easier to reason about, and the focus on structure allows us to compare specific syntactic proposals according to their processing predictions.

3 COMPLEXITY METRICS FOR PROCESSING

The previous section recast Stabler's top-down parsing algorithm for MGs as a particular kind of tree annotation, but this raises the question how a simple annotation of derivation trees can be linked to psycholinguistic processing effects. This is accomplished via a linking theory, which takes the form of *complexity metrics*.³ The next section discusses what we mean by complexity metrics and how all our metrics are rooted in notions of memory usage. Sections 3.2 and 3.3 then provide formal definitions of all the relevant metrics. The full set comprises 1,600 metrics, of which only a handful will prove able to account for all the data in Section 4.

3.1 *Complexity metrics and three notions of memory usage*

A complexity metric is any procedure that ranks strings according to processing difficulty. For instance, Kimball's (1973) principle that the human parser cannot work on more than two CPs at the same time provides a simple complexity metric that is computed over phrase structure trees. O'Grady (2011) suggests that the length of movement dependencies affects processing difficulty. The Derivational Theory of Complexity (Miller and Chomsky 1963; Miller and McKean 1964; see also Phillips 1996, Chapter 5) equates complexity with the number of syntactic operations that are required to build said sentence. Syntactic Prediction Locality Theory (Gibson 1998) and Dependency Locality Theory (Gibson 2000) instead operate directly over the string and measure the length and interaction of certain dependencies. There are also metrics that consider more than one isolated structure: surprisal and entropy reduction (Hale 2001, 2003, 2011; Levy 2013), for

³Following the advice of two reviewers, we refrain from using Graf *et al.*'s (2015) term *parsing metric*, which already has an established but distinct meaning in the formal parsing community.

instance, measure how the search space shrinks and grows during incremental processing.

The open-endedness of complexity metrics reflects the fact that the number of conceivable linking theories between the parser and the observed processing phenomena is dauntingly large. In the face of such an overabundance of choices, the methodologically soundest position is to explore simple metrics before moving on to more complicated ones. This is the stance we adopt throughout this paper. The MG parser has already been simplified to a degree where all ambiguity is abstracted away and parsing is reduced to index/outdex annotations of derivation trees. Sticking with our focus on derivation trees and maximal simplicity, we only consider complexity metrics that predict processing difficulty based on how the geometry of derivation trees affects memory usage.

That processing difficulty correlates with memory usage is a very common hypothesis in the psycholinguistic literature. The idea can be traced back to Kaplan (1974) and Wanner and Maratsos (1978),⁴ with Joshi (1990), Gibson (1998, 2000) and many other as more recent examples (see Gerth (2015, Section 2.3.1) for a detailed discussion). Memory usage may be measured in many different ways, though, and as a result there is a myriad conceivable complexity metrics that differ only in minor details. This paper compares over a thousand memory-based complexity metrics, but fortunately they can be reduced to three basic concepts, which will be gradually refined and modified as we go along.

As we briefly remarked in Section 2.3, the MG parser does not actually hold nodes of the derivation tree in memory but rather *parse items* that encode various pieces of information about each node, in particular whether the node is the root of a subtree containing movers. For a parser that has to hold such parse items in memory, one can distinguish at least three kinds of memory usage:

Tenure How long is the item kept in memory?

Payload How many items are held in memory?

Size How many bits does the item consume in memory?

⁴We thank an anonymous reviewer for bringing these early works to our attention.

Each category is part of some complexity metric that has been invoked in previous work on MG parsing (Kobele *et al.* 2013; Graf and Marcinek 2014; Graf *et al.* 2015; Gerth 2015). In terms of annotated derivation trees, the three notions can be formalized as follows:

Definition 2 *Let t be some Minimalist derivation tree annotated with indices and outdices.*

- *For every node n with index i and outdex o ($i < o$), its tenure $\text{ten}(n)$ is $o - i$. A node's tenure is trivial iff $\text{ten}(n) \leq 2$.*
- *The payload of t is equal to the number of nodes in t with non-trivial tenure: $|\{n \mid \text{ten}(n) > 2\}|$.*
- *For every node n its size is identical to the number of phrases that are reflexively dominated by n , distinct from n , and are associated to a Move node that reflexively dominates n .*

For a concrete example, consider again the derivation tree in Figure 7. The tenure of the empty C-head is $11 - 3 = 8$, whereas the tenure of TP is just $4 - 3 = 1$. The derivation tree's payload is 5 as there are five nodes with non-trivial tenure (indicated by boxed outdices): the empty C-, T-, and v -heads, as well as DP and *likes*. The size of a parse item corresponding to node n is the same as the number of nodes below n that have a movement arrow pointing to somewhere above n . So the size of CP and v' is 1 and the size of T' is 2, whereas the size of DP and v is 0.

The definition of size may strike the reader as very stipulative. It derives from how information about movers is stored by Stabler's (2013) top-down parser. For a detailed discussion, the reader is referred to Graf *et al.* (2015). Similarly, readers may wonder why the threshold for payload is set to 2 rather than 1. Once again this is done for technical reasons, discussed in Graf and Marcinek (2014).

3.2 *From memory usage to complexity metrics*

Note that tenure, size and payload are not exactly on equal footing. While payload is a property of derivation trees, tenure and size are properties of individual nodes/parse items. Consequently, payload can already be used as a complexity metric for our simple purposes: given two derivation trees, the one with lower payload is predicted to be easier to process. Graf and Marcinek (2014) use the name **Box** to distinguish payload as a complexity metric over derivation trees from

payload as general concept of memory usage. The name is motivated by the notational convention to draw a box around the outdices of nodes with non-trivial tenure, which we also adhere to in this paper. In contrast to payload, tenure and size can be applied to derivation trees in multiple ways.

Tenure was incorporated into three distinct complexity metrics by Kobele *et al.* (2013). Let T be the set of nodes of derivation tree t . Then

$$\mathbf{MaxT} \quad \max(\{\text{ten}(n) \mid n \in T\})$$

$$\mathbf{SumT} \quad \sum_{n \in T, \text{ten}(n) > 2} \text{ten}(n)$$

$$\mathbf{AvgT} \quad \frac{\mathbf{SumT}(t)}{\mathbf{Box}(t)}$$

So **MaxT** reports the maximum memory usage used by any single one node, **SumT** the total (non-trivial) tenure of the entire derivation tree, and **AvgT** the average memory usage of a node with non-trivial tenure. Recall that the derivation in Figure 1 has a payload of 5, which is also its **Box** value. Moreover we have **MaxT** = 10 (due to T), **SumT** = 8 + 10 + 5 + 8 + 8 = 39 (summing the tenure of all boxed nodes), and **AvgT** = $\frac{39}{5} = 7.8$.

Graf *et al.* (2015) furthermore generalize size to the complexity metric **Gap** in a fashion that mirrors **SumT** for tenure. To highlight this similarity, we call this metric **SumS** instead of **Gap** (the original name is motivated by the parallels to measuring the length of filler-gap dependencies). Let M be the set of all nodes of derivation tree t that are the root of a subtree undergoing movement. Also, for each $m \in M$, $i(m)$ is the index of m and $f(m)$ is the index of the highest Move node that m 's subtree is moved to. With the visual aids in our derivation trees, M can be taken to consist of exactly those nodes that are the starting point of an arrow, while $f(m)$ is the target node of the highest arrow that starts in m . **SumS** sums the differences between these indices.

$$\mathbf{SumS} \quad \sum_{m \in M} i(m) - f(m)$$

Considering once more the derivation tree in Figure 1, we see that $M = \{\text{DP}, \text{John}\}$, $i(\text{DP}) - f(\text{DP}) = 7 - 3 = 4$, and $i(\text{John}) - f(\text{John}) = 9 - 1 = 8$. So the whole derivation tree has a **SumS** value of 12. The motivation behind **SumS** is again hard to convey without drilling deep into the bowels of the MG top-down parser. Intuitively, though, **SumS**

expresses the idea (independently argued for in O’Grady 2011) that moving a subtree is computationally expensive – the longer it takes to actually get to the subtree that needs to be moved, the higher the resource cost.

Even though **SumS** is transparently a size-based analog of **SumT**, no complexity metrics have been previously proposed for size that operate similar to **Box**, **MaxT** or **AvgT**. We introduce these metrics here for the sake of completeness, even though they will eventually turn out to be inadequate for sentence processing.

Movers $|M|$, where M is the set of all nodes that are the root of a subtree undergoing movement

MaxS $\max(\{i(n) - f(n) \mid n \in T\})$, where T is the set of all nodes of the derivation tree

AvgS $\frac{\text{SumS}(t)}{\text{Movers}(t)}$

For the example in Figure 7, we have **Movers** = 2 (only subject and object move), **MaxS** = 8 (topicalization of *John*), and **AvgS** = $\frac{(9-1)+(7-3)}{2} = 6$.

3.3 Further refinements

3.3.1 Recursive application

Another metric briefly mentioned in Graf and Marcinek (2014) is **MaxT^R**, which applies **MaxT** recursively. With **MaxT**, two derivation trees may receive exactly the same score and would thus be predicted to be equally difficult. **MaxT^R** instead assigns each derivation tree a weight that enumerates in decreasing order the tenure of all nodes in the payload. Then derivation u is easier than derivation v iff their weights are identical up to position i , at which point u ’s weight contains a smaller number than v ’s weight. A similar strategy can also be used for size, yielding the complexity metric **MaxS^R**.

Our example derivation tree has the values **MaxT^R** = [10, 8, 8, 8, 5] and **MaxS^R** = [8, 4]. Therefore it would be harder than a competing derivation with **MaxT^R** = [10, 8, 8, 5], but easier than one with **MaxS^R** = [8, 3].

3.3.2 Restriction by node type

Graf and Marcinek (2014) refine the set of metrics even more by relativizing them to specific types of nodes. For each metric \mathbf{M} , an additional four variants \mathbf{M}_I , \mathbf{M}_L , \mathbf{M}_P , and \mathbf{M}_U are defined.

\mathbf{M}_I restriction of metric \mathbf{M} to interior nodes

\mathbf{M}_L restriction of metric \mathbf{M} to leaf nodes

\mathbf{M}_P restriction of metric \mathbf{M} to pronounced leaf nodes

\mathbf{M}_U restriction of metric \mathbf{M} to unpronounced leaf nodes

For instance, the unrestricted \mathbf{MaxT} value of our derivation was 10, but the refined values are $\mathbf{MaxT}_I = 5$ (on DP), $\mathbf{MaxT}_L = 10$ (on T), $\mathbf{MaxT}_P = 8$ (on *likes*), and $\mathbf{MaxT}_U = 10$ (on T).

Note that these restrictions make little sense for size-based metrics since moving subtrees usually contain pronounced material and the corresponding Move node is necessarily an interior node. Therefore we do not consider type-based restrictions of size metrics. At this point, then, the set of defined metrics includes \mathbf{Box} , \mathbf{MaxT} , \mathbf{SumT} , \mathbf{AvgT} , \mathbf{MaxT}^R , four restricted subtypes for each one of them, as well as the size-based metrics \mathbf{Movers} , \mathbf{MaxS} , \mathbf{SumS} , \mathbf{AvgS} , and \mathbf{MaxS}^R (for a total of 30 metrics).

3.3.3 Time course of memory usage

The final metric to be considered refines payload so that it reflects maximum memory usage more faithfully. As we saw earlier, \mathbf{Box} simply reports how many parse items had to be held in memory. However, a high \mathbf{Box} value need not imply a heavy memory burden as long as one item is removed from memory before the next one is inserted. That is to say, if nodes u and v contribute to the payload of derivation t but the outdex of u is less than the index of v , then the two never reside in memory at the same time. In order to home in on this aspect, we define two metrics *convergence* \mathbf{Con} and *divergence* \mathbf{Div} that keep track of how many distinct nodes do or do not reside in memory at the same time.

\mathbf{Con} $|\{\langle u, v \rangle \mid \text{ten}(u) \geq 2, \text{ten}(v) \geq 2, \text{index}(u) \leq \text{index}(v) \leq \text{outdex}(u)\}|$

\mathbf{Div} $|\{\langle u, v \rangle \mid \text{ten}(u) \geq 2, \text{ten}(v) \geq 2, \text{outdex}(u) < \text{index}(v)\}|$

As before these metrics can be relativized to the four subtypes **I**, **L**, **P**, and **U**. Returning one final time to the derivation in Figure 7, we see that $\mathbf{Con}_U = |\{\langle C, T \rangle, \langle C, v \rangle, \langle T, v \rangle\}| = 3$ and $\mathbf{Div} = |\emptyset| = 0$.

3.3.4 Ranked complexity metrics

With just a handful of psycholinguistically plausible factors such as maximum and average memory usage and restrictions to specific types of nodes the number of metrics has quickly risen to a bewildering degree. But things do not stop here. Graf *et al.* (2015) argue in favor of a combined metric **MaxT** + **SumS** which uses **MaxT** to predict processing difficulty but relies on **SumS** whenever **MaxT** results in a tie. So given two derivation trees t_1 and t_2 , t_1 is predicted to be easier than t_2 if either t_1 has a lower **MaxT** or t_1 and t_2 have the same **MaxT** value but t_1 has a lower **SumS** value. This is similar to constraint ranking in OT (Prince and Smolensky 2004), where a lower ranked constraint matters only if all higher ranked constraints have failed to pick out a unique winner. If complexity metrics are allowed to be ranked in such a way, their number quickly reaches an astronomical size. We have introduced 40 metrics, wherefore a ranked complexity metric can consist of up to 40 metrics. It follows that there are over 40 factorial (40!) distinct metrics that are ranked combinations of our 40 basic metrics – a truly astounding number. Even if one only allows pairs of our 40 complexity metrics, there are 1,600 distinct metrics (pairs of the form $\langle m, m \rangle$ are equivalent to just the metric m).

Ranked Metric Given a set C of complexity metrics, a ranked metric is an n -tuple $\langle c_1, \dots, c_n \rangle$ such that for $1 \leq i, j \leq n$ it holds that $c_i \in C$ and that $i \neq j$ implies $c_i \neq c_j$. Given a ranked metric $\langle c_1, \dots, c_n \rangle$ and two derivation trees t_1 and t_2 , t_1 is predicted to be easier than t_2 iff there is some $j \leq n$ such that $c_i(t_1) = c_i(t_2)$ for all $i \leq j$ and c_j predicts t_1 to be easier than t_2 .

3.4 Discussion

The large number of metrics poses a significant problem. Remember the promise of the MG parser and the psycholinguistic modeling work that builds on it: processing phenomena are explained in terms of the syntactic structures they involve, and in the other direction, syntactic analyses can be evaluated based on their processing predictions.

But the processing claims of these models arise from the interaction of three factors, which are the parser (represented via index/outdex annotation), the syntactic analysis (in the form of derivation trees), and the complexity metric.

There are few alternatives to the current top-down parser. Despite some suggestive evidence such as merely local syntactic coherence effects (Tabor *et al.* 2004; Konieczny 2005; Konieczny *et al.* 2009; Bicknell *et al.* 2009), there is still a large consensus among psycholinguists that if the human parser builds any kind of tree structures, it does not do so in a pure bottom-up fashion. The other prominent option is left-corner parsing. Unfortunately, no left-corner parser exists for MGs at this time because the notion *left corner* does not carry over neatly from CFGs (but see Hunter 2015b). Without a readily available alternative to top-down parsing, the two major parameters in the model are the choice of metric and the choice of syntactic analyses. But the larger the set of metrics, the higher the risk that just about any syntactic analysis will make the right predictions with some metric. This would significantly weaken the link between syntactic structure and processing effects, which is the very heart of the work carried out by Kobele *et al.* (2013), Graf and Marcinek (2014), Graf *et al.* (2015), and Gerth (2015).

Fortunately, this worst-case scenario does not seem to arise. It turns out that a few constructions involving relative clauses are sufficient to rule out the vast majority of these metrics. We have no principled explanation as to why this is the case – it is far from a logical necessity. But this result, established in the next section, strengthens the viability of the enterprise started by Kobele *et al.* (2013) to model processing phenomena with MGs and use these findings to distinguish competing Minimalist analyses. It demonstrates that I) a very simplified processing model can still account for a noteworthy range of challenging processing phenomena, and II) the set of workable complexity metrics is small enough to give the model discriminative power with respect to syntactic analyses.

4 TESTING METRICS WITH RELATIVE CLAUSES

Now that we have a precisely defined parsing model (abstractly represented in terms of annotated derivation trees) as well as a collection of

complexity metrics that link the parser's behavior to processing predictions, we are finally in a position to investigate how well these tools model a collection of phenomena from human sentence processing. All these phenomena, which will be presented in detail in Section 4.1, involve relative clauses (RCs) to some extent and have been studied separately in Kobele *et al.* (2013), Graf and Marcinek (2014), and Graf *et al.* (2015).⁵ In contrast, we consider the full data set and test our much bigger collection of metrics against each one of them. We furthermore compare two competing analyses of RCs (4.2) using a fairly simple methodology of automated comparisons (4.3). We conclude that this small set of data points is highly discriminative in that it rules out a large number of metrics for each analysis (4.4) while still allowing for linguistically natural explanations of the observed patterns (4.5).

4.1 *Overview of relative clause constructions*

Our testing data for the comparison of metrics and syntactic proposals relies on several well-known processing contrasts involving RCs. RCs are a promising test case because they are complex enough to allow for syntactically interesting structures while factoring out aspects that aren't purely structural in nature such as co-reference resolution. The general idea is to take a pair of constructions A and B such that A is easier to process than B. This result then has to be replicated by the complexity metrics given a specific analysis of RCs.

The specific behavioral contrasts to be accounted for were chosen according to several criteria. First, the processing effects must be well-documented in the psycholinguistic research. Second, the phenomenon should involve a clear structural contrast, rather than just a meaning contrast (e.g. pronoun resolution). Third, ambiguity should not be a major factor, which rules out garden path effects.

1. **SC/RC < RC/SC**

A sentence with a relative clause embedded inside a sentential complement (SC/RC) is easier to parse than a sentence with a

⁵Gerth (2015) investigates some additional phenomena which were not included in our data sample as we were not aware of her findings until recently, unfortunately.

sentential complement embedded inside a relative clause (RC/SC; Gibson 1998, 2000).

2. SRC < ORC

- Subject relative clauses (SRCs) are easier to parse than object relative clauses (ORCs) in languages like English, where relative clauses are post-nominal and therefore follow their head noun (Mecklinger *et al.* 1995; Gibson 1998, 2000; Mak *et al.* 2002, 2006; Gordon *et al.* 2006).
- SRCs are also easier to parse than ORCs in Chinese, Korean, and Japanese, where relative clauses are pre-nominal, that is to say, they precede their head noun (Miyamoto and Nakamura 2003, 2013; Lin and Bever 2006; Ueno and Garnsey 2008; Kwon *et al.* 2010; Gibson and Wu 2013).

3. Right < Center

Right embedding is easier than center embedding.

These generalizations have been carefully established in the literature via self-paced reading experiments and ERP studies with minimal pairs such as the ones listed in (1)–(6).

(1) SC/RC < RC/SC

- a. The fact [_{SC} that the employee_i [_{RC} who the manager hired t_i] stole office supplies] worried the executive.
- b. The executive_i [_{RC} who the fact [_{SC} that the employee stole offices supplies] worried t_i] hired the manager.

(2) SRC < ORC in English

- a. The reporter_i [_{RC} who t_i attacked the senator] admitted the error.
- b. The reporter_i [_{RC} who the senator attacked t_i] admitted the error.

(3) SRC < ORC in Chinese

- a. [_{RC} t_i gongji yiyuan] de jizhe chengren-le cuowu
attack senator REL reporter admit-PRF error
- b. [_{RC} yiyuan gongji t_i] de jizhe chengren-le cuowu
senator attack REL reporter admit-PRF error

(4) SRC < ORC in Korean

- a. [_{RC} t_i uywon-ul kongkyekha-n] kica_i-ka
senator-ACC attacked-REL reporter-NOM
silswu-lul siinhayssta
error-ACC admitted
- b. [_{RC} uywon-i t_i kongkyekha-n] kica_i-ka
senator-NOM attacked-REL reporter-NOM
silswu-lul siinhayssta
error-ACC admitted

(5) **SRC < ORC in Japanese**

- a. [_{RC} t_i giin-ga hinanshita] kisha_i-ga ayamari-o
senator-ACC attacked reporter-NOM error-ACC
mitometa
admitted
- b. [_{RC} giin-ga t_i hinanshita] kisha_i-ga ayamari-o
senator-NOM attacked reporter-NOM error-ACC
mitometa
admitted

(6) **Right < Center**

- a. The boy disappeared [_{RC} who the man saw [_{RC} who the woman praised]].
- b. The boy [_{RC} who the man [_{RC} who the woman praised] saw] disappeared.

It should be pointed out that the SRC preference is less robust in Chinese than Korean or Japanese. This has been attributed to structural ambiguities (Gibson and Wu 2013), which is corroborated by Yun *et al.* (2014) and their ambiguity-based account rooted in entropy reduction. Recall from Section 2.3, though, that we deliberately ignore ambiguity in this paper so that only tree-geometric aspects of the derivation can derive processing effects. For this reason, we assume that Chinese would also exhibit a uniform preference for SRCs over ORCs if it were not for the confound of structural ambiguity.

Some of the contrasts above have previously proven difficult to account for. While the preference for SC/RC and SRC in English can be explained by string-based models such as the Dependency Locality Theory (Gibson 1998) or the Active-Filler strategy (Frazier 1987),

these models erroneously derive an ORC preference for East Asian languages with their pre-nominal RCs. This is because the head noun is closer to the object than the subject position of the RC in this case. A functional account like Keenan and Comrie's (1977) accessibility hierarchy, on the other hand, derives the SRC preference across languages but has little to say about the ease of SC/RC in comparison to RC/SC. That right embeddings are much easier than center embeddings has an elegant explanation in terms of left-corner parsing (Resnik 1992), but this account in turn does not generalize to the other configurations. Overall, then, the data points above have been accounted for individually, but their unification is challenging.

4.2 *Promotion and wh-analysis of relative clauses*

As one of the promises of the MG processing model is the ability to distinguish syntactic analyses based on their processing predictions, our evaluation uses two popular proposals for the structure of RCs: the *wh-movement* analysis (Chomsky 1965, 1977; Montague 1970; Heim and Kratzer 1998), and the *promotion* analysis (Vergnaud 1974; Kayne 1994). Graf et al. (2015) did the same in their investigation of the SRC preference in East Asian, whereas Kobele et al. (2013) and Graf and Marcinek (2014) only used a promotion analysis.

Both the promotion analysis and the *wh-analysis* posit that the gap inside the RC is initially filled by some element, but disagree on what this element is and where it moves. In the promotion analysis, it is the head noun itself that starts from the gap position. The *wh-analysis* has two variants. Either the relative pronoun⁶ moves from the gap position, or it acts as the C-head of the RC while a silent operator undergoes movement from the base position. For the purposes of this paper the two variants of the *wh-movement* analysis are fully equivalent.

⁶The use of "relative pronoun" is slightly misleading here in that the relative clause markers in Chinese and Korean are not pronouns (as is rightfully noted by an anonymous reviewer). But since the syntactic category of LIs is ignored by all complexity metrics, we freely change between the terms relative pronoun and RC marker in the discussion. We also represent the East Asian RC markers with *who* in the derivation trees in an attempt to ease the comparison to the English derivation trees.

Notably absent are proposals that do not involve any movement at all. This is because in the absence of movement, the MG parser behaves exactly like a recursive descent parser for CFGs and thus would have little new to offer. In addition, the comparison and detailed analysis of the complexity metrics already involves a multitude of factors, so that increasing the number of analyses would run the risk of rendering the discussion (even more) impenetrable.

With both the promotion analysis and the wh-analysis, the target of movement depends on whether RCs are post-nominal or pre-nominal in the language under investigation. Let us consider languages with post-nominal RCs like English, French, and German. All these languages also have overt complementizers, although they may optionally remain unpronounced, as is the case in English. The general template is [_{DP} Det head-noun [_{RC} complementizer subject verb object]], with either the subject or the object unrealized. The position of the verb depends on language-specific word order constraints, but we can safely ignore this aspect because English is the only language with post-nominal RCs in our data set. Figures 8 and 9 show the promotion analysis and the wh-analysis, respectively, for the SRC *The reporter who attacked the senator admitted the error*. In both derivation trees the element that fills the gap in the SRC moves to the CP specifier (Spec,CP), i.e. the left edge of the relative clause. But note that the head noun is outside the RC in the wh-movement analysis, whereas it is in Spec,CP (and thus inside the RC) in the promotion analysis.

The only difference between SRC and ORC under these analyses is the position that the mover occupies initially. In the SRC, the mover fills the base position of subjects (equated with Spec,_vP here), whereas the ORC requires the mover to start out in object position (i.e. as the VP complement). This is illustrated in Figure 10, which depicts an ORC with an embedded sentential complement.

Languages with pre-nominal RCs, such as Chinese, Japanese, and Korean, can also be accommodated, but the word order differences render both analyses more complex. Below is an example of pseudo-English SRCs and ORCs with Chinese word order.

- (7) a. [_{DP} [_{RC} _ invited the tycoon who] the mayor] likes wine.
b. [_{DP} [_{RC} the tycoon invited _ who] the mayor] likes wine.

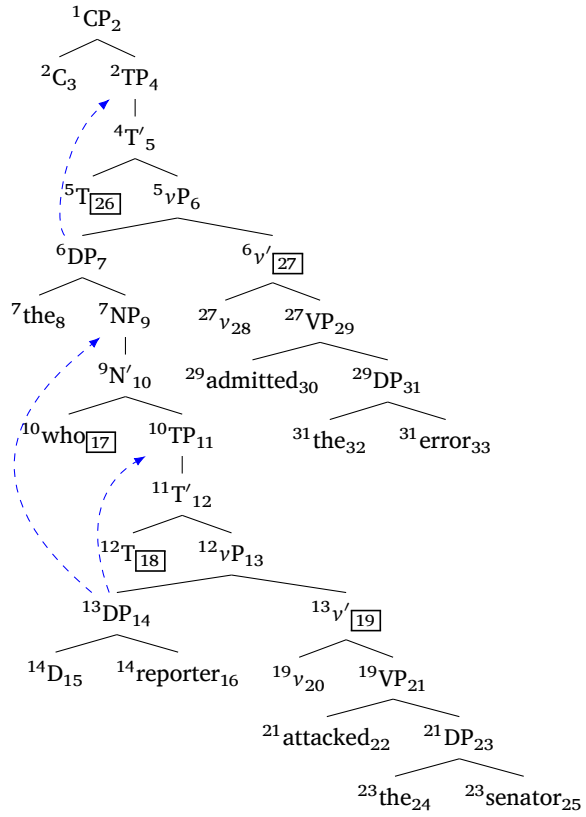


Figure 8: English SRC, promotion analysis

Since standard MGs do not provide a headedness parameter to determine the linearization of arguments (following the received view in Minimalist syntax), the pre-nominal word order must be derived from the post-nominal one via movement. This causes the wh-movement analysis and the promotion analysis to diverge more noticeably.

In the promotion analysis, the RC is no longer a CP, but rather a RelP that contains a CP (see also Yun *et al.* 2014). The head noun still moves from within the RC to Spec,CP, but this is followed by the TP moving to Spec,RelP. This creates the desired word order with the complementizer between the rest of the RC in Spec,RelP and the head noun in Spec,CP. In the wh-movement analysis, on the other hand, the head noun is once again outside the RC, which is just a CP instead of a RelP. The complementizer starts out in subject or object position

Relative clauses as a benchmark for Minimalist parsing

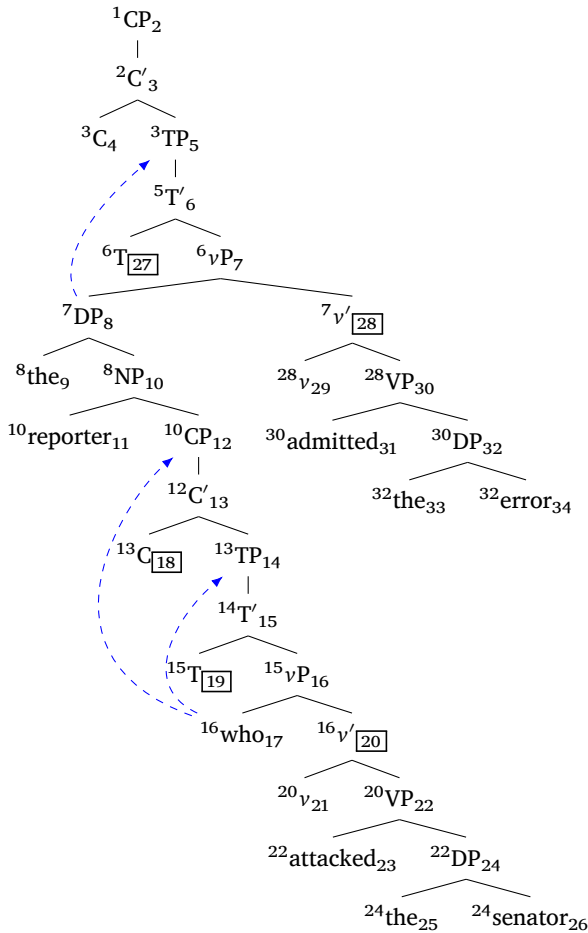


Figure 9: English SRC, wh-movement analysis

depending on the type of RC, and then moves into a right specifier of the CP (rightward movement is not part of Stabler’s (2013) MG parser, but we can easily add it without modifying the annotation rules from Definition 1 as they are defined in terms of s-precedence). The CP subsequently moves to the specifier of the DP of the head noun, once again yielding the desired word order with the complementizer between the RC and the head noun. In sum, the promotion analysis needs to posit a new phrase RelP but all movement is leftward and takes place within this phrase. This contrasts with the wh-movement

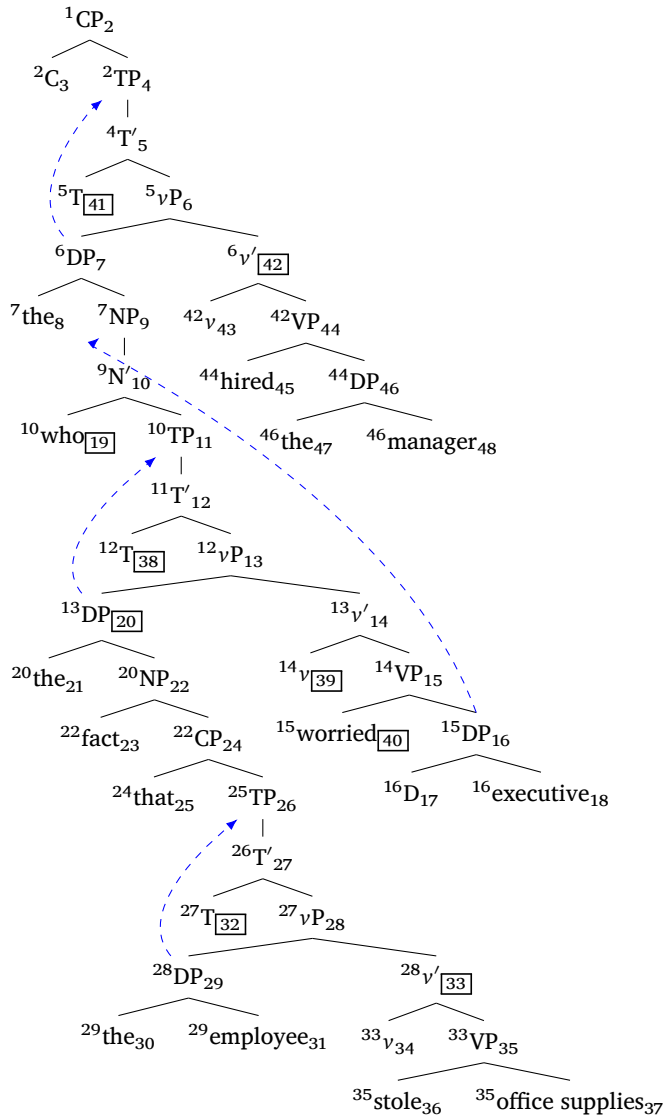


Figure 10: ORC containing a sentential complement, promotion analysis

analysis, which sticks with a single CP but invokes one instance of rightward movement and moves the RC into Spec,DP, a higher position than Spec,RelP. Examples of the two derivation trees for a Chinese SRC are given in Figures 11 and 12, where dotted arrows are used instead of dashed ones for rightward movement.

Among the three East Asian languages, Chinese still has the simpler analysis thanks to its SVO word order, whereas Japanese and Korean are SOV languages. As was the case with the linear order of RCs relative to their head noun, Minimalist syntax assumes that the SOV word order is derived via Move rather than simply linearizing the complement of the verb to its left. The standard assumption is that SOV

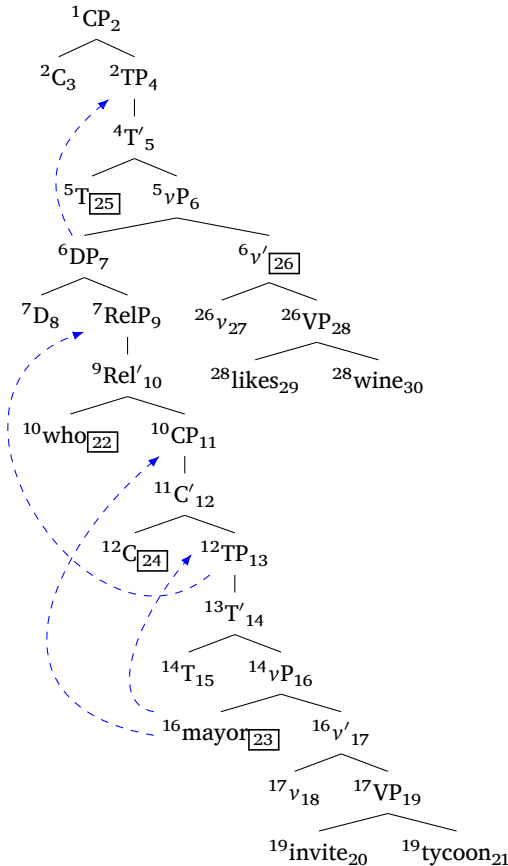


Figure 11: SRC in Chinese, promotion analysis

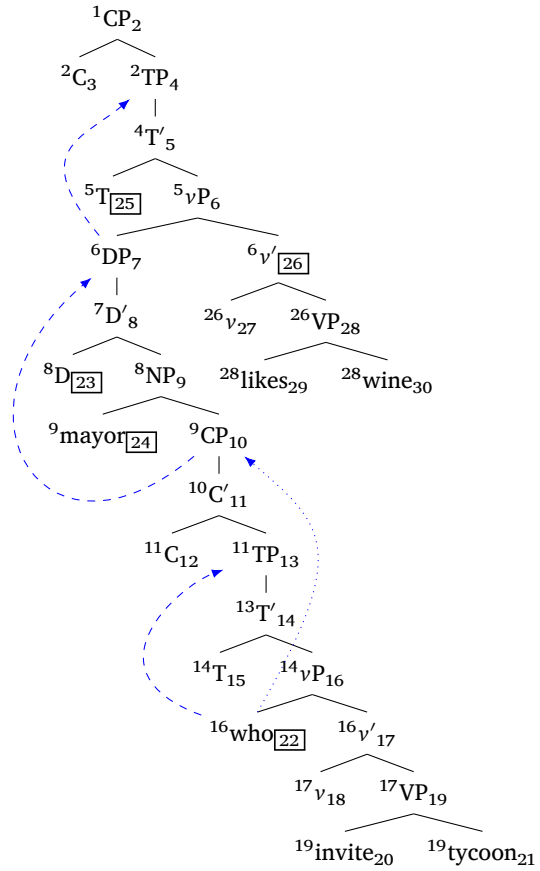


Figure 12: SRC in Chinese, wh-movement analysis

languages require the object to move from the VP-complement position to a specifier of vP as exemplified in Figure 13. While this might seem like a minor complication, we will see in the next section that it actually causes many metrics to incorrectly prefer ORCs over SRCs. Korean and Japanese thus show that the complexity metrics are indeed exquisitely sensitive to minor structural alterations.

We also use rightward movement in right embedding constructions (Figure 14), as embedding without additional movement yields center embedding structures. Whether these instances of extraposition are best analyzed as rightward movement has been called into question in recent research (Hunter and Frank 2014), but it is the best

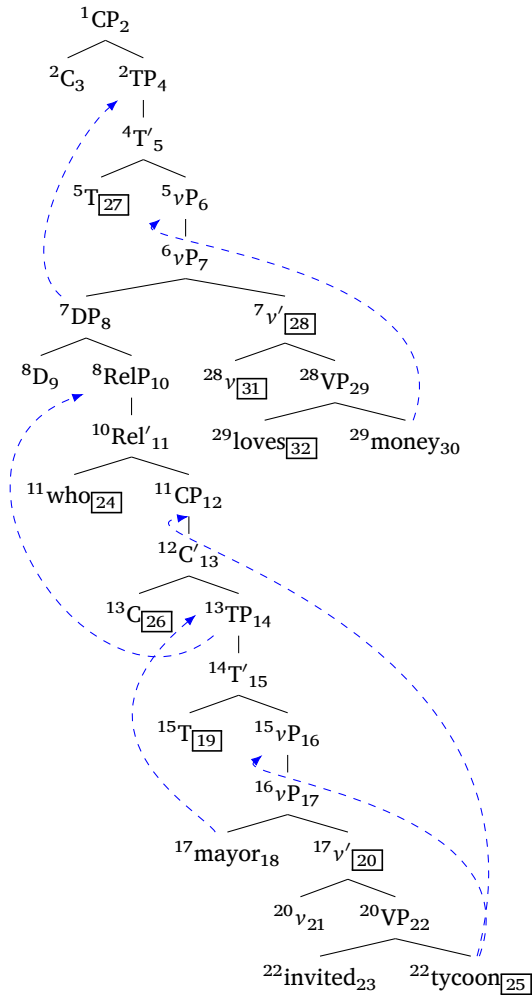


Figure 13: ORC in Korean, promotion analysis

choice here to maintain analytical consistency across the different constructions.

For a full listing of all the analyses with annotated derivation trees, the reader is referred to the supplementary material for this article. Derivation trees for Japanese are omitted since they are identical to the Korean ones except that the RC complementizer remains unpronounced.

4.3

Methodological remarks

As 1,600 metrics cannot be accurately compared by hand, we rely on a collection of Python scripts, available in the Github repository of the Stony Brook Computational Linguistics lab: <https://github.com/CompLab-StonyBrook>. For each basic metric, these scripts perform pairwise comparisons of minimally different derivation trees, e.g. the English SRC in Figure 8 and its ORC counterpart. Whichever one receives a better (= lower) score has a lower memory burden is thus predicted to be easier to process. From the relative rankings that are obtained this way one can then automatically compute all the metrics – including combinations of multiple metrics – that correctly predict all processing contrasts.

Note that the difficulty metric only has to account for overall sentence difficulty. This is different from more ambitious approaches such as Hale (2001) and Yun *et al.* (2014), which seek to predict online difficulty, i.e. how difficulty increases or decreases with each word in the input. Modeling online processing is feasible with certain complexity metrics like **MaxT** (see Kobele *et al.* 2013 and Gerth 2015), but it is hard to automatically compare metrics at this level of granularity. Finally, we reiterate that all ambiguity is factored out – we only consider how the parser builds a specific derivation tree, rather than how it finds this tree among many alternatives.

4.4

Quantitative evaluation of complexity metrics

The performance of the basic metrics with the respective syntactic analyses is summarized in Tables 1 and 2. A checkmark (✓) indicates that the metric correctly predicts structure A to be easier than structure B, a tie that they are expected to be equally difficult, and a cross (✗) that the complexity metric incorrectly reverses difficulty, making B easier than A. Consequently, all basic metrics that contain a cross in at least one column can be discarded. This leaves only one metric for the promotion analysis – **MaxT_U^R** – and one for the wh-movement analysis – **AvgT_P**.

Many inadequate basic metrics, though, may still occur as the second component in a ranked metric. As the second component is only invoked to handle ties for the first component, wrong predictions for a given contrast have no effect unless the first component could not conclusively resolve this contrast. When ranked metrics are also taken

Table 1:
Predictions of
complexity
metrics with
promotion
analysis

	SC/RC < RC/SC	Eng	SRC < ORC			Right < Center
			Chi	Kor	Jap	
Box	tie	✓	✓	tie	tie	✗
Box_I	tie	tie	✓	✓	✓	✗
Box_L	tie	✓	✓	✗	✗	tie
Box_P	tie	✓	tie	✗	✗	✓
Box_U	✓	✓	✓	tie	tie	✗
AvgT	✓	✓	✗	✗	✗	✓
AvgT_I	✗	✓	✗	✗	✗	✗
AvgT_L	✓	✓	✗	✓	✓	✓
AvgT_P	✓	✓	✗	✗	✗	✓
AvgT_U	✓	✓	✗	✓	✓	✓
MaxT	tie	tie	tie	tie	tie	✓
MaxT_I	tie	tie	tie	tie	tie	✗
MaxT_L	tie	tie	tie	tie	tie	✓
MaxT_P	✓	✓	tie	tie	✗	✓
MaxT_U	tie	tie	tie	tie	tie	✓
MaxT^R	✓	✓	✗	✗	✗	✓
MaxT_I^R	✗	✓	✓	✓	✓	✗
MaxT_L^R	✓	✓	✗	✗	✗	✓
MaxT_P^R	✓	✓	✗	✗	✗	✓
MaxT_U^R	✓	✓	✓	✓	✓	✓
SumT	✓	✓	✓	✗	✗	✗
SumT_I	✗	✓	✓	✓	✓	✗
SumT_L	✓	✓	✗	✗	✗	✓
SumT_P	✓	✓	✗	✗	✗	✓
SumT_U	✓	✓	✓	✓	✓	✗
AvgS	✗	✓	✓	✓	✓	✗
Movers	tie	✓	✓	tie	tie	✗
MaxS	tie	✓	✓	tie	tie	✗
MaxS^R	✗	✓	✓	✓	✓	✗
SumS	✗	✓	✓	✓	✓	✗
Con	✗	✓	✓	✗	✗	✗
Con_I	✗	tie	✓	✓	✓	✗
Con_L	✗	✓	✓	✗	✗	✓
Con_P	tie	✓	tie	✗	✗	✓
Con_U	tie	✓	✓	tie	tie	✗
Div	✓	tie	✓	✓	✓	✗
Div_I	✓	tie	tie	tie	tie	✗
Div_L	✓	tie	✓	✗	✗	✗
Div_P	tie	tie	tie	✗	✗	tie
Div_U	tie	tie	tie	tie	tie	✗

Relative clauses as a benchmark for Minimalist parsing

	SC/RC	Eng	SRC < ORC			Right
	< RC/SC		Chi	Kor	Jap	< Center
Box	tie	✓	✓	✗	✗	✗
Box _I	tie	tie	✓	✓	✓	✗
Box _L	tie	✓	tie	✗	✗	✗
Box _P	tie	✓	✗	✗	✗	tie
Box _U	tie	✓	✓	tie	✗	✗
AvgT	✓	✓	✗	✓	✓	✓
AvgT _I	✗	✓	✗	✗	✗	✗
AvgT _L	✓	✓	✗	✓	✓	✓
AvgT _P	✓	✓	✓	✓	✓	✓
AvgT _U	✓	✓	✗	✓	✓	✓
MaxT	tie	tie	tie	tie	tie	✓
MaxT _I	tie	tie	tie	tie	tie	✗
MaxT _L	tie	tie	tie	tie	tie	✓
MaxT _P	✓	✓	tie	tie	tie	✓
MaxT _U	tie	tie	tie	tie	tie	✓
MaxT _I ^R	✓	✓	✗	✗	✗	✓
MaxT _L ^R	✗	✓	✓	✓	✓	✗
MaxT _L ^R	✓	✓	✗	✗	✗	✓
MaxT _P ^R	✓	✓	✗	✗	✗	✓
MaxT _U ^R	✓	✓	✓	✓	✗	✓
SumT	✓	✓	tie	✗	✗	✓
SumT _I	✗	✓	✓	✓	✓	✗
SumT _L	✓	✓	✗	✗	✗	✓
SumT _P	✓	✓	✗	✗	✗	✓
SumT _U	✓	✓	✓	✓	✗	✓
AvgS	✗	tie	✓	✓	✓	✗
Movers	tie	✓	✓	tie	tie	✗
MaxS	tie	✓	✓	tie	tie	✗
MaxS ^R	✗	✓	✓	✓	✓	✗
SumS	✗	✓	✓	✓	✓	✗
Con	✗	✓	✗	✗	✗	✓
Con _I	✗	tie	✗	✓	✓	tie
Con _L	✗	✓	tie	✗	✗	✓
Con _P	tie	tie	✓	✗	✗	✓
Con _U	✗	✓	✗	tie	✗	✓
Div	✓	tie	tie	✗	✗	✗
Div _I	✓	tie	tie	tie	tie	✗
Div _L	✓	tie	tie	✗	✗	✗
Div _P	tie	tie	tie	✗	✗	✗
Div _U	✓	tie	tie	tie	✗	✗

Table 2: Predictions of complexity metrics with wh-movement analysis

Table 3:
List of empirically viable
complexity metrics

	<i>Promotion</i>	<i>Wh-Movement</i>
<i>Basic</i>	MaxT_U^R	AvgT_P
<i>Ranked</i>	⟨MaxT, SumT_U⟩	⟨MaxT_P, AvgS⟩
	⟨MaxT_L, SumT_U⟩	⟨MaxT_P, Box_I⟩
	⟨MaxT_U, SumT_U⟩	⟨MaxT_P, Con_I⟩
		⟨MaxT_P, MaxS^R⟩
		⟨MaxT_P, MaxI_I^R⟩
		⟨MaxT_P, SumS⟩
		⟨MaxT_P, SumT_I⟩

into consideration, the number of metrics increases from 40 to 1,600. The number of empirically adequate metrics, on the other hand, does not increase by the same factor and grows from 1 to 4 (promotion) and 8 (wh-movement), respectively. No metric is a viable candidate for both analyses (see Table 3). Note that these numbers do not include ranked metrics whose first component is an empirically adequate basic metric (**MaxT_U^R** or **AvgT_P**) because the second metric would never be used in those cases. If those pair metrics are included, the respective numbers grow to $4 + 39 = 43$ and $8 + 39 = 47$. Depending on how one counts, then, between $\frac{4}{1600-39} = 0.2\%$ and $\frac{47}{1600} = 2.9\%$ of the full space of complexity metrics can account for the five observed processing contrasts with relative clauses. In addition, all the remaining ranked metrics have some variant of **MaxT** as their first component. This shows that the underspecification problem is not nearly as bad as one might expect, with a few contrasts ruling out the great majority of metrics.

In fact, the five constructions differ in their discriminatory power in a manner that roughly reflects how difficult they are to account for. For example, the preference for SRCs over ORCs in English requires no structure at all and can be explained purely in terms of string distance (Gibson 1998, 2000), and no metric reverses difficulty for this construction. Even the number of ties is comparatively low. The same contrast is much harder to account for in East Asian languages with their pre-nominal RCs. String-based explanations fail in this case, and so do more than half of all the basic metrics. The processing differ-

ence between right embedding and center embedding is interesting in this case because there are a variety of explanations in the psycholinguistic literature, and except for the size- or diversion-based metrics, all the core metrics have some variant that captures the contrast. The failure of size-based metrics is not surprising in this case. Recall that right embedded RCs induce additional syntactic complexity because they start out as center embedded RCs that have to undergo rightward movement. The additional movement steps inevitable cause size-based metrics to make the wrong predictions. Crucially, though, not all metrics fall into this trap, which proves that well-chosen complexity metrics can factor out irrelevant aspects of structural complexity.

4.5 *Qualitative evaluation of complexity metrics*

Since the connection between complexity metrics and the structure of derivation trees is very subtle and sensitive to even minor differences, determining why a complexity metric fails to capture a specific contrast while succeeding at another can be difficult. An exhaustive discussion of all the patterns reported in Table 1–3 is not feasible within the confines of a single paper. Instead, we present a few general observations on the role of **MaxT**, which has been a prominent metric in all previous work on MG parsing and is a component of almost all successful metrics.

First it is instructive, though, to consider why **AvgT_p** works for the wh-analysis but fails for the promotion analysis. The problematic constructions are the East-Asian RCs. Recall that in the promotion analysis, it is the head noun that moves from the gap, whereas in the wh-movement analysis it is the RC marker (simply transcribed as *who* in our derivation trees). Since RCs in East-Asian languages are prenominal and have the RC marker at their very end, the wh-movement analysis has

1. high tenure on the head noun outside the RC (which is encountered before the RC but cannot be finished until the latter is complete),
2. medium tenure on the RC marker in SRCs (as it occupies the structurally prominent subject position, which means that it is hypothesized early by the parser but must wait until the rest of the RC is completed),

3. low tenure on the verb in Korean and Japanese (which is introduced at the same time as the object but only finished after it due to object movement to the left).

In an ORC, the object moves to the right of the RC, so the low tenure on the RC marker disappears, and since it is an ORC the verb does not have any tenure either. But \mathbf{AvgT}_p divides the sum of tenure of pronounced nodes by the number of pronounced nodes with non-trivial tenure. Removing two entries with low tenure ends up increasing the \mathbf{AvgT}_p value for ORCs. The final numbers are $\frac{16+6+3}{3} = 8.3$ for SRCs in contrast to $\frac{16}{1} = 16$ for an ORC.

The structural differences in the promotion analysis, on the other hand, mean that although the switch from SRC to ORC reduces the tenure on the mover (the head noun, rather than the relative pronoun) it does not completely eliminate it. Hence the mover still counts towards the payload and thus greatly lowers the \mathbf{AvgT}_p value for ORCs: $\frac{13+7+3}{2} = 11.5$ in contrast to $\frac{13+3}{2} = 8$. The success of \mathbf{AvgT}_p with the wh-movement thus relies on completely eliminating non-trivial tenure on some nodes in ORCs, rather than just reducing it. The counterintuitive prediction of \mathbf{AvgT} and its variants – if a derivation contains a node with high tenure, it will become easier the more nodes have low tenure instead of no tenure – accidentally makes the right prediction for SRCs and ORCs.

Let us now turn to \mathbf{MaxT} , which strikes us as a more insightful and overall more robust choice of metric. The non-recursive variants of \mathbf{MaxT} are a good choice for ranked metrics because they rarely make a completely wrong prediction but instead produce many ties. This is the reason why all successful ranked metrics contain them as their first component: a complexity metric with a cross in at least one column cannot be the first component of a ranked metric, which rules out all basic metrics except the “tie-heavy” \mathbf{MaxT} variants (and the basic metrics that capture all the data, for which we do not list any ranked metrics).

The high frequency of ties with \mathbf{MaxT} variants is a natural consequence of our focus on embedding constructions. All embedding constructions follow a template where different subtrees are inserted into a fixed main clause. For instance, the English SRC and ORC sentences differ only in the shape of their RCs; the main clause always has the

same structure. The overall number of steps it takes to parse an RC is independent of whether it is an SRC or an ORC. But this, in turn, implies that I) the nodes in the main clause that are introduced before the RC but cannot be worked on until the RC is finished (e.g. T and v' in Figure 8) have very large tenures exceeding that of any node inside the RC, and II) the tenure of these nodes is independent of whether the RC is an SRC or an ORC. As **MaxT** metrics only pay attention to the largest tenure in a tree, the differences between SRCs and ORCs get drowned out by the high tenure on nodes in the main clause.

This accidental flattening of contrasts does not occur in the case of right and center embedded RCs because the movement of an RC in right embedding directly interacts with the nodes in the clause containing the RC. In particular, moving an RC to the right of an LI l means that l can be worked on before the RC is explored by the parser, thus reducing its tenure. With center embedding, the parser would first have to explore the full RC, so the sister node of the RC would wind up with very high tenure. The overall generalization, then, is that **MaxT** metrics flatten contrasts where the differences between constructions are restricted to nodes within the embedded subtree.

MaxT_p is an exception because its restriction to pronounced nodes filters out the tenure of interior nodes like v' and unpronounced lexical heads like T. This improves its performance for the SC/RC versus RC/SC contrast as well as English SRC and ORC constructions. If our analysis had treated T as a pronounced head (e.g. for *do* support, or as the carrier of inflection that affix hops onto the verb), **MaxT_p** would also produce ties in these cases. But even in this case the behavior of **MaxT_p** could still be replicated by a metric that ignores interior nodes and functional heads, irrespective of whether they are pronounced.

While **MaxT_p** improves on other variants in some respects, it is also the only non-recursive **MaxT** version to incorrectly derive an ORC advantage in Japanese with the promotion analysis. This is due to the RC marker being unpronounced in Japanese, so that the only pronounced nodes with tenure are the head noun and the embedded verb. The head noun has the same tenure for SRC and ORC, but the embedded verb has non-trivial tenure in the SRC as it is introduced at the same time as the object but must wait for it to move leftward to Spec, v P. In the ORC, on the other hand, the object moves to a position to the right of the embedded verb, so that the latter can be completed

as soon as it is introduced. The ORC advantage thus is due to object extraction negating the inherent disadvantage of object movement.

In sum, it seems that any variant of **MaxT** that does not restrict itself to just pronounced nodes provides a solid baseline for a ranked metric with a suitably chosen ancillary metric to resolve ties. **MaxT** has previously been studied by Kobele *et al.* (2013), Graf and Marcinek (2014) and Gerth (2015), and it can even be traced back to Kaplan (1974) and Wanner and Maratsos (1978). It also plays a role in the TAG processing models of Joshi (1990) – in fact, Joshi (1990) ignores the memory usage of empty nodes and thus uses what amounts to our **MaxT_p**, which is part of the majority of viable metrics. That three very different processing models home in on the same kind of memory usage as a benchmark for processing difficulty is very suggestive.

From the perspective of Minimalist syntax, $\langle \mathbf{MaxT}, \mathbf{SumS} \rangle$ and $\langle \mathbf{MaxT}, \mathbf{MaxS}^R \rangle$ are arguably the most elegant metric as they, intuitively speaking, combine maximum memory load with the total resource demand of all movement dependencies. In the generative literature, O’Grady (2011) has independently argued for the impact of movement dependencies on sentence processing, supporting a size-based metric. Our study confirms that these conceptually pleasing metrics have a lot of explanatory power to offer, although there are still some viable alternatives.

5 FURTHER OBSERVATIONS AND DISCUSSION

While the present study considers a much wider range of constructions and metrics than previous work on MG processing, it is still more limited in its scope than is desirable. The set of syntactic analyses, processing phenomena, and MG parsing algorithms all need to be extended to get a fuller picture of the empirical feasibility of this approach.

Our syntactic analyses still fix a plethora of parameters that need to be carefully modulated. For example, the low starting position of subjects and the movement of objects to Spec,vP cause tenure on T and v, respectively, which affects certain metrics. Replacing rightward movement by sequences of leftward movement (also known as remnant movement) will also be picked up on by some metrics, as would the introduction of a general headedness parameter to do away with certain movement steps. Since the derivation trees using these alter-

native proposals need to be carefully constructed by hand, a piecewise comparison that alters only one parameter at a time is a very laborious process.

A reviewer raises similar concerns and asks how useful these results are considering that the structure of East Asian languages is not nearly as well understood as that of English, wherefore their Minimalist analyses are much more likely to be fatally flawed. We agree that if push comes to shove, a metric's failure to account for the East Asian processing patterns has less weight than its performance on English data. However, the English data that is available and easily tested in this framework lacks some discriminatory power that the East Asian RC data provides. In science, we have to work with the data that is available, even if that data is sometimes sub-optimal.

But suppose that the structure of East Asian RCs does indeed need to be reanalyzed. We do not believe that this would lead to completely different metrics being chosen. We did some tests with an analysis of Korean and Japanese that simply linearizes the object to the left of the verb rather than moving it to Spec, ν P. This made the processing predictions for them more similar to Chinese, and as a result widened the set of feasible metrics to also include ranked metrics whose first component is **SumT** for the wh-movement analysis or a variant of **Box** for the promotion analysis. Crucially, though, all the previously successful metrics were still available.

In the other direction, we also experimented with adding the preference for crossing dependencies over nested dependencies (Bach *et al.* 1986) to our data set. This preference was already shown in Kobele *et al.* (2013) to be predicted by **MaxT**. So it comes as little surprise that this contrast has no discriminative power relative to our current data set. All of our successful metrics correctly predict the contrast. Preliminary work on attachment preferences for dative arguments in Korean and quantifier scope preferences in English suggest that these, too, can be accounted for with the metrics identified in this paper. Overall, then, it seems that the class of complexity metrics carved out in this paper is fairly robust and more than just an accident.

CONCLUSION

We defined a large set of reasonably simple complexity metrics that make predictions about processing behavior based on the shape of index/outdex annotated MG derivation trees that closely mimic well-known analyses from Minimalist syntax. Only a few metrics could cover the full range of relative clause constructions, suggesting that the choice of metric is much more restricted than one might initially expect, and that underspecification is not too much of an issue in practice. In addition, the fact that it was at all possible to give a unified explanation of relative clause processing effects, which have proven challenging to deal with in the psycholinguistic literature, is encouraging. The MG processing model we advocate deliberately abstracts away from many aspects of sentence processing in order to clearly bring out the role that might be played by syntactic factors. It seems that at least in the case of relative clauses, structural considerations go a long way.

ACKNOWLEDGMENTS

We are grateful to the three anonymous reviewers, whose feedback has led to several changes in the presentation of the material (for the better, we like to believe). This research project would not exist if it were not for the continued encouragement by John Drury and the unique atmosphere created by the participants of the Fall 2014 parsing seminar at Stony Brook University.

REFERENCES

- Emmon BACH, Colin BROWN, and William MARSLEN-WILSON (1986), Crossed and Nested Dependencies in German and Dutch: A Psycholinguistic Study, *Language and Cognitive Processes*, 1:249–262.
- Klinton BICKNELL, Roger LEVY, and Vera DEMBERG (2009), Correcting the Incorrect: Local Coherence Effects Modeled with Prior Belief Update, in *Proceedings of the 35th Annual Meeting of the Berkeley Linguistics Society*, pp. 13–24, doi:10.3765/bls.v35i1.3594.
- Noam CHOMSKY (1965), *Aspects of the Theory of Syntax*, MIT Press, Cambridge, MA.

- Noam CHOMSKY (1977), *Essays on Form and Interpretation*, New York, North Holland.
- Noam CHOMSKY (1995a), Bare Phrase Structure, in Gert WEBELHUTH, editor, *Government and Binding Theory and the Minimalist Program*, pp. 383–440, Blackwell, Oxford.
- Noam CHOMSKY (1995b), *The Minimalist Program*, MIT Press, Cambridge, MA, doi:10.7551/mitpress/9780262527347.003.0003.
- Noam CHOMSKY (2001), Derivation by Phase, in Michael J. KENSTOWICZ, editor, *Ken Hale: A Life in Language*, pp. 1–52, MIT Press, Cambridge, MA.
- Philippe DE GROOTE (2001), Towards Abstract Categorical Grammars, in *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter*, pp. 148–155.
- Meaghan FOWLIE (2013), Order and Optionality: Minimalist Grammars with Adjunction, in András KORNAI and Marco KUHLMANN, editors, *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pp. 12–20.
- Lyn FRAZIER (1987), Sentence Processing: A Tutorial Review, in M. COLTHEART, editor, *Attention and Performance XII: The Psychology of Reading*, pp. 559–586, Lawrence Erlbaum Associates, Inc.
- Werner FREY and Hans-Martin GÄRTNER (2002), On the Treatment of Scrambling and Adjunction in Minimalist grammars, in Gerhard JÄGER, Paola MONACHESI, Gerald PENN, and Shuly WINTNER, editors, *Proceedings of the Conference on Formal Grammar*, pp. 41–52.
- Hans-Martin GÄRTNER and Jens MICHAELIS (2007), Some Remarks on Locality Conditions and Minimalist Grammars, in Uli SAUERLAND and Hans-Martin GÄRTNER, editors, *Interfaces + Recursion = Language? Chomsky's Minimalism and the View from Syntax-Semantics*, pp. 161–196, Mouton de Gruyter, Berlin.
- Hans-Martin GÄRTNER and Jens MICHAELIS (2010), On the Treatment of Multiple-Wh-Interrogatives in Minimalist Grammars, in Thomas HANNEFORTH and Gisbert FANSELOW, editors, *Language and Logos*, pp. 339–366, Akademie Verlag, Berlin.
- Sabrina GERTH (2015), *Memory Limitations in Sentence Comprehension. A Structure-Based Complexity Metric of Processing Difficulty*, Ph.D. thesis, University of Potsdam.
- Edward GIBSON (1998), Linguistic Complexity: Locality of Syntactic Dependencies, *Cognition*, 68:1–76.
- Edward GIBSON (2000), The Dependency Locality Theory: A Distance-Based Theory of Linguistic Complexity, in Y. MIYASHITA, Alec MARANTZ, and W. O'NEIL, editors, *Image, Language, Brain*, pp. 95–126, MIT Press, Cambridge, MA.
- Edward GIBSON and H.-H. Iris WU (2013), Processing Chinese Relative Clauses in Context, *Language and Cognitive Processes*, 28(1-2):125–155.

Peter C. GORDON, Randall HENDRICK, Marcus JOHNSON, and Yoonhyoung LEE (2006), Similarity-Based Interference During Language Comprehension: Evidence From Eye Tracking During Reading, *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 32(6):1304.

Thomas GRAF (2011), Closure Properties of Minimalist Derivation Tree Languages, in Sylvain POGODALLA and Jean-Philippe PROST, editors, *LACL 2011*, volume 6736 of *Lecture Notes in Artificial Intelligence*, pp. 96–111, Springer, Heidelberg, doi:10.1007/978-3-642-22221-4_7.

Thomas GRAF (2012a), Locality and the Complexity of Minimalist Derivation Tree Languages, in Philippe DE GROOT and Mark-Jan NEDERHOF, editors, *Formal Grammar 2010/2011*, volume 7395 of *Lecture Notes in Computer Science*, pp. 208–227, Springer, Heidelberg, doi:10.1007/978-3-642-32024-8_14.

Thomas GRAF (2012b), Movement-Generalized Minimalist Grammars, in Denis BÉCHET and Alexander J. DIKOVSKY, editors, *LACL 2012*, volume 7351 of *Lecture Notes in Computer Science*, pp. 58–73, doi:10.1007/978-3-642-31262-5_4.

Thomas GRAF (2013), *Local and Transderivational Constraints in Syntax and Semantics*, Ph.D. thesis, UCLA.

Thomas GRAF, Alëna AKSËNOVA, and Aniello DE SANTO (2016), A Single Movement Normal Form for Minimalist Grammars, in Annie FORET, Glyn MORRILL, Reinhard MUSKENS, Rainer OSSWALD, and Sylvain POGODALLA, editors, *Formal Grammar: 20th and 21st International Conferences*, pp. 200–215, doi:10.1007/978-3-662-53042-9_12.

Thomas GRAF, Brigitta FODOR, James MONETTE, Gianpaul RACHIELE, Aunika WARREN, and Chong ZHANG (2015), A Refined Notion of Memory Usage for Minimalist Parsing, in *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pp. 1–14, Association for Computational Linguistics, Chicago, USA.

Thomas GRAF and Bradley MARCINEK (2014), Evaluating Evaluation Metrics for Minimalist Parsing, in *Proceedings of the 2014 ACL Workshop on Cognitive Modeling and Computational Linguistics*, pp. 28–36.

John HALE (2001), A Probabilistic Earley Parser as a Psycholinguistic Model, in *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, pp. 1–8.

John HALE (2011), What a Rational Parser Would Do, *Cognitive Science*, 35:399–443.

John T. HALE (2003), *Grammar, Uncertainty and Sentence Processing*, Ph.D. thesis, John Hopkins University.

Henk HARKEMA (2001), A Characterization of Minimalist Languages, in Philippe DE GROOTE, Glyn MORRILL, and Christian RETORÉ, editors, *Logical*

Aspects of Computational Linguistics (LACL'01), volume 2099 of *Lecture Notes in Artificial Intelligence*, pp. 193–211, Springer, Berlin.

Irene HEIM and Angelika KRATZER (1998), *Semantics in Generative Grammar*, Blackwell, Oxford.

Tim HUNTER (2011), Insertion Minimalist Grammars: Eliminating Redundancies between Merge and Move, in Makoto KANAZAWA, András KORNAI, Marcus KRACHT, and Hiroyuki SEKI, editors, *The Mathematics of Language: 12th Biennial Conference*, pp. 90–107, Springer, Berlin, Heidelberg, ISBN 978-3-642-23211-4, doi:10.1007/978-3-642-23211-4_6.

Tim HUNTER (2015a), Deconstructing Merge and Move to Make Room for Adjunction, *Syntax*, 18:266–319, doi:10.1111/synt.12033.

Tim HUNTER (2015b), Left-Corner Parsing of Minimalist Grammars, slides of a talk presented at the First Workshop on Minimalist Parsing, October 10–11 2015, MIT.

Tim HUNTER and Robert FRANK (2014), Eliminating Rightward Movement: Extraposition as Flexible Linearization of Adjuncts, *Linguistic Inquiry*, 45:227–267.

Aravind JOSHI (1990), Processing Crossed and Nested Dependencies: An Automaton Perspective on the Psycholinguistic Results, *Language and Cognitive Processes*, 5:1–27.

Ronald M. KAPLAN (1974), *Transient Processing Load in Relative Clauses*, Ph.D. thesis, Harvard University.

Richard S. KAYNE (1994), *The Antisymmetry of Syntax*, MIT Press, Cambridge, MA.

Edward L. KEENAN and Bernard COMRIE (1977), Noun Phrase Accessibility and Universal Grammar, *Linguistic Inquiry*, 8:63–99.

John KIMBALL (1973), Seven Principles of Surface Structure Parsing in Natural Language, *Cognition*, 2:15–47.

Gregory M. KOBELE (2006), *Generating Copies: An Investigation into Structural Identity in Language and Grammar*, Ph.D. thesis, UCLA.

Gregory M. KOBELE (2011), Minimalist Tree Languages are Closed Under Intersection with Recognizable Tree Languages, in Sylvain POGODALLA and Jean-Philippe PROST, editors, *LACL 2011*, volume 6736 of *Lecture Notes in Artificial Intelligence*, pp. 129–144, doi:10.1007/978-3-642-22221-4_9.

Gregory M. KOBELE (2015), LF-Copying Without LF, *Lingua*, 166:236–259, doi:10.1016/j.lingua.2014.08.006.

Gregory M. KOBELE, Sabrina GERTH, and John T. HALE (2013), Memory Resource Allocation in Top-Down Minimalist Parsing, in Glyn MORRILL and Mark-Jan NEDERHOF, editors, *Formal Grammar: 17th and 18th International Conferences*, pp. 32–51, doi:10.1007/978-3-642-39998-5_3.

- Gregory M. KOBELE, Christian RETORÉ, and Sylvain SALVATI (2007), An Automata-Theoretic Approach to Minimalism, in James ROGERS and Stephan KEPSEK, editors, *Model Theoretic Syntax at 10*, pp. 71–80.
- Alexander KOLLER and Marco KUHLMANN (2011), A Generalized View on Parsing and Translation, in *Proceedings of the 12th International Conference on Parsing Technologies*, pp. 2–13, Association for Computational Linguistics, Stroudsburg, PA.
- Lars KONIECZNY (2005), The Psychological Reality of Local Coherences in Sentence Processing, in *Proceedings of the 27th Annual Conference of the Cognitive Science Society*.
- Lars KONIECZNY, Daniel MÜLLER, Wibke HACHMANN, Sarah SCHWARZKOPF, and Sascha WOLFER (2009), Local Syntactic Coherence Interpretation. Evidence from a Visual World Study, in *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, pp. 1133–1138.
- Nayoung KWON, Peter C. GORDON, Yoonhyoung LEE, Robert KLUENDER, and Maria POLINSKY (2010), Cognitive and Linguistic Factors Affecting Subject/Object Asymmetry: An Eye-Tracking Study of Prenominal Relative Clauses in Korean, *Language*, 86(3):546–582.
- Richard LARSON (1988), On the Double Object Construction, *Linguistic Inquiry*, 19(3):335–391.
- David LEBEAUX (1988), *Language Acquisition and the Form of the Grammar*, Ph.D. thesis, University of Massachusetts, Amherst, doi:10.1075/z.97, reprinted in 2000 by John Benjamins.
- Roger LEVY (2013), Memory and Surprisal in Human Sentence Comprehension, in Roger P. G. VAN GOMPEL, editor, *Sentence Processing*, pp. 78–114, Psychology Press, Hove.
- Chien-Jer Charles LIN and Thomas G. BEVER (2006), Subject Preference in the Processing of Relative Clauses in Chinese, in *Proceedings of the 25th West Coast Conference on Formal Linguistics*, pp. 254–260, Cascadilla Proceedings Project Somerville, MA.
- Thomas MAINGUY (2010), A Probabilistic Top-Down Parser for Minimalist Grammars, arXiv:1010.1826v1.
- Willem M. MAK, Wietske VONK, and Herbert SCHRIEFERS (2002), The Influence of Animacy on Relative Clause Processing, *Journal of Memory and Language*, 47(1):50–68.
- Willem M. MAK, Wietske VONK, and Herbert SCHRIEFERS (2006), Animacy in Processing Relative Clauses: The Hikers That Rocks Crush, *Journal of Memory and Language*, 54(4):466–490.
- Axel MECKLINGER, Herbert SCHRIEFERS, Karsten STEINHAEUER, and Angela D. FRIEDERICI (1995), Processing Relative Clauses Varying on Syntactic and

- Semantic Dimensions: An Analysis with Event-Related Potentials, *Memory & Cognition*, 23(4):477–494.
- Jens MICHAELIS (2001), Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars, *Lecture Notes in Artificial Intelligence*, 2099:228–244.
- George A. MILLER and Noam CHOMSKY (1963), Finitary Models of Language Users, in R. LUCE, R. BUSH, and E. GALANTER, editors, *Handbook of Mathematical Psychology*, volume 2, pp. 419–491, John Wiley, New York.
- George A. MILLER and Kathryn Ojemann MCKEAN (1964), A Chronometric Study of Some Relations Between Sentences, *Quarterly Journal of Experimental Psychology*, 16:297–308.
- Edson T. MIYAMOTO and Michiko NAKAMURA (2003), Subject/Object Asymmetries in the Processing of Relative Clauses in Japanese, in *Proceedings of WCCFL*, volume 22, pp. 342–355.
- Edson T. MIYAMOTO and Michiko NAKAMURA (2013), Unmet Expectations in the Comprehension of Relative Clauses in Japanese, in *Proceedings of the 35th Annual Meeting of the Cognitive Science Society*, pp. 3074–3079.
- Uwe MÖNNICH (2006), Grammar Morphisms, ms. University of Tübingen.
- Richard MONTAGUE (1970), English as a Formal Language, in Bruno VISENTINI and ET AL., editors, *Linguaggi nella Società e nella Tecnica*, pp. 189–224, Edizioni di Comunità, Milan.
- Frank MORAWIETZ (2003), *Two-Step Approaches to Natural Language Formalisms*, Walter de Gruyter, Berlin, doi:10.1515/9783110197259.
- William O’GRADY (2011), Relative Clauses. Processing and Acquisition, in Evan KIDD, editor, *Processing, Typology, and Function*, pp. 13–38, John Benjamins, Amsterdam.
- Colin PHILLIPS (1996), *Order and Structure*, Ph.D. thesis, MIT.
- Alan PRINCE and Paul SMOLENSKY (2004), *Optimality Theory: Constraint Interaction in Generative Grammar*, Blackwell, Oxford.
- Owen RAMBOW and Aravind JOSHI (1995), A Processing Model for Free Word Order Languages, Technical Report IRCS-95-13, University of Pennsylvania.
- Philip RESNIK (1992), Left-Corner Parsing and Psychological Plausibility, in *Proceedings of COLING-92*, pp. 191–197.
- Sylvain SALVATI (2011), Minimalist Grammars in the Light of Logic, in Sylvain POGODALLA, Myriam QUATRINI, and Christian RETORÉ, editors, *Logic and Grammar — Essays Dedicated to Alain Lecomte on the Occasion of His 60th Birthday*, number 6700 in Lecture Notes in Computer Science, pp. 81–117, Springer, Berlin, doi:10.1007/978-3-642-21490-5_5.
- Stuart M. SHIEBER, Yves SCHABES, and Fernando C. PEREIRA (1995), Principles and Implementations of Deductive Parsing, *Journal of Logic Programming*, 24:3–36.

- Klaas SIKKEL (1997), *Parsing Schemata*, Texts in Theoretical Computer Science, Springer, Berlin.
- Edward P. STABLER (1997), Derivational Minimalism, in Christian RETORÉ, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pp. 68–95, Springer, Berlin, doi:10.1007/BFb0052152.
- Edward P. STABLER (2006), Sideways Without Copying, in Gerald PENN, Giorgio SATTA, and Shuly WINTNER, editors, *Formal Grammar '06, Proceedings of the Conference*, pp. 133–146, CSLI, Stanford.
- Edward P. STABLER (2011), Computational Perspectives on Minimalism, in Cedric BOECKX, editor, *Oxford Handbook of Linguistic Minimalism*, pp. 617–643, Oxford University Press, Oxford, doi:10.1093/oxfordhb/9780199549368.013.0027.
- Edward P. STABLER (2013), Two Models of Minimalist, Incremental Syntactic Analysis, *Topics in Cognitive Science*, 5:611–633, doi:10.1111/tops.12031.
- Mark STEEDMAN (2001), *The Syntactic Process*, MIT Press, Cambridge, MA.
- Whitney TABOR, Bruno GALANTUCCI, and Daniel RICHARDSON (2004), Effects of Merely Local Syntactic Coherence on Sentence Processing, *Journal of Memory and Language*, 50:355–370.
- Shoichi TAKAHASHI and Sarah HULSEY (2009), Wholesale Late Merger: Beyond the A/ \bar{A} Distinction, *Linguistic Inquiry*, 40:387–426.
- James W. THATCHER (1967), Characterizing Derivation Trees for Context-Free Grammars Through a Generalization of Finite Automata Theory, *Journal of Computer and System Sciences*, 1:317–322.
- Mieko UENO and Susan M GARNSEY (2008), An ERP Study of the Processing of Subject and Object Relative Clauses in Japanese, *Language and Cognitive Processes*, 23(5):646–688.
- Jean-Roger VERGNAUD (1974), *French Relative Clauses*, Ph.D. thesis, MIT.
- Eric WANNER and Michael MARATSOS (1978), An ATN Approach to Comprehension, in Morris HALLE, Joan BRESNAN, and George A. MILLER, editors, *Linguistic Theory and Psychological Reality*, pp. 119–161, MIT Press, Cambridge, MA.
- Jiwon YUN, Zhong CHEN, Tim HUNTER, John WHITMAN, and John HALE (2014), Uncertainty in Processing Relative Clauses Across East Asian Languages, *Journal of East Asian Linguistics*, pp. 1–36.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.

<http://creativecommons.org/licenses/by/3.0/>



Rewrite rule grammars with multitape automata

Mans Hulden

Department of Linguistics
University of Colorado, USA

ABSTRACT

The majority of computational implementations of phonological and morphophonological alternations rely on composing together individual finite state transducers that represent sound changes. Standard composition algorithms do not maintain the intermediate representations between the ultimate input and output forms. These intermediate strings, however, can be very helpful for various tasks: enriching information (indispensable for models of historical linguistics), providing new avenues to debugging complex grammars, and offering explicit alignment information between morphemes, sound segments, and tags. This paper describes a multitape automaton approach to creating full models of sequences of sound alternation that implement phonological and morphological grammars. A model and a practical implementation of multitape automata is provided together with a multitape composition algorithm tailored to the representation used in this paper. Practical use cases of the approach are illustrated through two common examples: a phonological example of a complex rewrite rule grammar where multiple rules interact and a diachronic example of modeling sound change over time.

Keywords:
grammar design,
multitape
automata,
morphology,
phonology,
finite-state
phonology

1

INTRODUCTION

Finite-state transducer based phonological and morphological models tend to be built by the composition of individual transducers that encode morphotactics and morphophonological alternations (Beesley

and Karttunen 2003). Apart from cases such as nonconcatenative morphologies where augmented techniques tend to be favored (Beesley and Karttunen 2000; Habash *et al.* 2005; Hulden 2009d; Kiraz 2001), this well-established approach is indeed quite successful and streamlined in the domain of morphophonology if the goal is to produce a single transducer that maps underlying forms (parses) to surface forms and vice versa.

Some types of grammatical information are difficult to include in such a design, however. In morphological modeling, one may want to recover the alignment of morphological tags to the actual morphemes; in phonological modeling, one may want to recover intermediate representations that show how a particular phonological alternation targets specific segments in a word, what order phonological alternations occur in, and what they were conditioned on. This is particularly important in developing finite-state models of historical sound change, where it is imperative to retain intermediate alignment information so that the model may indicate what sound laws proto-segments are subject to and in what order changes occur. In some respect, the “intermediate representations” in diachronic derivations are more crucial to the linguist than their counterparts in synchronic models since in the latter case they are bound to a particular model of phonology. The ability to model such sequences would make finite-state devices more attractive for linguistic research, where computational methods could help streamline the work of lining up large amounts of data and testing hypothetical generalizations; it might therefore increase linguists’ use of finite-state methods, whose potential has to date been underexploited in the linguistics literature (Karttunen 2003).

In this paper, I show that a multitape model constructed by composition of individual multitape lexicon or alternation transducers offers a simple framework that addresses the problem of intermediate forms, while at the same time retaining the straightforward design of morphology and morphophonology. Apart from expanding the expressive power of the grammar, the method also offers the grammar designer the option to re-convert the multitape grammar to a simple underlying-to-surface transducer, if desired – as may be the case if the multitape representation is only used for obtaining debugging information. Indeed, debugging the alternation rules and lexicon description involved in drafting a morphological grammar becomes much

less burdensome under the multitape model, since information about each step in the process of mapping from underlying to surface form is retained and is available for inspection.¹

The methods described in this paper are implemented as a stand-alone library in Python. The library itself is built on top of the *foma* library (Hulden 2009b) which provides a backbone implementation of standard transducer algorithms. The implementation allows users to develop multitape grammars in standard regular expression and rewrite-rule notation, automatically and transparently converting the compiled transducers to multitape equivalents and performing multitape composition on the components. This enables a relatively linguist-friendly grammar design procedure that relies on well-known formalisms and offers the possibility of quick conversion of existing grammars into a multitape representation where word-forms can be parsed and generated with a rich intermediate structure.

This paper is structured as follows: first, some background on rewrite-rule grammars is presented, motivating the need for more richly structured representations; this is followed by a description of the multitape encoding with special focus on the composition algorithm for multitape automata; following this, a system for augmenting the multitape automata with extra annotation (such as rule names) is presented; two case studies are then provided to illustrate in concrete terms the possibilities of the multitape formalism.

2 TRADITIONAL REWRITE-RULE GRAMMARS

A significant portion of morphological analysis tools are written with the design described above: (1) a transducer that encodes morphotactics and tag sequences, and (2) a series of transducers that model morphophonological/orthographic alternation. The latter may be expressed as Sound Pattern of English-inspired ‘rewrite rules’ (Chomsky and Halle 1968) or as two-level parallel constraints (Koskenniemi 1983), the former being arguably the more popular choice at present due to simplicity of debugging complex rule interactions (Alegria *et al.* 2010). The result of composing the lexicon transducer and the mor-

¹The code and the examples in this article are available at <https://fomafst.github.io/>.

Table 1: Interaction of multiple phonological processes in Lardil

tupalan-uɿ	papi-uɿ	pulpu-un	pulpu	kiɿkiɿi	muŋkumuŋku	Underlying form
tupalankuɿ	papiwuɿ	pulpun	pulpa	kiɿkiɿæ kiɿkiɿ	muŋkumuŋka muŋkumuŋk muŋkumuŋ muŋkumu	/k/-epenthesis /w/-epenthesis Vowel Deletion Final Lowering Apocope Cluster Reduction Non-apical Truncation Sonorantization
tupalankuɿ	papiwuɿ	pulpun	pulpa	kiɿkiɿ	muŋkumu	Surface form

phophonological transducers is one monolithic transducer that directly performs the bidirectional mapping from underlying-to-surface forms (generation) and vice versa (parsing). The prevalence of this design is probably partly due to known algorithms (Kaplan and Kay 1994; Kempe and Karttunen 1996; Mohri and Sproat 1996; Hulden 2009c) or software tools designed around this paradigm (such as *lexc/xfst/twol* by Xerox (Beesley and Karttunen 2003), *foma* (Hulden 2009b), or *Kleene* (Beesley 2012)). In the following, I shall assume the more common ‘rewrite-rule’ paradigm.

Table 1 illustrates this standard design using some example words from a grammar of Lardil (iso 639-3: lbz, a Pama-Nyungan language spoken on Mornington Island in Australia). This is an example language often used to illustrate complex rule ordering and word-final phonology with rules that are sensitive to ordering. The table is laid out in a manner often employed by phonologists to quickly give an overview of interacting processes. The original data stems from Hale (1973), and I follow analyses by Kenstowicz and Kisseberth (1979); Hayes (2011); Round (2011). Due to the rich interaction of word-final deletion rules, this is a commonly cited data set that has been a target of many analyses, all of which illustrate the difficulty of marshaling a complex set of phonological alternations. In the language, we find three independently motivated deletion rules (apocope, cluster reduction, non-apical truncation) which interact in complex ways, sometimes conspiring to elide multiple segments word-finally. The rules in question are shown here in traditional phonological notation:

apocope	$V \rightarrow \emptyset / V C_0 V C_0 _ \#$
cluster reduction	$C \rightarrow \emptyset / C _ \#$
non-apical truncation	$C \rightarrow \emptyset / _ \#$ (unless $C = [-\text{distr.}]$)

To explain the workings of the grammar, the table shows all the intermediate steps in mapping from lemma-and-inflection forms to actual surface realizations. In actuality, however, if modeled by transducer composition, all the intermediate forms are lost through the composition process, which is one of the shortcomings addressed below. That is, a final composite transducer simply provides mappings between parse and surface. For phonological analysis, possible grammar debugging, and perhaps language documentation purposes, it would be very desirable to be able to produce a rich representation such as any of the columns shown in Table 1 from either an underlying form (morphological information) or the surface form showing all the processes that the word undergoes step-by-step.

Under the standard composition model, there is no easy way to do this, save by applying an underlying form to each of the individual transducers representing the alternation rules in order, saving the results, and passing them on as input to the next transducer. However, in the inverse direction, such a strategy is not directly feasible, in addition to the fact that not composing the transducers partly defeats the purpose of using a finite-state model in the first place.

There is no principled reason, however, why the composition algorithm should destroy the intermediate representations. In other words, when creating a composite transducer modeling $x:z$ from transducers $x:y$ and $y:z$, one can in principle expand the composition algorithm to yield $x:y:z$ in some representation, retaining all the intermediate information. As will be seen below, a combination of a multitape design together with a rule-decoration mechanism allows us to automatically produce rich analyses very much like the ones given in Table 1.

Multitape automata in general have been proposed as viable models for morphology and phonology, particular when addressing nonconcatenative phenomena abundant in Semitic languages such as Arabic, Hebrew, and Syriac (Altantawy *et al.* 2010; Kay 1987; Habash *et al.*

2005; Habash and Rambow 2006; Hulden 2009d; Kiraz 2000, 2001). In these approaches, different phonological tiers are represented by different tapes in a multitape model. Most of these earlier models could in fact be called multitape transducer models, since they typically work akin to transducers, although with an extended symbol representation where instead of manipulating symbol pairs, as in the transducer case, transitions are labeled with n -tuples of symbols. Specialized algorithms are then used to handle this representation and to enforce symbol correspondences across tapes – Kiraz (2000), for example, works with a constraint formalism similar to that of two-level morphology (Koskenniemi 1983), extended to operate in a multitape transducer scenario.

By contrast, the current work assumes as a starting point that regularities across multiple levels of representation will be captured not by constraints across multiple tapes, but that adjacent tapes will be constrained by (morpho)phonological *rewrite rules*. To make this feasible, the compilation of rewrite rules must be extended to a multitape scenario, and a composition algorithm is required that is able to join multitape representations together, preserving intermediate information.

The importance of the preservation of intermediate results in composition has been noted and partly addressed in Kempe *et al.* (2004), among others. The formulation presented below differs from earlier work in both representation and algorithms, and also in that it is intended to be simple and easily implementable without special algorithms for multitape automata, i.e. using only established algorithms for single-tape automata and transducers. The same representation (without a composition design) has been used earlier for the construction of Arabic multitape grammars (Hulden 2009a). In that work, conversion from transducers is not considered, and no composition algorithm is given, as the assumption is that multitape automata are constructed through intersections of constraints on co-occurrence of symbols on the various tapes, analogously to two-level grammars (Koskenniemi 1983). The multitape representation in this paper uses the encoding from (Hulden 2009d) and builds upon extensions to it given in Hulden (2015).

In discussing algorithmic aspects, familiarity with standard regular expression notation to construct automata and transducers is assumed. For regular languages or automata X and Y , the description below will make use of the operations union ($X \cup Y$), concatenation (XY), Kleene closure (X^*), Kleene plus (X^+), intersection ($X \cap Y$), complement ($\neg X$), and difference ($X - Y$). The n -ary concatenation of a language X with itself is denoted X^n . From two languages represented as automata, their string-wise cross-product and resulting regular relation (representable as a transducer) is denoted with $X : Y$. If X and Y are transducers, their composition is ($X \circ Y$). The input and output projections of a relation/transducer X are denoted $\text{domain}(X)$ and $\text{range}(X)$. Whenever a regular language (or automaton) X appears in a transducer context, it is assumed to represent the identity relation, i.e. a transducer that simply repeats the set of words accepted by X . In some algorithms subtraction is performed in a transducer context ($X - Y$); in such cases the subtraction refers to transducer path subtraction and not relation subtraction which regular relations are not closed under, i.e. the result represents the set of valid sequences of symbol pairs in X but not in Y . We use the special symbol $?$ to represent any single symbol.

When describing linguistic grammars, the well-known Xerox regular expression notation (Beesley and Karttunen 2003) is used in this paper to define and manipulate automata and transducers, rewrite rule transducers in particular; the examples should be directly compilable with the *foma* library. The formalism used is summarized in Table 2. Multitape additions are implemented through a Python interface discussed in Section 9.

In the implementations below, a multitape representation is assumed to be a simple single-tape automaton that either accepts or rejects a string s in the standard way. However, the strings in question are intended to represent valid computations of a multitape automaton where certain positions in s pertain to certain tapes. Which symbol in the linear string s belongs to which tape is modeled by a simple “interleaving” encoding where the length of any accepted string s is

Table 2:
Regular expression notation in
xfst/foma

AB	Concatenation
A B	Union
A*	Kleene Star
~A	Complement
?	Any symbol in alphabet
∅	The empty string (epsilon)
A [^] k	k-ary concatenation
%	Escape symbol
[and]	Grouping brackets
A:B	Cross product
A/B	A ignoring intervening B
T.2	Output projection of T
A -> B	Rewrite A as B
C _ D	Context specifier
.#.	End or beginning of string
def F(X1, . . . , Xn)	definition of macro
def X	definition of language constant

always an even multiple of the number of tapes in the multitape model it is intended to represent. Informally, the string first encodes the first column of the legal contents of an n -tape multitape automaton, top-down, then the second column, etc. etc. Every symbol in position k in the linear string representation corresponds to – in the case of n tapes – position $\lfloor k/n \rfloor$ on tape $(k \bmod n)$. A special representation for empty symbols (ϵ -symbols) in the single-tape model is assumed whereby they are represented with the symbol \square – a so-called “hard zero”. A string of length $l \times n$ in the single-tape string would correspond to the multitape representation as follows, where, in parentheses, the position within a tape is shown first, followed by the tape number in the multitape representation.

T_0	(0,0)	(1,0)	...	($l,0$)
...				
T_{n-1}	(0, $n-1$)	(1, $n-1$)	...	($l, n-1$)
T_n	(0, n)	(1, n)	...	(l, n)

For example, if a single-tape representation contains in its language the string $abcde\square$, this is assumed to correspond to a valid

configuration

$$\begin{bmatrix} a & d \\ b & e \\ c & \square \end{bmatrix}$$

seen from the multitape point-of-view (a 3-tape configuration); i.e. a multitape automaton that accepts the string ad as input, translates it into be , and then translates this into c (the \square -symbol representing the empty string).

6 CONVERSION FROM TRANSDUCERS

It is evident that an existing transducer can be converted to this multitape representation – that is, to a 2-tape representation – without much effort. To convert a standard transducer, where transitions are encoded as symbol pairs, one simply expands each transition with a symbol pair $x : y$ to a two-symbol sequence xy in the corresponding n -tape automaton. This operation will be referred to as “flattening.”² If the original transducer T maps a string $x_1 \dots x_n$ to $y_1 \dots y_n$ by a sequence of transitions with labels $((x_1, y_1), \dots, (x_n, y_n))$, then the automaton $\text{flatten}(T)$ accepts a string $(x_1 y_1 \dots x_n y_n)$. In the result, ϵ -symbols are replaced with the \square -symbol. This \square -symbol is only used to mark the alignment of epsilons and need not be specified by the user in any way, as will be discussed below.

So-called UNKNOWN symbols – placeholders for future alphabet expansion in incremental construction of automata – are denoted by $@$. These are symbols that match any symbol outside the alphabet of an automaton. Note that this is different from the semantics of the $?$ -symbol in regular expressions which represent any single symbol at all with no reference to an alphabet (Beesley and Karttunen 2003).

Conversion of transducers is particularly convenient since we can take advantage of existing algorithms for building complex transducers for NLP use. This includes replacement-rule transducers available in many toolkits, as well as lexicon transducers constructed through essentially right-linear grammars. Figure 1 shows a replacement rule that deletes x -symbols at the end of a string compiled into a transducer, and the result of subsequently converting that transducer to

²A symmetrical unflattening operation can also be defined.

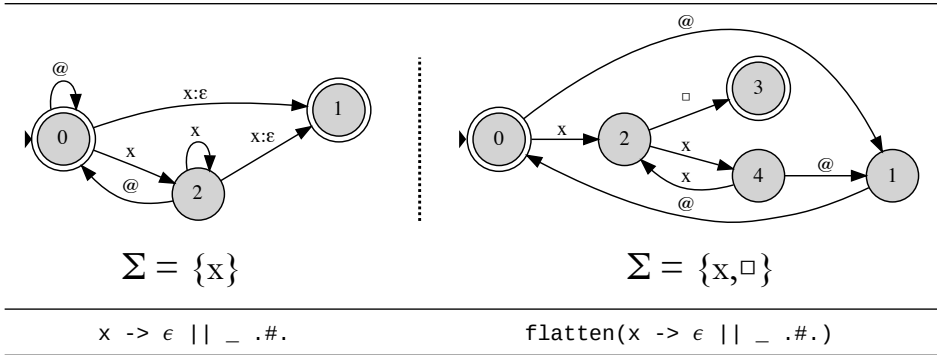


Figure 1: Illustration of a replacement-rule encoded as a transducer (left) and subsequently converted to a 2-tape automaton using the encoding presented here

a standard automaton representing a 2-tape layout in the encoding used here. In other words, we can rely on existing algorithms to build phonological transducers, and only convert them to 2-tape automata before multitape composition.

7 MULTITAPE COMPOSITION

The overall usefulness of converting transducers to 2-tape automata, and then combining a number of individual such 2-tape automata by composition, is illustrated in Figure 2. By combining the individual 2-tape representations into a monolithic n -tape representation

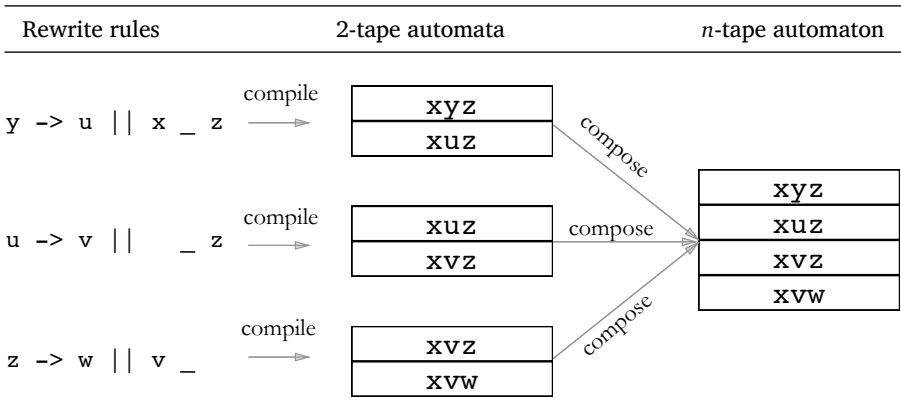


Figure 2: Workflow for converting rewrite rule specifications to transducers, then 2-tape automata, then n -tape automata

tation, all the intermediate representations which would normally be destroyed in a series of compositions can be preserved. As will be seen below, if such a strategy is augmented with the possibility of adding decoration and comment symbols to the individual tapes, very user-friendly grammars for parsing and generation can be developed.

Interestingly, a generic multitape composition algorithm in this representation can be encoded entirely algebraically, which is to say, as regular expressions. Given two multitape automata, A and B , encoded as above, each representing some specified number of tapes m and n , the core idea is to break down their composed representation as a two-step process, which yields an $m + n - 1$ tape representation of the composite. Informally, this multitape composition process for any m and n -tape automata in the representation at hand can be described as follows:

1. Force automata A and B to be of the same number of tapes ($m + n - 1$) by alternatively inserting columns of empty (\square) symbols followed (in A) or preceded (in B) by arbitrary symbols, or retaining the original columns in A and B but inserting arbitrary symbols after each column (in A) or before each column (in B).
2. Call the new automata A_{extend} and B_{extend} : now, the result of intersecting the two $A_{\text{extend}} \cap B_{\text{extend}}$ (using standard automaton intersection) represents their composition $A \circ_{MT} B$, seen from a multitape point of view (with intermediate steps retained).

An illustration of the main logic behind the padding and column insertion mechanisms is given in Figure 3. The exact algorithm is given in Algorithm 1.

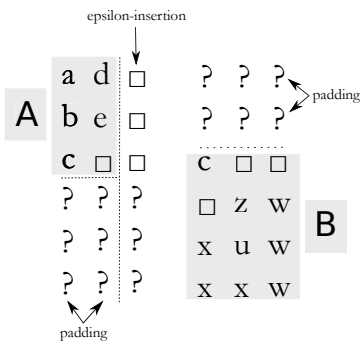


Figure 3:

Illustration of multitape composition: the shaded areas show possible contents of the original multitape automata A and B , while the remaining areas show the result of insertions to coerce the automata to have the same dimensions and epsilon-behavior before intersection of A and B

Algorithm 1: **Input:** $A = \text{FSM}$ with m tapes, $B = \text{FSM}$ with n tapes
 Multitape Composition

- 1 $\text{eInsertA} \leftarrow (\epsilon : \square)^m \left((\epsilon : ?)^{n-1} - (\epsilon : \square)^{n-1} \right)$
- 2 $\text{padA} \leftarrow ?^m (\epsilon : ?)^{n-1}$
- 3 $\text{eInsertB} \leftarrow \left((\epsilon : ?)^{m-1} - (\epsilon : \square)^{m-1} \right) (\epsilon : \square)^n$
- 4 $\text{padB} \leftarrow (\epsilon : ?)^{m-1} ?^n$
- 5 $\text{ExtendA} \leftarrow \text{range}(A \circ (\text{eInsertA} \cup \text{padA})^*)$
- 6 $\text{ExtendB} \leftarrow \text{range}(B \circ (\text{eInsertB} \cup \text{padB})^*)$
- 7 $\text{Sync} \leftarrow ?^{m+n-1}$
- 8 $X0 \leftarrow ?^{m-1} \square^n$
- 9 $XY \leftarrow \left(?^{m-1} - \square^{m-1} \right) \square \left(?^{n-1} - \square^{n-1} \right)$
- 10 $OY \leftarrow \square^m ?^{n-1}$
- 11 $\text{Filter} \leftarrow \neg(\text{Sync}^* (OY(X0 \cup XY) \cup X0(OY \cup XY)) ?^*)$
- 12 $\text{Result} \leftarrow \text{ExtendA} \cap \text{ExtendB} \cap \text{Filter}$

7.1 Path filtering

A well known problem of standard composition algorithms for transducers also carries over to the multitape representation; this is the problem of producing multiple alternate paths in the resulting transducer when epsilon-symbols are present (ϵ -multiplicity). The cause of this is that there exist many equivalent paths that yield the same transduction: e.g. $a:\epsilon \circ \epsilon:b$ can be represented as $a:b$, a sequence $a:\epsilon \epsilon:b$, or a sequence $\epsilon:b a:\epsilon$. Figure 4 illustrates different but equivalent outputs for the composition of two multitape automata. None of the multiple paths for describing a relation are incorrect, but the inconvenience of handling the possibility of multiple equivalent parses or generations motivates an attempt to provide unambiguous paths for each composition during the process itself. Furthermore, in a weighted automaton/transducer scenario – which we will not specifically deal with here – use of a non-idempotent semiring can yield incorrect results if multiple paths are not filtered out.

The common solution in the classical transducer domain is to either design a separate filter transducer that serves to prefer some specific order of epsilon-interleaving (Mohri *et al.* 2002) or to incorporate this filter mechanism directly into the composition algorithm (Hulden 2009a). In the multitape case, however, this filtering mechanism can be encoded entirely as a regular language filter which disallows certain interleavings of epsilon-symbols in the string representation, in

A	B	
$\begin{bmatrix} a & d \\ b & e \\ c & \square \end{bmatrix}$	$\begin{bmatrix} c & \square & \square \\ \square & z & w \\ x & u & w \\ x & x & w \end{bmatrix}$	
A \circ_{MT} B		
$\begin{bmatrix} a & d & \square \\ b & e & \square \\ c & \square & \square \\ \square & z & w \\ x & u & w \\ x & x & w \end{bmatrix}$	$\begin{bmatrix} a & \square & d \\ b & \square & e \\ c & \square & \square \\ \square & z & w \\ x & u & w \\ x & x & w \end{bmatrix}$	$\begin{bmatrix} a & d & \square & \square \\ b & e & \square & \square \\ c & \square & \square & \square \\ \square & \square & z & w \\ x & \square & u & w \\ x & \square & x & w \end{bmatrix}$
$\begin{bmatrix} a & \square & d & \square \\ b & \square & e & \square \\ c & \square & \square & \square \\ \square & z & \square & w \\ x & u & \square & w \\ x & x & \square & w \end{bmatrix}$	$\begin{bmatrix} a & \square & \square & d \\ b & \square & \square & e \\ c & \square & \square & \square \\ \square & z & w & \square \\ x & u & w & \square \\ x & x & w & \square \end{bmatrix}$	

Figure 4: Composition of automata **A** and **B**, illustrating different alignments of epsilon-symbols. This shows composition behavior with respect to two particular configurations in **A** and **B**. A subsequent filter, expressed as an automaton, removes all the solutions except the upper leftmost one

particular those where an $x:\square$ -transition (when automaton *A* has an epsilon on the last tape in some position) immediately follows or precedes a $\square:y$ -transition (when automaton *B* inserts a symbol on its first pair of tapes). This filter can then be intersected with the output of the earlier algorithm. As mentioned, this regular expression (`Filter`) can simply be intersected with the earlier result to remove redundant paths in the composition (shown in lines 7-11 in the algorithm).

7.2 Algorithm details

The algorithm in 1 essentially reiterates the above, with a few details worth mentioning. In lines 1–4, constants that perform the insertion and padding are declared. Lines 5 and 6 create the transducers A_{extend} and B_{extend} . Lines 7–11 create the filter automaton which is independent of *A* and *B*, and the three-element intersection at line 12 yields the result of the final composition.

The composition algorithm is the only extension needed to retain all the intermediate information in an ordered rewrite-rule grammar. One can simply convert any individual transducers to a multitape representation and proceed with the composition, yielding a multitape representation of the same grammar. Parsing and generation of a string s can be performed by creating a padded multitape automaton where either the underlying representation or the surface representation is in place, with arbitrary symbols present on the other tapes. This multitape automaton can then be intersected with the grammar G , yielding a string representation of the set of legal parses or generations, with their intermediate representations intact.

That is to say, if we have an n -tape automaton grammar G and want to parse a string s , we can convert the string to an automaton that accepts that string (ignoring possible intervening blanks \square), pad the automaton to match the number of tapes in G (making sure s is on the last tape), and then intersect with G . The padding operation may be performed by the standard method of composing with a transducer that inserts the right amount of arbitrary symbols, and then extracting the range of the transducer.

$$\text{Parse}(s, G) \stackrel{\text{def}}{=} \text{range}(s/\square \circ ((\epsilon ?)^{n-1} ?)^*) \cap G \quad (1)$$

Likewise, to generate, we may perform the same calculation with the padding done in such a manner that s is on the first tape:

$$\text{Generate}(s, G) \stackrel{\text{def}}{=} \text{range}(s/\square \circ ((? \epsilon ?)^{n-1})^*) \cap G \quad (2)$$

Again, these functions are intended to make the system transparent to the user so that no knowledge of the actual multitape representation is needed to design and apply grammars.

8.1 *Adding intermediate information*

It was hinted above that annotating the effect of various transducers is a very useful feature (as seen in Table 1) for debugging or phonological analysis. Incorporating such information can be done separately from the multitape encoding; that is, one can first incorporate the desired decorative information in a standard transducer and then perform the conversion to a multitape representation, retaining the decoration. For

morphophonological processes, it suffices to modify the transducers that encode the relevant replacement rules in such a way as to add information about each process. In most cases, this would only entail naming the process in question. Such an annotation mechanism can be added separately to each rule transducer before converting it to a 2-tape representation.

8.2 *Decoration example*

In the examples below, each alternation rule transducer is augmented with a textual description of that rule. This allows us to pair up rule descriptions with rules, so that when parsing or generating with a multitape automaton, informative descriptions will appear for each rule in a chain of compositions. In essence, this allows for the inclusion of comments whenever a phonological alternation rule fires, similar to those given in Table 1.

For example, a rule that deletes the latter of consecutive vowels can be encoded as follows as a rule-description pair:

```
('V -> 0 || V _ ', 'Vowel Del')
```

and would have the following effect on input words (a) **papiin** and (b) **papi**, respectively, when generating words:

(a)	(b)
p a p i i n	p a p i
p a p i n # Vowel Del	p a p i # Vowel Del

making it clear to the user that this particular rule applies at that point in the derivation.

9 IMPLEMENTATION

As the *foma* tool has existing Python bindings that can be used to call the underlying standard algorithms for manipulating automata and transducers, providing an extension to that library becomes a matter of implementing the above algorithms. The multitape encoding has been implemented as a standard Python-class that (1) provides a multitape automaton data type *MTFSM* and (2) can perform composition together with rule decoration on arbitrary transducers. This allows for

a certain level of transparency in the bookkeeping needed. For example, the information about how many tapes are encoded in an FSM is auxiliary information that it is necessary to store during a composition process, since the multitape encoding does not inherently contain this information. The interface to the *foma* formalism allows for automatic conversion of transducers to 2-tape automata, which may then be incrementally composed to yield representations with multiple tapes. In effect, designing a complete grammar does not require the user to possess knowledge about or keep track of the underlying machinery, such as the number of tapes used, the padding performed, etc. Even the padding symbols – though helpful for debugging individual rules – can be omitted from the output as they are only used internally to produce a consistent alignment of different-length strings.

For example, to simply compose two rules, without any decoration, the user may enter arbitrary regular expressions (in this example rewrite rules) which automatically convert to two-tape representations that can be composed and inspected:

```
>>> r1 = MTFSM("x -> y || c _ ")
>>> r2 = MTFSM("y -> z || _ d")
>>> composed = r1 + r2

>>> print composed
States: 35
Transitions: 126
Final states: 7
Deterministic: 1
Minimized: 1
Numtapes: 3
```

Entire grammars can be compiled through a separate and more involved `mtgrammar` module. This module allows for the type of rule decoration described above, and provides for a method of composing the different multitape automata in order, as well parsing and generation functionality:

```
from mtgrammar import *

G = compilemt([('b -> x || a _ c', 'Rule 1'), ('x -> 0 || a _ c', 'Rule 2')])
printparses('ac', G, dir='up')
```

Here, two rewrite rules are compiled, converted automatically to multitape automata through the `compilemt` statement and composed

in the order given. After this, the resulting 3-tape automaton is used to parse the word *ac* in the “upward” direction, that is, assuming that the string is on the output tape. This produces the three aligned outputs:

abc□□	axc□□	ac□□
axc#Rule 1	axc#Rule 1	ac#Rule 1
a□c#Rule 2	a□c#Rule 2	ac#Rule 2
a□c□□	a□c□□	ac□□

Here, we see that there are three ways the two phonological rules in question could produce the output *ac* – by starting from the underlying forms *abc*, *axc*, and *ac*, respectively. The blanks are automatically positioned in their correct positions without the user having to specify anything except the input string to be parsed and the direction of parsing (up = from surface form to underlying form, down = from underlying form to surface form).

9.1 *Illustrative example 1: phonology (Lardil)*

Returning now to the original Lardil example: annotating replacement rules with additional descriptive symbols to be inserted at the ends of strings every time a rule fires in combination with the multitape composition mechanism allows us to essentially automatically replicate the linguist-friendly representation given in Table 1. The following snippet illustrates some key points in the design of such grammars:

```

1 from foma import *
2 from mtgrammar import *
3
4 # Definitions #
5 FST.define(u'{jilijili}|{kitikiʔi}|{munʔkumʔku}', u'Stems')
6 FST.define(u'[a | æ | i | u]', 'Vow')
7 FST.define(u'[m | n | ŋ | ŋ | ŋ | ŋ | nʲ]', 'Nasal')
8 ...
9 # Rules #
10 kEpenthesis = (u'[..] -> k || Nasal _ u ɿ ', 'k-Epenthesis')
11 wEpenthesis = (u'[..] -> w || i _ u', 'w-Epenthesis')
12 ...
13 G = compilemt((Lex, kEpenthesis, wEpenthesis, VowelDeletion, FinalLowering,
14 Apocope, ClusterRed, NonApicalDel, Sonorantization))

```

That is, we may write grammars in much the same way as in established formalisms, defining regular expression constants such as *Vow* and *Nasal* which are later used in building more complex rewrite rules such as *k-Epenthesis* and *w-Epenthesis*, etc. These decorations are

automatically added to the right end of each tier, as illustrated in two different parses below:

```
>>> printparses(u'muŋkumu', G)           >>> printparses(u'putu', G)

muŋkumuŋku[Uninflected]□□             putuka[Uninflected]□□
muŋkumuŋku□#Lexicon Output             putuka□#Lexicon Output
muŋkumuŋku□#k-Epenthesis               putuka□#k-Epenthesis
muŋkumuŋku□#w-Epenthesis              putuka□#w-Epenthesis
muŋkumuŋku□#Vowel Deletion            putuka□#Vowel Deletion
muŋkumuŋka□#Final Lowering            putuka□#Final Lowering
muŋkumuŋk□#Apocope                    putuk□#Apocope
muŋkumuŋ□□#Cluster Reduction          putuk□□#Cluster Reduction
muŋkumu□□□#Non-Apical Deletion        putu□□□#Non-Apical Deletion
muŋkumu□□□□#Sonorantization          putu□□□□#Sonorantization
muŋkumu□□□□□□□□                    putu□□□□□□□□
```

In the generation direction, the same procedure applies, and the library offers an up/down parameter to control for the direction of operation; a command `printparses(u'putuka[Uninflected]', G, dir='down')` in the above would have produced the same output as the example on the right hand side.

9.2 *Illustrative example 2: Historical Linguistics (Proto-Indo-European)*

As alluded to above, another scenario where intermediate, possibly annotated strings provide important information is in the modeling of historical sound change by finite-state means. In the development of models of diachronic sound change, this provides the possibility of providing annotated parses from modern variants to proto-language forms given hypothesized chronological sound changes. The following parses show the behavior of an ordered set of rewrite rules in multitape form that model the path of sound changes from Proto-Indo-European (PIE) to German and Latin. The relevant rules are implemented as rewrite transducers as in the Lardil example above.

```
>>> printparses(u'pátēr', Latin)       >>> printparses(u'fáter', German)

ph2tér□□                               ph2tér□□
ph2tér□#PIE                             ph2tér□#PIE
ph2tér□#Szemerényi's law                ph2tér□#Szemerényi's law
pa□tér□#*H > a between consonants      pa□tér□#*H > a between consonants
pá□tér□#Proto-Italic stress              fa□tér□#Grimm's Law
pá□tér□□□                               fa□dér□#Verner's Law
pá□tér□□□                               fá□dér□#Stress Shift
pá□tér□□□                               fá□tér□#High Germanic Consonant Shift
pá□tér□□□                               fátér□#Lengthening
pá□tér□□□                               fáter□#Reduction
pá□tér□□□                               fáter□□□
```

Here, we see the parsing of the Latin form for the word “father”, *pátēr* as well as the German form *fátvr*, using two different grammars that share part of the rewrite rules (the early sound changes affecting both). Both correspond to the underlying, hypothesized PIE form *ph₂térs*. The relevant sound changes in this grammar were modeled following Beekes (2011); Trask (1996). As opposed to synchronic phonological grammars, the chains of sound changes over long periods can grow quite extensive. For example, the German surface form is subject to a number of them: first, a sound change called *Szemerényi’s law* deleting coda fricatives takes place, followed by a process of laryngeal vocalization,³ Grimm’s and Verner’s Laws, a stress shift, as well as a number of processes that affect vowels. The multitape parse in this case illustrates the value of such a design in checking correctness of very complex sequences of sound changes. Such sequences could plausibly be generated in the chronological direction through non-finite-state means, but the direction of interest for the linguist is generally the inverse one – parsing from surface form to underlying form, which is what is calculated here.

More advanced usage scenarios can also be explored with the method through more complex intersections of individual tapes in multitape representations for different languages. For example, having postulated a sequence of sound changes that two modern languages have undergone from the proto-language, we can calculate the set of *possible* proto-forms for some modern cognates *x* and *y* in two languages. In the above parses of “father”, only a single parse per cognate is given, since we have included the postulated proto-form in the grammar. There might, however, exist other plausible PIE-forms that fit the sequence of sound changes. For example, removing the proto-form from the grammar yields two plausible parses in the intersection of Latin and German, *patér* and *patérs*. Such techniques can be extended to a larger scale to support the endeavor of verifying consistency of postulated sound changes with the possibility of immediate feedback when minor changes are made in the various sound laws.

³Laryngeals are abstract segments proposed to have been present in Proto-Indo-European (De Saussure 1879) but later disappeared, leaving behind different vowel qualities and a compensatory lengthening. The laryngeals are commonly labeled **h₁*, **h₂*, and **h₃*, and **H* is used as a cover symbol for all three.

This paper has presented a general, automatic method for extending finite-state grammars in the composed rewrite-rule tradition. The method in effect replaces the use of transducers with multitape automata, which are shown to have the capacity to provide rich parses and to support elaborate annotation of intermediate forms. Existing algorithms for constructing transducers from rewrite-rule specifications can still be used, once converted to multitape representations. We can also take advantage of specialized string-rewriting and constraint systems to handle syllabification (Hulden 2006), Semitic interdigitation (Beesley and Karttunen 2000), and, with some caution, unification features such as flag diacritics to model long-distance dependencies (Beesley 1998). Potentially, steps in candidate removal in Optimality Theoretic grammars could also be implemented by incorporating proposals to model such processes by finite-state composition (Karttunen 1998; Gerdemann and van Noord 2000; Gerdemann and Hulden 2012).

The model itself assumes little machinery beyond the ability to compose the resulting multitape automata, but offers a way to produce rich representations of grammars constructed in this vein. If desired (for memory efficiency reasons), the resulting multitape automata can still be re-converted to transducers by eliminating the intermediate representations. This offers the possibility to only use the multitape representation for debugging purposes, if the final intent is to produce a simpler underlying-to-surface mapping or vice versa.

The above techniques may be useful for applications outside standard designs of morphophonological grammars. In modeling historical sound changes, for example, ‘debugging’ problems similar to those in phonology and morphology tend to arise – much exacerbated by the fact that one is often dealing with multiple languages at the same time. Keeping track of hundreds of proposed sound laws together with their effect on lexical items across languages is a task that is well suited for the type of modeling presented in this paper.

Although the application focus of this paper has been more along the lines of modeling traditional non-probabilistic grammars, the methods presented above – the composition algorithm in particular – are also adaptable to weighted automata.

APPENDIX A – LARDIL GRAMMAR

```
# -*- coding: utf-8 -*-
from foma import *
from mtgrammar import *

# Definitions needed for rules
FST.define(u'{papi}|{witæ}|{ŋuku}|{wanka}|{karikari}|{jukarpa}|{putuka}|
          {jilijili}|{kitikitī}|{muŋkumuŋku}', u'Stems')
FST.define(u'[a | æ | i | u]', u'Vow')
FST.define(u'[p | t | t̥ | t̄ | k | m | n | ŋ | ŋ̣ | ŋ̤ | ŋ̥ | ŋ̦ | r | l |
          w | ɹ | j ]', u'Cons')
FST.define(u'[t | t̥ | n | ŋ | r | l | ɹ ]', u'Apical')
FST.define(u'[m | n | ŋ | ŋ̣ | ŋ̤ | ŋ̥]', u'Nasal')
FST.define(u'""|. #.', u'E') # Word edge

# Grammar Lexicon + Rules
Lex = (u'Stems ["[Acc. Nonfuture]":{in} |
              "[Acc. Future]":{uɹ} |
              "[Uninflected]":0 ], u'Lexicon Output')
kEpenthesis = (u'[.] -> k || Nasal _ u ɹ ', u'k-Epenthesis')
wEpenthesis = (u'[.] -> w || i _ u', u'w-Epenthesis')
VowelDeletion = (u'Vow -> 0 || Vow _ ', u'Vowel Deletion')
FinalLowering = (u'i -> æ, u -> a || _ E', u'Final Lowering')
Apocope = (u'Vow -> 0 || Vow Cons* Vow Cons* _ E', u'Apocope')
ClusterRed = (u'Cons -> 0 || Cons _ E', u'Cluster Reduction')
NonApicalDel = (u'Cons - Apical -> 0 || _ E', u'Non-Apical Deletion')
Sonorantization = (u't̥ -> ɹ || _ E', u'Sonorantization')

Grammar = compilemt((Lex, kEpenthesis, wEpenthesis, VowelDeletion,
                    FinalLowering, Apocope, ClusterRed, NonApicalDel, Sonorantization))

## Parse ##
mtgrammar.printparses(u'muŋkumu', Grammar)
mtgrammar.printparses(u'putu', Grammar)
mtgrammar.printparses(u'ŋukuɹ', Grammar)
```

REFERENCES

- Iñaki ALEGRIA, Izaskun ETXEBERRIA, Mans HULDEN, and Montserrat MARITXALAR (2010), Porting Basque morphological grammars to foma, an open-source tool, 6062:105–113.
- Mohamed ALTANTAWY, Nizar HABASH, Owen RAMBOW, and Ibrahim SALEH (2010), Morphological Analysis and Generation of Arabic Nouns: A Morphemic Functional Approach, in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta.
- Robert S. P. BEEKES (2011), *Comparative Indo-European Linguistics: an Introduction*, John Benjamins Publishing.
- Kenneth R BEESLEY (1998), Constraining separated morphotactic dependencies in finite-state grammars, in *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pp. 118–127, Association for Computational Linguistics.
- Kenneth R BEESLEY (2012), Kleene, a Free and Open-Source Language for Finite-State Programming, in *10th International Workshop on Finite State Methods and Natural Language Processing (FSMNLP)*, pp. 50–54.
- Kenneth R BEESLEY and Lauri KARTTUNEN (2000), Finite-state Non-Concatenative Morphotactics, in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*.
- Kenneth R. BEESLEY and Lauri KARTTUNEN (2003), *Finite State Morphology*, CSLI Publications, Stanford, CA.
- Noam CHOMSKY and Morris HALLE (1968), *The Sound Pattern of English*, Harper & Row.
- Ferdinand DE SAUSSURE (1879), *Mémoire sur le système primitif des voyelles dans les langues indo-européennes*, B.G. Teubner.
- Dale GERDEMANN and Mans HULDEN (2012), Practical Finite State Optimality Theory, in *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pp. 10–19, Association for Computational Linguistics, Donostia–San Sebastián.
- Dale GERDEMANN and Gertjan VAN NOORD (2000), Approximation and exactness in finite state optimality theory, in *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*.
- Nizar HABASH and Owen RAMBOW (2006), MAGEAD: A Morphological Analyzer and Generator for the Arabic Dialects, in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pp. 681–688, Association for Computational Linguistics, Sydney, Australia, doi:10.3115/1220175.1220261, <http://www.aclweb.org/anthology/P06-1086>.

- Nizar HABASH, Owen RAMBOW, and George KIRAZ (2005), Morphological analysis and generation for Arabic dialects, in *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pp. 17–24, Association for Computational Linguistics.
- Kenneth HALE (1973), Deep-surface canonical disparities in relation to analysis and change: An Australian example, *Current trends in linguistics*, 11:401–458.
- Bruce HAYES (2011), *Introductory Phonology*, John Wiley & Sons.
- Mans HULDEN (2006), Finite-State Syllabification, *Lecture Notes in Artificial Intelligence*, 4002:86–96.
- Mans HULDEN (2009a), *Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology*, Ph.D. thesis, University of Arizona.
- Mans HULDEN (2009b), Foma: a finite-state compiler and library, in *Proceedings of the 12th conference of the European Chapter of the Association for Computational Linguistics*, pp. 29–32.
- Mans HULDEN (2009c), Regular Expressions and Predicate Logic in Finite-State Language Processing, in Jakub PISKORSKI, Bruce WATSON, and Anssi YLI-JYRÄ, editors, *Finite-State Methods and Natural Language Processing—Post-proceedings of the 7th International Workshop FSMNLP 2008*, volume 191 of *Frontiers in Artificial Intelligence and Applications*, pp. 82–97, IOS Press.
- Mans HULDEN (2009d), Revisiting multi-tape automata for Semitic morphological analysis and generation, *Proceedings of the EACL 2009 Workshop on Computational Approaches to Semitic Languages*, pp. 19–26.
- Mans HULDEN (2015), Grammar design with multi-tape automata and composition, in *Proceedings of the The 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP)*, Association for Computational Linguistics.
- Ronald M. KAPLAN and Martin KAY (1994), Regular models of phonological rule systems, *Computational Linguistics*, 20(3):331–378.
- Lauri KARTTUNEN (1998), The proper treatment of optimality theory in computational phonology, in *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing (FSMNLP)*.
- Lauri KARTTUNEN (2003), Computing with realizational morphology, in *Computational Linguistics and Intelligent Text Processing*, pp. 203–214, Springer.
- Martin KAY (1987), Nonconcatenative Finite-State Morphology, in *Proceedings of EACL 1987*.
- André KEMPE, Franck GUINGNE, and Florent NICART (2004), Algorithms for weighted multi-tape automata, *XRCE Research Report 2004/031*.

André KEMPE and Lauri KARTTUNEN (1996), Parallel replacement in finite state calculus, in *Proceedings of the 34th annual meeting of the Association for Computational Linguistics*.

Michael KENSTOWICZ and Charles KISSEBERTH (1979), *Generative phonology*, Academic Press.

George Anton KIRAZ (2000), Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic, *Computational Linguistics*, 26(1):77–105.

George Anton KIRAZ (2001), *Computational nonlinear morphology: with emphasis on Semitic languages*, Cambridge University Press, Cambridge.

Kimmo KOSKENNIEMI (1983), *Two-level morphology: A general computational model for word-form recognition and production*, Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.

Mehryar MOHRI, Fernando PEREIRA, and Michael RILEY (2002), Weighted finite-state transducers in speech recognition, *Computer Speech & Language*, 16(1):69–88.

Mehryar MOHRI and Richard SPROAT (1996), An efficient compiler for weighted rewrite rules, in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 231–238, Association for Computational Linguistics.

Erich ROUND (2011), Word final phonology in Lardil: Implications of an expanded data set, *Australian Journal of Linguistics*, 31(3):327–350.

Robert Lawrence TRASK (1996), *Historical Linguistics*, Oxford University Press.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.

<http://creativecommons.org/licenses/by/3.0/>



A probabilistic model of Ancient Egyptian writing

Mark-Jan Nederhof and Fahrurrozi Rahman
School of Computer Science
University of St Andrews
United Kingdom

ABSTRACT

This article offers a formalization of how signs form words in Ancient Egyptian writing, for either hieroglyphic or hieratic texts. The formalization is in terms of a sequence of sign functions, which concurrently produce a sequence of signs and a sequence of phonemes. By involving a class of probabilistic automata, we can define the most likely sequence of sign functions that relates a given sequence of signs to a given sequence of phonemes. Experiments with two texts are discussed.

Keywords:
Ancient Egyptian,
writing systems,
language models

1

INTRODUCTION

Ancient Egyptian writing, used in Pharaonic Egypt, existed in the form of *hieroglyphs*, often carved in stone or painted on walls, and sometimes written on papyrus (Allen 2000). Hieroglyphs depict people, animals, plants and various kinds of objects and geographical features. A cursive form of Ancient Egyptian writing, called *hieratic*, was predominantly written on papyrus. Most hieratic symbols can be seen as simplified hieroglyphs, to such an extent that it is difficult for the modern untrained eye to tell what is depicted. Because hieratic handwriting varied considerably over time, with notable differences between regions and scribes, the creation of computer fonts for hieratic is problematic, and consequently scholars commonly resort to publishing hieratic texts in a normalized hieroglyphic font. Since Version 5.2,

Unicode contains a selection of 1071 hieroglyphs. Henceforth we will use the term *sign* to refer to a hieroglyph or a hieratic symbol.

The Ancient Egyptian language is in the family of Afro-Asiatic languages, which includes the Semitic languages (Loprieno 1995). As in writing systems of several Semitic languages (e.g. Hebrew, Arabic, Phoenician), only consonants are written. Modern scholars use 24 or 25 letters to transliterate Ancient Egyptian texts in terms of these consonants. Most are written as Latin characters, some with diacritical marks, plus aleph ʾ and ayin ʿ. An equal sign is commonly used to precede suffix pronouns; thus *sḏm* means “to hear” and *sḏm=f* “he hears”. A dot can be used to separate other morphemes; for example, in *sḏm.tw=f*, “he is heard”, the morpheme *tw* indicates passive.


The Ancient Egyptian writing system itself is a mixture of phonetic and semantic elements. The most important are *phonograms*, *logograms* and *determinatives*. A phonogram is a sign that represents a sequence of one, two or three letters, without any semantic association. A logogram represents one particular word, or more generally the lemma of a word or a group of etymologically related words. A determinative is commonly written at the end of a word, following phonograms, to clarify the meaning of a word; in their most obvious use, determinatives disambiguate between homophones, or more precisely, different words consisting of the same consonants. In addition, there are *typographical* signs, for example, three strokes that indicate the plural form of a noun (also used for collective nouns). These and more classes of signs are discussed in detail in Section 2.

What makes automatic analysis of Ancient Egyptian writing so challenging is that there was no fixed way of writing a word, so that table-lookup is largely ineffective. Even within a single text, the same word can often be found written in several different ways. Moreover, one sign can often be used in different functions, e.g. as phonogram or as determinative. Some signs can be used as different phonograms with different sound values. Together with the absence of word boundary markers, this makes it even hard to segment a text into words.

Generalizing statements can be made about writings of words. Typically, either a word starts with a number of phonograms, covering all the letters of the stem, possibly some covered more than once, followed by one or more determinatives, or a word starts with a logogram, possibly followed by one or more phonograms, possibly fol-

lowed by one or more determinatives. More phonograms can follow the determinatives for certain suffixes. This coarse description is inadequate however to model the wide spectrum of writings of words, nor would it be sufficient to disambiguate between alternative analyses of one sequence of signs.

These factors motivate the search for an accurate and robust model that can be trained on data, and that becomes more accurate as more data becomes available. Ideally, the model should be amenable to unsupervised training. Whereas linguistic models should generally avoid unwarranted preconceptions, we see it as inevitable that our model has some knowledge about the writing system already built in, for two reasons. First, little training material is currently available, and second, the number of signs is quite large, so that the little training material is spread out over many parameters. The *a priori* knowledge in our model consists of a sign list that enumerates possible functions of signs and a formalization of how these functions produce words. This knowledge sufficiently reduces the search space, so that probabilistic parameters can be relatively easily estimated.

In our framework, a *sign function* is formally identified by the combination of (a) the one or more signs of its writing, (b) its class, which could be ‘phonogram’, ‘logogram’, ‘determinative’, etc., and (c) a sequence of letters or a description of a semantic value, depending on the class. One example is the phonogram function for sign  with sound value *r*. There is a logogram function for the same sign, with as value the transliteration of the lemma *r3*, which means “mouth”. A typographical function for the three strokes may have a semantic value ‘plurality or collectivity’.

The first attempt to systematically classify functions of signs in context may have been Schenkel (1984). The proposed system used a notation that is close to traditional transliteration, but with additional elements, capturing *some* functional aspects of *some* used signs. For example, for each determinative in the writing of a word, a superscript giving the name of the sign is added to the transliteration. Use of logograms was indicated by capitalizing letters of the stem in the transliteration. It is not possible however to reconstruct a complete hieroglyphic writing from an instance of this notation, and moreover this system does not seem to lend itself to formalization.

The problem we will address in the experiments is guessing the sign functions given the signs and the letters. This is related to the problem of automatically obtaining transliteration from hieroglyphic text. As far as we are aware, the earliest work to attempt this was Billet-Coat and Hérin-Aime (1994), which focussed on a multi-agent architecture to combine expert knowledge about signs, words and clauses. Another approach to automatic transliteration, by Tsukamoto (1997), used Unix applications such as ‘grep’ and ‘sed’. The approach by Rosmorduc (2008) used manually produced rewrite rules. Further work along these lines by Barthélemy and Rosmorduc (2011) used two approaches, namely cascades of binary transducers and intersections of multitape transducers, with the objective to compare the sizes of the resulting automata.

A more modest task is to automatically align given hieroglyphic text and transliteration, as considered by Nederhof (2008), who used an automaton-based approach with configurations, similar to that in Section 5, except that manually determined penalties were used instead of probabilities. As we will demonstrate, the use of probabilities allows training of parameters of the model.

Relating hieroglyphic texts and their Egyptological transliteration is an instance of relating two alternative orthographic representations of the same language. The problem of mechanizing this task is known as machine transliteration. For example, Knight and Graehl (1998) consider translation of names and technical terms between English and katakana, and Malik *et al.* (2008) consider transliteration between Hindi and Urdu. Another very related problem is conversion between graphemes and phonemes, considered for example by Galescu and Allen (2002).

Typical approaches to solve these tasks involve finite-state transducers. This can be justified by the local dependencies between input and output, that is, ultimately the transliteration can be broken down into mappings from at most n to at most m symbols, for some small n and m . For Ancient Egyptian however, it is unclear what those bounds on n and m would be. We therefore depart from finite-state methods, and propose a model that involves a tape, with a tape head that can jump left as well as right. This idea is reminiscent of alignment models of machine translation (Brown *et al.* 1993) and of the Operation Sequence Model (Durrani *et al.* 2015).




Sproat (2000) formulates the *Regularity* hypothesis, stating that orthographic processes can be realized in terms of finite-state methods. For Ancient Egyptian, he singles out two isolated phenomena, namely a particular writing of plurality (cf. Section 2.6) and honorific transposition (cf. Section 4). He argues that whereas their realization requires extra care, they can be realized in terms of finite-state methods nonetheless. He ignores more problematic phenomena however, such as phonetic complements (cf. Section 2.2) and phonetic determinatives (cf. Section 2.4), which are core elements of the writing system and form the main motivation for our non-finite-state automaton model. Thereby, Ancient Egyptian remains a significant challenge to the Regularity hypothesis.


In the sequel, we let ‘Egyptian’ refer to ‘Ancient Egyptian’. The structure of this paper is as follows. Section 2 explains in more detail the sign functions that are distinguished in our model of Egyptian writing. An annotated sign list couples sign functions to signs, as explained in Section 3. The annotated texts themselves, which were used for training and testing, are presented in Section 4. A formal model of Egyptian writing is the subject of Section 5, extended with probabilities in Section 6. Experiments are discussed in Section 7.

2 SIGN FUNCTIONS

In our formal framework, we distinguish the sign functions that are explained in the following sections. Except for ‘spurious’ functions, each function has exactly one value, specified at the end of each section.

2.1 *Logograms*


A *logogram* is a sign that represents a word, or more accurately, the lemma of a word, or possibly a group of etymologically related words with closely related meanings. Often a logogram depicts the word it represents. For example, the aforementioned sign  can be a logogram for *ꜣ*, “mouth”. In other cases, a logogram may represent an idea that can be associated with the thing that is depicted, rather than the thing itself. For example,  depicts a (standing) leg, while its meaning is the word *bw*, “place”. A related example is the sign  depicting (walking) legs, with meaning *jw*, “to come”.



An example where we would include etymologically related words is the following. The sign  can literally mean the thing that is depicted, namely *bjt*, “bee”, but the same sign is used in much the same way for the etymologically related word *bjt*, “honey”.

The value of a logogram is the transliteration of the lemma that it represents.



2.2


Phonograms


Much of the Ancient Egyptian writing system evolved via the principle of *rebus writing* (Daniels and Bright 1996), that is, the use of a sign solely for its sound value, derived from one or more sounds that occur in the word expressing what the sign depicts. For example, from the logographic use of sign  for *bw*, “place”, the use as *phonogram* evolved, allowing it to represent the letter *b* in the writing of any word.

For each letter, there is at least one phonogram that represents that letter in isolation. We call such a phonogram *unilateral*. There are also several dozens of phonograms for sequences of two or three letters. For example,  is a (biliteral) phonogram with sound value *wn* and  is a (trilateral) phonogram with sound value *tjw*.

A word is often written using several phonograms, which together cover some letters more than once. A unilateral phonogram representing a letter that is also represented by a neighboring biliteral or trilateral phonogram is known as a *phonetic complement*; there are examples in Figure 3 that will be discussed later.

As pointed out by e.g. Schenkel (1984), it can be very hard to distinguish between logograms and phonograms, especially in the case of trilateral phonograms that can by themselves write a whole word. For example,  can stand for the word *whmt*, “hoof”, and in this use it is obviously a logogram, but it can also stand for the word *whm*, “to repeat”. (The *t* in *whmt* is the feminine ending.) It is plausible that the two words are etymologically related, as the depicted cloven hoof ‘repeats’ a toe. However, traditionally the use of  in “to repeat” is analyzed as phonogram, as if its use was motivated by accidental similarity of the pronunciations of the two words. We have adopted that view.

One more example is the sign , which is primarily used as logogram for *ntr*, “god”. It is also used in the writing of the word *sntr*,







“incense”, and one may naively interpret it as phonogram there. However, it is very likely that the sign is not merely chosen for its sound value, but for its semantic relationship to *ntr*, “god”, in combination with the causative prefix *s-*. An alternative etymology suggested by de Vartavan (2010) involves the verb *sn*, “to smell”, in combination with *ntr*, but either way, the sign  in *sntr* is best analyzed as logogram.





In later stages of Egyptian, some pairs of sounds from earlier stages merged together. As a result, the corresponding signs were sometimes confused. One example is the use of a sign representing the sound *t* for writing a word whose historical pronunciation had a different sound *t̄*. In our framework, we let the value of a phonogram be its historical sound value, regardless of how it is used. However, we follow Hannig (1995) in not distinguishing *s* from *z*. In Middle Egyptian, these two sounds had merged together to such an extent that even the (conservative) writing system treated both as largely exchangeable. Both sounds are therefore transliterated as *s*.

2.3

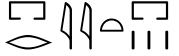


Determinatives


A *determinative* is a sign that derives a semantic value from what is depicted, much like many logograms. However, determinatives are not used in isolation to form writings of words. Instead they must be combined with logograms and phonograms together covering all the letters. Typically, determinatives occur at the end of a writing, following the logograms and phonograms.




Most determinatives do not pertain to any particular word. For example, the “tree” determinative  is used with various nouns related to trees, plants and wood. Another example is , depicting a papyrus scroll with ties, which is used as determinative for words that express abstract notions. Thus we have    , *jqr*, “excellent”, where the first three signs are each uniliteral phonograms for the three letters in that word.



The sign  can be used as determinative with the general meaning “man and his occupations”. For example, it is used in   , *šmsw*, “follower” (someone accompanying the king). Here the first sign is a logogram for the verb *šms*, “to follow” and the second sign is a

unilateral phonogram with value *w*, a suffix which turns the verb into a masculine noun.

The distinction between determinatives and logograms is illustrated by the word *pryt*, “settlement”, written as . The first sign (reading left-to-right and top-to-bottom for stacked signs) is a logogram depicting the plan of a house, with meaning *pr*, “house”, and derivatives. The next four signs are phonograms together covering the letters *r*, *y* and *t*. Note the *r* in *pryt* is covered by both the logogram and the first phonogram, which makes  here a phonetic complement. The second occurrence of  has a different function from the first. Here it is a determinative, clarifying that the written word has something to do with buildings.


A determinative may also be specific to one lemma. For example,  is generally used only for the noun *mnjt*, “mooring post”, and its derivatives. One may ask what distinguishes such a determinative from a logogram, which is by definition also specific to one lemma. The answer lies in the different roles that logograms and determinatives fulfil in the writing of words, as illustrated above for *pryt*, “settlement”.

When a determinative is specific to one lemma, the same sign can often be used as logogram as well, that is, the sign can be used to write a word without accompanying phonograms. For example,  can as logogram stand on its own for *hr*, “to fall”, but it is determinative in the alternative writing  , *hr*, where it is preceded by two unilateral phonograms.

The value of a determinative specific to a word is the transliteration of that word, such as *mnjt* for . The value of other determinatives is a general description of the kinds of concepts that are covered, such as “building, seat, place” for .

2.4 *Phonetic determinatives*

A *phonetic determinative* is similar to a determinative in that it tends to be placed near the end of a word, next to normal determinatives. However, its value is phonetic, repeating letters already written by logograms and phonograms.

An example is given by the writing of the word *mnḥt*, “splendid (fem.)” in Figure 1. The phonetic determinative  here has phonetic reading *mnḥ*. Note that unlike the phonograms, it occurs near the end of the word, even following the feminine *t* ending.

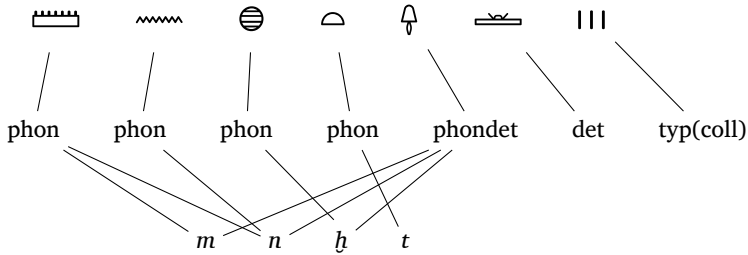






Figure 1:
Use of a phonetic
determinative

Many signs can be both phonograms and phonetic determinatives, even with the same sound value. We then classify an occurrence as the latter only if the corresponding letters have already been accounted for by earlier signs.

The value of a phonetic determinative is its (earliest historical) sound value.

2.5 *Typographical signs*

Signs that fall outside any of the classes above will be called *typographical*. One example is the single stroke written under, or next to, another sign, most often a logogram. Often its function is to indicate that the meaning of that other sign is what is depicted, rather than, say, the sound value of that sign. We then call the single stroke a *semogram marker*. For example,  might mean *r*, “mouth”, while , without semogram marker, might stand for the preposition *r*, “to”. The sign  here is logogram or phonogram, respectively.

The function of the single stroke is not always clear however. More often than not, it acts as *space filler*; at this point we should explain that Egyptian writing is often influenced by aesthetical considerations, in particular the desire to fill up empty spaces between signs. As a consequence,  can either mean *ḥr*, “face”, or *ḥr*, “on”. In the first case, the single stroke is clearly a semogram marker, but in the second it is merely a space filler.

Further typographical symbols consist of a combination of two or three strokes. These are typically written at the end of a noun as

marker of duality or plurality. (Egyptian had a dual form next to a plural.) The three strokes were however also used for singular nouns with collective meanings, such as *rmt*, “people” and *jmnt*, “what is hidden”. The three strokes are also written behind plural personal pronouns. Similarly, two strokes can be used for singular words whose meaning involves the idea of pairing two things or two people. An example is , *snw* = *f*, “his fellow”.

There are also *false* dual and *false* plural writings, with two or three strokes for words that happen to end on *-wj*, *-tj* or *-w*, the masculine and feminine dual and masculine plural endings, while these words are not grammatically dual or plural. In these cases the group of two or three strokes is analyzed as phonogram with sound value *wj*, *j* (without the feminine ending *t*) or *w*. It is not always easy however to determine whether words ending on *-wj/-tj/-w* are (historically) dual or plural.

Further typographical symbols include the numerals. We analyze a number written using a sequence of numerals as one sign function. Egyptian numerals are a topic by themselves (Ifrah 1981) and further discussion here would not be productive.

A peculiar typographical function exists in a combination of signs that indicates the preceding (phrase, word or sequence of letters) should be read twice. An example is , *sksk*, “to destroy”. Here the first three signs are phonograms together accounting for the first two letters *sk* of *sksk*. The following group then indicates the letters *sk* should be read a second time.

As value of a typographical function we take a description, which can be, for example, “semogram marker”, “space filler”, “duality”, “plurality or collectivity”, “replaces human figure, or sign difficult to draw” and “number”.

2.6 Multiplication of signs

We discussed above that duality and plurality (and collectivity) can be expressed by two or three strokes. There is an alternative way to express the same, by repeating a sign once or twice. For example, the logogram stands for *ntr*, “god”. By repeating it twice, we obtain , *ntrw*, “gods”. We recall *-w* is the masculine plural ending.

Typically only the last sign of a singular writing is repeated to obtain a dual or plural writing, but sometimes larger groups of signs are repeated. For example, $\overline{\text{m}}$ stands for *m*, “name”, written with two uniliteral phonograms for *r* and *n*, respectively. The plural can be written $\overline{\text{m}}\overline{\text{m}}\overline{\text{m}}$, *mw*, “names”.

Also determinatives may be repeated. An example is the writing of “the two lands (Upper and Lower Egypt)” as $\overline{\text{t}}\overline{\text{w}}\overline{\text{j}}$. We recall $\overline{\text{w}}\overline{\text{j}}$ is the masculine dual ending. A typical writing for the singular is $\overline{\text{t}}\overline{\text{w}}\overline{\text{j}}$, *t*, “land”, written with a logogram for *t*, depicting a strip of land with three grains of sand, a semogram marker, and a determinative depicting irrigated land.







We have chosen a modeling of such writings that allows straight-forward automatic processing. This consists in taking all repeated signs together to correspond to a single function indicating plurality. In the example of “names”, the first occurrences of $\overline{\text{r}}$ and $\overline{\text{n}}$ are analyzed as phonograms *r* and *n*, respectively, as they would be in the singular writing of the word. The two remaining occurrences each of $\overline{\text{r}}$ and $\overline{\text{n}}$ together indicate plurality. An example for duality, as illustrated in Figure 2a, will be discussed later.

As in the case of the dual and plural strokes (Section 2.5), there are false dual and false plural writings using duplication of signs. Common examples concern the nisbe form. A *nisbe* is an adjective derived from a noun by adding the ending *-j*. For example, $\overline{\text{n}}\overline{\text{j}}\overline{\text{w}}\overline{\text{t}}$ means “town” while $\overline{\text{n}}\overline{\text{j}}\overline{\text{w}}\overline{\text{t}}\overline{\text{j}}$ means “concerning the town; local”. The latter word is typically written as $\overline{\text{n}}\overline{\text{j}}\overline{\text{w}}\overline{\text{t}}\overline{\text{j}}$, which should be read “local” and not “the two towns”.

There are cases of plural and collective nouns that are written using three similar but *distinct* signs. For example, the word that means “cattle” can be accompanied by three determinatives depicting different kinds of cattle, and the word that means “birds” can be accompanied by three determinatives depicting different species of birds. These cases are rare enough to be ignored for the purposes of our model in Section 5. At this point we should emphasize that playfulness and creativity are important features of Egyptian writing, and this precludes existence of an exhaustive list of orthographic phenomena.


The value of a multiplicative function is a number, which can be 2 for dual and 3 for plural. In rare cases, we also find multiplicative functions for the numbers 4 and 9.




2.7 *The spurious functions*

Occasionally we find signs that do not have a clear function. Some can be plausibly attributed to scribal errors. There are also cases however for which a historical explanation can be given. For example, the two signs , representing crossing streets, and , the phonogram for *t*, are often written as one ‘frozen’ group. This makes sense in the writing of the word *njw*t, “town”, which has the (feminine) ending *-t*, with  being a logogram. However, where  is used as determinative with meaning “inhabited area” at the end of a masculine word (not ending on *-t*), we sometimes also find . We then classify  as spurious.

The spurious functions also contribute to creating a robust model. By interpreting some signs as ‘spurious’, the model can complete the analysis of a problematic writing as fall-back option if nothing else works. We return to this matter in Section 5.

2.8 *Combinations of signs having a function*

In the above, we have seen a few instances of a group of signs together having one function, in the case of multiplications of signs and in the case of typographical signs. Another example is , which together represents the logogram *twj*, “the two lands”. The signs in isolation represent two different plants, lily and papyrus, symbolizing Upper and Lower Egypt, respectively.

The group  has a single function as phonogram with sound value *mn*. An isolated  can only be a phonogram *nhbt*. Similarly, the combination of signs  has a single function as a determinative for a “group of people”.

3 SIGN LIST

Essential to the application of our model is an annotated sign list. We have created such a list in the form of a collection of XML files.¹

¹<http://mjn.host.cs.st-andrews.ac.uk/egyptian/unicode/>

Apart from being machine-readable, these files can also be converted to human-readable web pages. Among other things, the files contain knowledge about the various functions of the 1071 signs from the Unicode repertoire, gathered from a number of sources, the foremost of which is Gardiner (1957). The annotated sign list is necessarily imperfect and incomplete, which is due to inadequacies of the Unicode set itself (Rosmorduc 2002/3; Polis and Rosmorduc 2013), as well as to the nature of Ancient Egyptian writing, which gave scribes considerable freedom to use existing signs in new ways and to invent new signs where existing signs seemed inadequate. We have furthermore ignored the origins of signs, and distinguish fewer nuances of sign use than e.g. Schenkel (1971). See Polis and Rosmorduc (2015) for a revised taxonomy of hieroglyphic sign functions, motivated by the goal of compiling sign lists.

The items in our annotated sign list most relevant to this article each consist of:



- a sequence of signs (sometimes multiple sequences of alternative writings),
- a sign function class of that sequence,
- a sequence of letters or a semantic value, depending on the class.

As discussed in Section 2, a sign can often be both a logogram or a determinative specific to a lemma. Similarly, sometimes a sign can be both a phonogram or a phonetic determinative. To avoid duplication, we have created two combined classes. Thus, the sign list distinguishes the following:

- logogram, with the transliteration of a lemma,
- determinative, with a description of meaning,
- logogram / determinative, with the transliteration of a lemma,
- phonogram, with a phonetic value,
- phonetic determinative, with a phonetic value,
- phonogram / phonetic determinative, with a phonetic value,
- typographical, with a description of meaning.

Note that multiplication of signs and spurious signs are not included in the sign list, as these are not properties of the signs themselves but consequences of particular use.

Some signs can be used instead of other signs. This happens in particular where one sign is a graphical variant of another. In order to avoid redundancy, the sign list then only contains a listing of the sign functions for the most representative of two or more graphical variants, plus references from less representative to more representative variants. Such a reference can be automatically expanded into the relevant functions of the most representative sign. Also the two combined classes (logogram / determinative, and phonogram / phonetic determinative) can be split into the individual classes for the purposes of the model of Section 5.

The sign list contains (very rudimentary) information about the morphological structure of the lemmas written by logograms, in particular the stem and the gender (of nouns). The motivation is that this is necessary in order to match sign occurrences to transliterations. For example, the information that the word *nmtt*, “step”, denoted by the logogram , is feminine can be used to infer that uses of the logogram in plural writings should be matched to *nmtwt*, “steps”, with the feminine plural ending *-wt* in place of the feminine singular ending *-t*. Similarly, the logogram , for *hnj*, “to row”, is accompanied by information that its stem is *hn*, so we can identify the use in the writing of *hn=f*, “he rows”, without the weak consonant *j*, which disappears in most inflections.

4

CORPUS



There is currently only one comprehensive corpus of Late Egyptian, which is still under development (Polis *et al.* 2013). Corpora of Middle Egyptian, the object of our study, are scarce however. Moreover, we are not aware of any available corpora of hieroglyphic texts in which each sign is annotated with its function. One attempt in that direction was reported by Hannig (1995, p. XXXV), with the objective to determine the ratios of frequencies of four main classes of signs, using the first 40 lines of the text of Sinuhe.

It follows that in order to train and test our model, we had to create our own annotated corpus.² As yet, it is of modest size, including

²as part of the St Andrews corpus: <http://mjn.host.cs.st-andrews.ac.uk/egyptian/texts/>

just two classical texts, known as The Shipwrecked Sailor (Blackman 1932) and Papyrus Westcar (Blackman 1988). Disregarding damaged parts of the manuscripts, the segmented texts comprise 1004 and 2669 words, respectively.

For the convenience of annotation with sign functions, the texts were linearized, that is, information about horizontal or vertical arrangement of signs was discarded. Whereas the positioning of signs relative to one another can be meaningful, our current models do not make use of this; if necessary in the future, the exact sign positions can be extracted from another tier of annotation.

We normalized the texts by replacing graphical variants, such as  and , by a canonical representative, using machine-readable tables that are part of our sign list (Section 3). We also replaced composite signs by smallest graphemic units. For example, we replaced a single sign consisting of three strokes (typographical sign for plurality or collectivity) by three signs of one stroke each. Motivations for this include convenience and uniformity: in typeset hieroglyphic texts one may prefer to use three separate strokes and fine-tune the distance between them to obtain a suitable appearance.

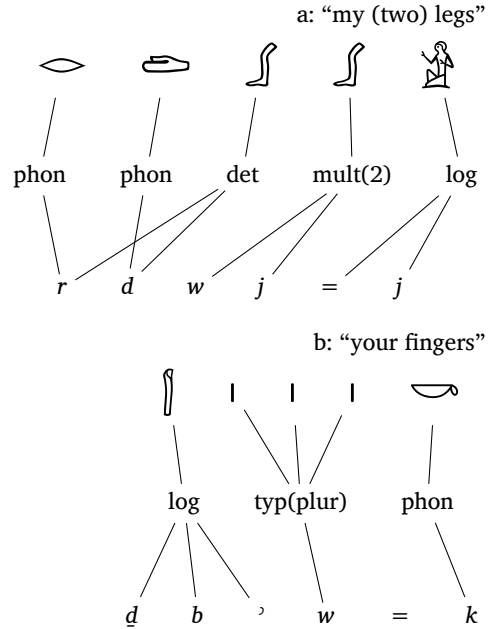
The texts were annotated with functions, using a customized, graphical tool. In this tool one can select known functions for signs, as present in the XML files mentioned in Section 3, but the tool also gives the option to create new functions that are not covered by the sign list. Many such functions were found during annotation.

A peculiar phenomenon in Egyptian writing is *honorific transposition*, which means that a sign or word is written first, even though its linguistic position is further to the end of a word or phrase. This applies in particular to gods and kings. For example, The Shipwrecked Sailor has $dw_3.n=f n=j ntr$, “he thanked the god for me”, with the sign for ntr , “god”, written before the signs for $dw_3.n=f n=j$. Where there is honorific transposition in the corpus spanning more than one word, all these words are put together in one *segment*. Apart from honorific transposition, a segment in the annotated corpus is simply one word.

For each word (or segment), the annotated corpus has:

- the sequence of functions, and
- the sequence of letters of the transliteration.

Figure 2:
Annotations in the corpus
(Shipwrecked Sailor)



The allowable functions are those listed in Section 2. Each function represents one or more signs, which are assumed to occur consecutively. Thereby the sequence of functions specifies the sequence of signs in the hieroglyphic writing. This was made possible by, among other things, our representation of the multiplicative functions (Section 2.6). An example is given in Figure 2a for *rdwj*, “pair of legs”. Whereas the first ‘leg’ sign of the writing is represented by a determinative function, the second such sign is represented by a multiplicative function with value ‘2’, that is, indicating duality.

Depending on their classes, functions may also represent letters, but due to such phenomena as phonetic complements, the sequence of letters of the transliteration is not determined uniquely by the sequence of functions. For this reason, the transliteration is present as separate tier, and functions are linked to the relevant letters of the transliteration, where applicable. In particular, phonograms and phonetic determinatives are linked in this way, and so are logograms and determinatives specific to words.

Also multiplicative functions may be linked to the letters of the dual/plural endings, as exemplified in Figure 2a. The same holds for

a: “he is seen”

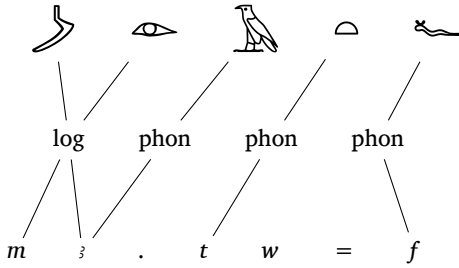
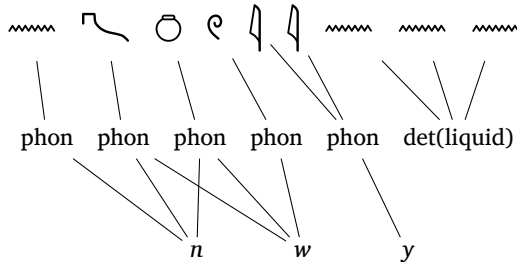


Figure 3:
Further annotations
(Shipwrecked Sailor)

b: “water”



the two or three strokes that indicate duality/plurality. An example is found in Figure 2b, for the plural of *ḏb*, “finger”. Recall that the masculine plural ending is *-w*. Not linked to letters are determinatives that are not specific to any word, as exemplified in Figure 3b.

In the diagrams, the values of the functions are abbreviated or omitted altogether to avoid clutter. For example, we do not explicitly indicate the sound values of phonograms, which usually follow from the links between functions and letters. Also the lemmas of logograms and determinatives specific to words are not shown in the diagrams. Note that these may not be equal to the relevant letters from the transliteration. For example, the lemma of the first function in Figure 3a is in fact *mʒ*, “to see”; the second *ʒ* disappears in some verb forms. Recall that the morpheme *tw* indicates passive; in this writing the *w* is not written out.

Figure 3b is interesting in that it shows two phonetic complements: both the first and the fourth signs are uniliteral phonograms that cover the letters *n* and *w*, which are also covered by the second and third signs, which are both biliteral phonograms.

An essential document while annotating the corpus was the annotation manual, which helped to disambiguate contentious cases, of

which there were many. Examples of such cases were discussed in Section 2.2. We have as far as possible relied on conventional wisdom, but on several occasions we had to resort to informed guesses, making additions to the annotation manual to ensure consistency.

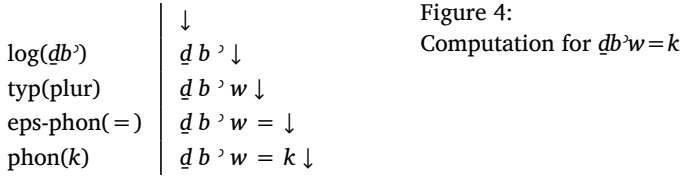
5

MODEL

In order to motivate our model, we investigate Figure 3a. If we string together the letters coming from the respective functions we obtain *mɜ:tf* rather than the correct transliteration *mɜ.tw=f*. Similarly, for Figure 3b we would obtain *nnwnwwy*. There are two causes for this mismatch. The first is that letters can be written more than once, by several functions. In most cases this is done with phonetic complements, that is uniliteral phonograms, but we also find biliteral and trilateral phonograms as well as phonetic determinatives that cover letters already covered before. The second cause is that some letters in the transliteration, often weak consonants, are not represented by any signs at all. For pragmatic reasons, we will treat the Egyptological punctuation symbols, such as the period and the equal sign, on a par with weak consonants not written by signs.

For the second issue, our solution is to introduce an additional type of function, which we call *epsilon-phonogram*. Such a function acts much like a normal phonogram in the sense that a letter is produced in the transliteration, but it does not correspond to any sign (in other words, it corresponds to the empty, or epsilon string of signs).

For the first issue, that of letters covered several times, we conceive of the transliteration as being produced incrementally, in terms of a tape with a head that can move in both directions. In the simplest case, a function appends letters at the end of the tape, and moves the head a corresponding number of places to the right. This suffices for Figure 2b, as shown in Figure 4. The left column indicates the kind of function that is applied, omitting the associated signs, and the right column indicates the tape content, with the arrow marking the position of the head. Initially the tape is empty, and the tape head is at position 0. The logogram function then puts *db'* on the tape, moving the tape head to position 3. Subsequently, the typographical function appends a *w*, moving the head to position 4. After application of an epsilon-phonogram and a phono-



gram function, = and k will have been appended and the head is at position 6.

The situation is only slightly more involved for Figure 2a. Here the determinative specific to rd should only be allowed if rd occurs at the beginning of the tape. This ‘lookback’ amounts to a check of validity of the computation, but it does not alter the fact that the tape is written strictly from left to right, and the tape head always moves rightward.

However, a different approach is needed for cases such as those in Figure 3, which involve phonograms that cover letters more than once, some appending more letters to the tape at the same time. Our solution is to add one more type of function, which we call *jump*. This decrements (or increments) the position of the head, so a string of letters can be written starting from a position other than the end of the tape. The computation for $m_3.tw=f$ is given by Figure 5. Here a jump one position back allows another occurrence of \mathfrak{z} corresponding to a phonogram, after \mathfrak{z} was already seen as part of the logogram. Recall that the second \mathfrak{z} of the lemma $m_3\mathfrak{z}$, “to see”, is omitted in many verb forms. The second feature of ‘ $\log(m_3\mathfrak{z}, m_3)$ ’ in our ad hoc notation attempts to convey that we are dealing with a particular use of this logogram that produces only the letters m_3 in the transliteration. For writing of other words, in which the full, geminated form is present,

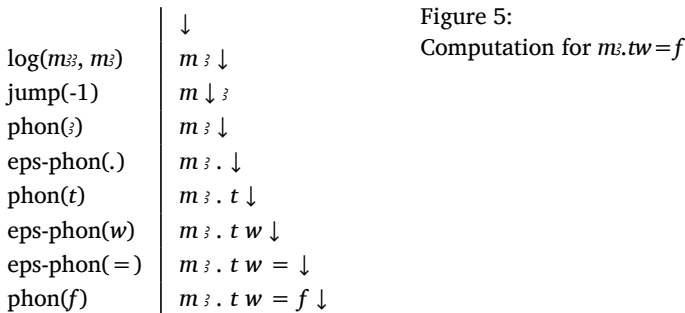


Figure 6:
Computation for *nwy*

	↓
phon(<i>n</i>)	<i>n</i> ↓
jump(-1)	↓ <i>n</i>
phon(<i>nw</i>)	<i>n w</i> ↓
jump(-2)	↓ <i>n w</i>
phon(<i>nw</i>)	<i>n w</i> ↓
jump(-1)	<i>n</i> ↓ <i>w</i>
phon(<i>w</i>)	<i>n w</i> ↓
phon(<i>y</i>)	<i>n w y</i> ↓
det(liquid)	<i>n w y</i> ↓

Figure 7:
Computation for *mnht*

	↓
phon(<i>mn</i>)	<i>m n</i> ↓
jump(-1)	<i>m</i> ↓ <i>n</i>
phon(<i>n</i>)	<i>m n</i> ↓
phon(<i>h</i>)	<i>m n h</i> ↓
phon(<i>t</i>)	<i>m n h t</i> ↓
phondet(<i>mnht</i>)	<i>m n h t</i> ↓
det(abstract)	<i>m n h t</i> ↓
typ(coll)	<i>m n h t</i> ↓

we could use alternatively 'log(m_{B3} , m_{B3})'. We will see more examples of functions having additional features later.

The computation for *nwy* is given by Figure 6. Here several jumps are needed to model that *n* and *w* are each covered by three different signs. Note that the determinative has a general description 'liquid' and so does not correspond to any letters.

The computation for *mnht* is given by Figure 7. As shown, application of a phonetic determinative does not require a jump. This is motivated by the observation that phonetic determinatives behave similarly to determinatives in that they tend to appear at the end of a word, even after phonograms for subsequent letters (cf. Figure 1). A phonetic determinative with a certain sound value is applicable if that value is a substring of the current content of the tape. Application of the function leaves the tape content and position of the head unchanged.

We impose two constraints on the use of jumps. The first is that jumps with positive values, moving the tape head rightward, should

not bring it beyond the end of the (written) tape. This is because the transliteration should be a sequence of letters without any gaps.

The second constraint is that no tape square that already contains a letter can be overwritten with a different letter. This is consistent with the application we are aiming to model, viz. Egyptian writing. This means for example that the first ‘phon(*nw*)’ in Figure 6 is applicable because the tape content to the right of the head, which is *n*, is a prefix of *nw*. Application of the function leaves that existing *n* unaffected and in addition appends the remaining suffix *w* at the end of the tape and moves the head to be after that suffix. In general, if the tape content to the right of the head is β , then we can apply a phonogram with value γ if:

- β is a prefix of γ (as in the case discussed above) or
- γ is a prefix of β (cf. phonogram for *f* in Figure 8a below).

Our aim is to complete the above framework to allow a sequence of functions to uniquely determine a sequence of signs and a sequence of letters. The sequence of signs is straightforwardly obtained as we already assumed from Section 1 onward that each function determines one or more consecutive signs. After having added epsilon-phonograms and jumps, we can now also account for letters not represented by signs, and for letters represented by several signs.

At least one more refinement remains to be explained. A phonogram for *t* or *d* is sometimes used for letters in a word that historically should have \underline{t} or \underline{d} , and vice versa; cf. the discussion in Section 2.2 about historical sound changes in Egyptian. Hence we sometimes need to give a phonogram an additional feature, so that for example ‘phon($\underline{t}, \underline{t}$)’ indicates that *t* is the historical sound value of the sign, say \ominus , but the sign is used in the writing of a word whose transliteration has \underline{t} instead.

After this and other minor refinements, any sequence of functions corresponds to at most one analysis of a word, in terms of a sequence of signs, a sequence of letters, and the links between them, as exemplified in Figures 1–3, or in other words, in terms of the kinds of annotations that exist in our corpus. We also aim to achieve the converse, namely to translate an annotation of a word to a unique sequence of functions. Part of this is straightforward, as most of the functions and the order

in which they occur in a sequence are determined by the order of the signs. However, if there are no further restrictions, jumps may be inserted anywhere, even when they are not useful. In particular, they may be applied just before applying a determinative, even though a determinative does not depend on the input positions. In principle we could even apply a number of jumps in sequence, moving the head back and forth.

We solve this by demanding that jumps only occur just before application of a phonogram, or a related function whose application relies on the input position. The concrete realization is by a flag \mathbf{fl}_{jump} , which is set to **true** after a jump. As long as the flag is **true**, no determinative, phonetic determinative, or another jump is applicable. A phonogram and a few other functions reset the flag to **false**. For similar reasons, we use a flag \mathbf{fl}_{eps} that is set of **true** after application of an epsilon-phonogram. As long as this flag is **true**, no jump is allowed. The effect is that epsilon-phonograms are applied as late as possible. Two more flags, \mathbf{fl}_{jp} and \mathbf{fl}_{end} , will be discussed later.

To make the preceding more precise, we introduce the concept of *configuration*, which contains:

- the tape content preceding the head position, denoted by α ,
- the tape content from the head position onwards, denoted by β ,
- the values of the four flags.

Initially, the tape is empty, so $\alpha = \beta = \varepsilon$, where ε denotes the empty string, and all flags are **false**.

In a given configuration, only a subset of functions is applicable. For example, if $\alpha = \varepsilon$ and $\beta = n$, then a function $\text{phon}(nw)$ would be applicable, but not say a function $\text{phon}(t)$. The flags also restrict the applicable functions, as explained above. In general, every function has a *precondition*, that is, a set of constraints that determines whether it is applicable in a certain configuration, and a *postcondition*, which specifies how its application changes the configuration. The most important functions are characterized in this manner in Table 1, with tape content and position of the head as specified by α and β .

The precondition of a logogram for lemma γ says that γ must occur from the position of the head onward, possibly after a prefix of γ was written already, e.g. using phonograms. Furthermore, the position of the head must be either 0 or 1, and in the latter case,

Table 1: Preconditions and postconditions

Logogram for γ	
Pre	$\alpha = \varepsilon$ or (for causative; see main text) $\alpha = s$, β is prefix of γ , $\mathbf{fl}_{eps} = \mathbf{false}$.
Post	$\alpha := \alpha\gamma$, $\beta := \varepsilon$, $\mathbf{fl}_{jump} := \mathbf{false}$.
Phonogram with sound value γ	
Pre	γ is prefix of β or β is prefix of γ .
Post	$\alpha := \alpha\gamma$, if β was of the form $\gamma\delta$ then $\beta := \delta$ else $\beta := \varepsilon$, $\mathbf{fl}_{jump} := \mathbf{false}$, $\mathbf{fl}_{eps} := \mathbf{false}$.
Determinative not specific to any word	
Pre	$\mathbf{fl}_{jump} = \mathbf{false}$, $\mathbf{fl}_{eps} = \mathbf{false}$.
Post	-
Determinative specific to γ	
Pre	$\alpha\beta = \gamma\delta$ or (for causative) $\alpha\beta = s\gamma\delta$ for some δ , $\mathbf{fl}_{jump} = \mathbf{false}$, if $\mathbf{fl}_{eps} = \mathbf{true}$ then $\delta = \varepsilon$.
Post	$\mathbf{fl}_{eps} := \mathbf{false}$.
Phonetic determinative with sound value γ	
Pre	$\alpha\beta = \delta_1\gamma\delta_2$ for some δ_1 and δ_2 , $\mathbf{fl}_{jump} = \mathbf{false}$, if $\mathbf{fl}_{eps} = \mathbf{true}$ then $\delta_2 = \varepsilon$.
Post	$\mathbf{fl}_{eps} := \mathbf{false}$.
Spurious	
Pre	$\mathbf{fl}_{jump} = \mathbf{false}$, $\mathbf{fl}_{eps} = \mathbf{false}$
Post	-
Jump with value j	
Pre	$\mathbf{fl}_{jump} = \mathbf{false}$, $\mathbf{fl}_{eps} = \mathbf{false}$, $\delta = \alpha\beta$, $i = \alpha $, $0 \leq i + j \leq \delta $.
Post	$\alpha := \alpha'$, $\beta := \beta'$ for some α' and β' such that $\alpha'\beta' = \delta$ and $ \alpha' = i + j$, $\mathbf{fl}_{jump} := \mathbf{true}$.
Epsilon-phonogram for letter ℓ	
Pre	$\beta = \varepsilon$.
Post	$\alpha := \alpha\ell$, $\mathbf{fl}_{eps} := \mathbf{true}$.

the first letter on the tape must be *s*. This is because in Egyptian, the prefix *s-* can be used to derive causative verbs from other verbs. The writing may then consist of a phonogram for *s* followed by a logogram for the original verb. The postcondition for logograms says simply that γ is written to the tape and the head moves rightward by $|\gamma|$ positions.

The precondition of a phonogram with value γ was discussed before. The postcondition is slightly complicated by the need to distinguish between two cases, where γ is a prefix of β or where β is a prefix of γ (if $\gamma = \beta$, the two cases collapse).

The preconditions and postconditions of determinatives not specific to any words are straightforward. For a determinative specific to word γ , we merely need to check whether γ is present near the beginning of the tape, possibly after the causative prefix *s-*; if the previous function was an epsilon-phonogram, then γ must be a suffix of the tape content (recall that we want epsilon-phonograms to be applied as late as possible). Phonetic determinatives are similar, except that the required string γ need not occur near the beginning of the tape.

Spurious functions require that the previously applied function is not a jump or epsilon-phonogram. A jump with value j , which can be positive or negative, is allowed for current position i of the head provided the previously applied function was not a jump or epsilon-phonogram, and provided the new position $i + j$ is not preceding the beginning of the tape nor beyond the end of the tape. An epsilon-phonogram is only allowed if the head is at the end of the tape.

Our model has a number of specialized functions in place of the generic typographical functions as they occur in the corpus. For example, the three strokes, for ‘plurality or collectivity’, in the model correspond to three different functions with different preconditions and postconditions. First, the three strokes may be purely semantic, in the writing of a collective noun in singular form, where they do not represent any letters. This function behaves much like a determinative not specific to any word, except that it can only occur at the end of a word. For this reason, the flag fl_{end} is set to **true**. The purpose of this flag is to prevent that further letters are appended behind the end of the tape, until possibly an Egyptological ‘=’ symbol marks the end of the current word proper, before a suffix pronoun.

The plural strokes may also signify plurality in the grammatical sense, in which case it corresponds to the *-w* ending of masculine plural, or to the *-wt* ending of feminine plural. A separate function is needed for the two genders, both of which set fl_{end} to **true**. Apart from the flag fl_{end} , the function of three strokes for masculine plural has preconditions and postconditions identical to those of the phonograms. The case of feminine plural will be discussed further below.

Similarly, our model distinguishes between three uses of the multiplicative functions with value ‘3’, with different preconditions and postconditions. As in the case of the plural strokes, their meaning may be purely semantic, without a word being grammatically plural, or they may be used as markers of either masculine plural or feminine plural.

In our corpus we have linked functions marking plural only to the *w* from the ending, whether it is the *-w* ending of masculine plural or the *w* that is the first letter of the *-wt* ending of feminine plural. This is because the *t* of the feminine ending would normally be accounted for already by another sign, which could be a phonogram or logogram, as illustrated in Figures 8a and 8b.

a: “beautiful (women)”

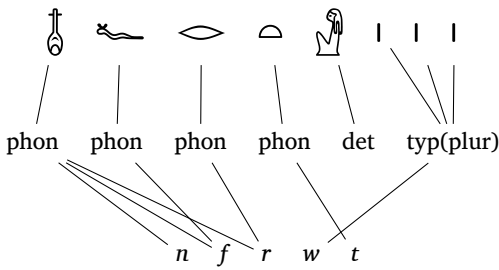
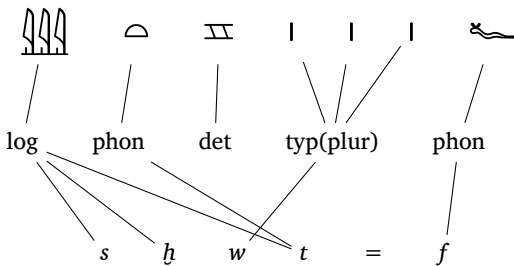


Figure 8:
Annotations of feminine plural words
(Papyrus Westcar)

b: “its fields”



The same two examples also illustrate the challenge that feminine plural poses to a left-to-right automaton model. When the feminine *t* is written to the tape, the function justifying the *w* in front of the *t* is not seen until many steps later. The use of jumps to handle this seems inappropriate, as jumps were designed for phonetic complements. Another potential solution is to use lookahead, but this appears difficult to extend with probabilities.

We have chosen for a different solution, using the flag \mathbf{fl}_{fp} , for ‘feminine plural’. This flag is set to **true** when a feminine plural is predicted by (nondeterministically) putting an extra *w* on the tape, in one of two cases. The first is if a logogram of a feminine word is seen, and the second is if a phonogram for *t* is seen.

The rest of the computation then has the obligation to reset \mathbf{fl}_{fp} to **false**, and this can only happen if a function for plurality (either the three strokes or a multiplicative function with value ‘3’) is seen later. While $\mathbf{fl}_{fp} = \mathbf{true}$, analysis of the current word cannot be completed.

Concretely for Figure 8a, we now have two functions ‘phon(*t,t,false*)’ and ‘phon(*t,wt,true*)’. Both correspond to a phonogram for the letter *t* (the first feature), but realized as *t* or *wt* in the transliteration (the second feature), while possibly predicting feminine plural (the third feature). The first function has the preconditions and postconditions of a normal phonogram (cf. Table 1), while the second writes *wt* on the tape instead of just *t* and sets \mathbf{fl}_{fp} to **true**. The resulting computation is in Figure 9.

Similarly for Figure 8b, we now have two functions ‘log(*sḥt,sḥt,false*)’ and ‘log(*sḥt,sḥwt,true*)’. Both are logograms for the same sign for lemma *sḥt* (first feature) while they are realized differently in the transliteration (second feature), possibly predicting feminine plural (the third feature). The first function behaves like a normal logogram, but the second writes *sḥwt* on the tape and sets \mathbf{fl}_{fp} to **true**. The resulting computation is in Figure 10.

Our model presently has no special provisions for the phenomenon of honorific transposition (Section 4). This implies that accuracy is poor for the (few) cases of honorific transposition in the corpus. To address this, one may consider refinements of the model that allow ‘gaps’ in the hieroglyphic writing to be filled in later, along the lines of the Operation Sequence Model (Durrani *et al.* 2015).

	↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
phon(<i>nfr</i>)	<i>n f r</i> ↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
jump(-2)	<i>n</i> ↓ <i>f r</i>	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
phon(<i>f</i>)	<i>n f</i> ↓ <i>r</i>	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
phon(<i>r</i>)	<i>n f r</i> ↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
phon(<i>t, wt, fp=true</i>)	<i>n f r w t</i> ↓	$\mathbf{fl}_{fp} = \mathbf{true}, \mathbf{fl}_{end} = \mathbf{true}$
det(<i>female</i>)	<i>n f r w t</i> ↓	$\mathbf{fl}_{fp} = \mathbf{true}, \mathbf{fl}_{end} = \mathbf{true}$
typ(<i>plur</i>)	<i>n f r w t</i> ↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{true}$

Figure 9:
Computation for *nfrwt*

	↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
log(<i>shwt, shwt, fp=true</i>)	<i>s h w t</i> ↓	$\mathbf{fl}_{fp} = \mathbf{true}, \mathbf{fl}_{end} = \mathbf{true}$
jump(-1)	<i>s h w</i> ↓ <i>t</i>	$\mathbf{fl}_{fp} = \mathbf{true}, \mathbf{fl}_{end} = \mathbf{true}$
phon(<i>t,t,fp=false</i>)	<i>s h w t</i> ↓	$\mathbf{fl}_{fp} = \mathbf{true}, \mathbf{fl}_{end} = \mathbf{true}$
typ(<i>plur</i>)	<i>s h w t</i> ↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{true}$
phon(=)	<i>s h w t =</i> ↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$
phon(<i>f</i>)	<i>s h w t = f</i> ↓	$\mathbf{fl}_{fp} = \mathbf{false}, \mathbf{fl}_{end} = \mathbf{false}$

Figure 10:
Computation for *shwt=f*

6

PROBABILITIES

After having captured the relation between sequences of signs and sequences of letters solely in terms of sequences of functions, the next step is to estimate their probabilities. An obvious candidate is a simple *N*-gram model:

$$P(f_1^n) = \prod_i P(f_i | f_1^{i-1}) \approx \prod_i P(f_i | f_{i-N+1}^{i-1})$$

Here f_1, \dots, f_n is a sequence of functions, ending in an artificial end-of-word function, and f_i^j is short for f_i, \dots, f_j . In our experiments, estimation of $P(f_i | f_{i-N+1}^{i-1})$ is by relative frequency.

About 4000 functions are compiled out of the entries of the sign list. Added to this are dynamically created functions, such as numbers, epsilon-phonograms and jumps. Because little training material is available, this means a considerable portion of these functions is never observed, and smoothing techniques become essential. We use Katz's back-off (Katz 1987) in combination with Simple Good-Turing (Gale and Sampson 1995).

Functions are naturally divided into a small number of classes, such as the class of all phonograms and the class of all logograms. Using these classes as states, we obtain a second type of model in

terms of (higher-order) HMMs (Rabiner 1989; Vidal *et al.* 2005). For fixed N , and with c_i denoting the class of function f_i , we have:

$$P(f_i | f_{i-N+1}^{i-1}) \approx P(c_i | c_{i-N+1}^{i-1}) * P(f_i | c_i)$$

Estimation of both expressions in the right-hand side is again by relative frequency estimation, in combination with smoothing.

It should be noted that not all sequences of functions correspond to valid writings. Concretely, in the configuration reached after applying functions f_1^{i-1} , the preconditions of function f_i may not hold. As a result, some portion of the probability mass is lost in invalid sequences of functions. We see no straightforward way to avoid this, as the model discussed in Section 5, which allows jumps of the tape head, cannot be captured in terms of finite-state machinery.

7

RESULTS

In our experiments, the training corpus was Papyrus Westcar and the test corpus was The Shipwrecked Sailor. We have considered but rejected the possibility of taking two disjoint parts of both texts together as training and test corpora, for example taking all odd words from both texts for training and all even words for testing. The argument against this is that many words occur repeatedly in the same text, and therefore there would be a disproportionate number of words that occur in both training and test material, potentially leading to skewed results.

Our objective is now to guess the correct sequence of functions, given the sequence of signs and the sequence of letters of a word. We determined recall, precision, and F-measure, averaged over all words in the test corpus. This was done after removing jumps and epsilon-phonograms, so that we could take the annotations from the corpus as gold standard. We have also ignored how functions are linked to letters; the main motivation for this was to be able to define a suitable baseline, as described next.

Among all sequences of functions that correspond to a given sequence of signs, the baseline model yields the one that maximizes the product of the (unigram) probabilities of those functions. Note that a function can correspond to one, two or more signs, so that all relevant partitions of the given sequence of signs need to be considered.

As this ignores the letters altogether, the baseline is independent of the model of Section 5, avoiding the intricacies of preconditions and postconditions.

For a concrete example, consider Figure 2b as gold standard. The ‘relevant’ items are (1) the logogram function of 𓆎 for the lemma \underline{db} , “finger”, tied to the first sign, (2) the typographical function of the three strokes, with meaning ‘plural’ and realized as letter w , tied to the next three signs, and (3) the phonogram function of 𓆏 with sound value k , tied to the last sign. Recall and precision are 100% if ‘retrieved’ are exactly these three items.

We implemented the N -gram models and HMMs from Section 6. An acyclic finite automaton is first created, with states representing configurations together with the last $N - 1$ functions or classes. Transitions are labelled by functions, and have weights that are negative log probabilities determined by the chosen probabilistic model. Most of the functions directly come from the sign list. Other functions are dynamically constructed, on the basis of the input signs, as for example typographical functions representing numbers. Another example is the class of multiplicative functions, which are generated if a pattern of one or more signs occurs two or more times. Final states correspond to configurations reached after processing all the signs of a word, with α equal to the transliteration of that word, $\beta = \varepsilon$, $\text{fl}_{\text{jump}} = \text{false}$ and $\text{fl}_{\text{sp}} = \text{false}$. A final state always exists, in the worst case by analyzing all signs as spurious, and applying one epsilon-phonogram for every letter.

The shortest path from the initial state to a final state is extracted using the shortest-path algorithm of Dijkstra (1959). The labels on this path then give us the list of functions on the basis of which we compute recall and precision.

Results are given in Table 2. It is unsurprising that the models with $N = 1$ improve over the baseline. Although the baseline is also defined in terms of unigram probabilities, it ignores consistency of the sequence of functions relative to the letters. The first-order HMM performs better than the unigram model. This can be attributed to smoothing. For example, the unigram model will assign the same low probability to a spurious function unseen in the training material as to an unseen phonogram, although phonograms overall are far more

Table 2:
Experimental results:
recall, precision, F-measure

	R	P	F1
baseline	86.0	86.0	86.0
<i>N</i> -gram			
<i>N</i> = 1	90.6	90.6	90.6
<i>N</i> = 2	94.4	94.4	94.4
<i>N</i> = 3	94.4	94.4	94.4
HMM			
<i>N</i> = 1	91.4	91.4	91.4
<i>N</i> = 2	91.8	91.8	91.8
<i>N</i> = 3	92.0	92.0	92.0
interpolation of <i>N</i> -gram and HMM			
<i>N</i> = 1	90.5	90.5	90.5
<i>N</i> = 2	94.8	94.8	94.8
<i>N</i> = 3	95.0	94.9	94.9

likely. The first-order HMM however suitably models the low probability of the class of spurious functions.

For *N* greater than 1, the HMMs perform less well than the *N*-gram models. This suggests that the probabilities of functions depend more on the exact identities of the preceding functions than on their classes. The best results are obtained with linear interpolation of the *N*-gram model and the HMM, weighted 9:1, for *N* = 3.

8

CONCLUSION AND OUTLOOK

Our contributions include the design of an annotated corpus of sign use, allowing quantitative study of the writing system, and serving to document rare uses of signs. The second main contribution is a probabilistic model of how signs follow one another to form words. The model is amenable to supervised training. Unsupervised training will be the subject of future investigation.

The probabilistic model is evaluated through computation of the most probable sequence *F* of functions given the sequence *S* of signs and the sequence *L* of letters, or formally $\operatorname{argmax}_F P(F | S, L) = \operatorname{argmax}_F P(F, S, L)$, where $P(F, S, L)$ is the joint model of Section 6. The model could also be the starting point for other tasks, such as automatic transliteration. However, evaluating $\operatorname{argmax}_L P(L | S) = \operatorname{argmax}_L \sum_F P(F, S, L)$, using the same model $P(F, S, L)$ as before, is

not likely to give satisfactory results. This is because, in general, a shorter sequence F tends to have a higher probability than a longer one, and handling of, for example, phonetic complements typically requires longer sequences involving jumps. As a consequence, overly long transliterations will be produced with repeated letters.

The solution we propose is to let the automaton model compute conditional probabilities $P(F, S | L)$, in combination with a prior model $P(L)$. This model would involve a probability distribution over the lengths of stems (most nouns and verbs have stems of two or three letters) and simple forms of morphosyntactic knowledge, including the Egyptological punctuation symbols. In the ideal case it would also include a lexicon. This is yet to be implemented and evaluated.

ACKNOWLEDGMENTS

This work evolved out of intense discussions of the first author with Serge Rosmorduc and François Barthélemy, in the summer of 2012. Gratefully acknowledged are the anonymous referee reports, which provided many insightful and useful suggestions.

REFERENCES

- J.P. ALLEN (2000), *Middle Egyptian: An Introduction to the Language and Culture of Hieroglyphs*, Cambridge University Press.
- F. BARTHÉLEMY and S. ROSMORDUC (2011), Intersection of multitape transducers vs. cascade of binary transducers: the example of Egyptian Hieroglyphs transliteration, in *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*, pp. 74–82, Blois, France.
- S. BILLET-COAT and D. HÉRIN-AIME (1994), A Multi-Agent Architecture for an Evolving Expert System Module, in *Database and Expert Systems Applications, 5th International Conference*, volume 856 of *Lecture Notes in Computer Science*, pp. 581–590, Springer-Verlag, Athens, Greece.
- A.M. BLACKMAN (1932), *Middle-Egyptian Stories – Part I*, Fondation Égyptologique Reine Élisabeth.
- A.M. BLACKMAN (1988), *The Story of King Kheops and the Magicians*, J.V. Books.
- P.F. BROWN, S.A. DELLA PIETRA, V.J. DELLA PIETRA, and R.L. MERCER (1993), The Mathematics of Statistical Machine Translation: Parameter Estimation, *Computational Linguistics*, 18(4):263–311.

- P.T. DANIELS and W. BRIGHT, editors (1996), *The World's Writing Systems*, Oxford University Press, New York.
- C.T. DE VARTAVAN (2010), Snt[r]/sn̄[r] means '[Divine/Godly] Scent', *Advances in Egyptology*, 1:5–17.
- E.W. DIJKSTRA (1959), A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik*, 1:269–271.
- N. DURRANI, H. SCHMID, A. FRASER, P. KOEHN, and H. SCHÜTZE (2015), The Operation Sequence Model — Combining N-Gram-Based and Phrase-Based Statistical Machine Translation, *Computational Linguistics*, 41(2):157–186.
- W.A. GALE and G. SAMPSON (1995), Good-Turing Frequency Estimation Without Tears, *Journal of Quantitative Linguistics*, 2(3):217–237.
- L. GALESCU and J.F. ALLEN (2002), Pronunciation of proper names with a joint n-gram model for bi-directional grapheme-to-phoneme conversion, in *Proceedings of the Seventh International Conference on Spoken Language Processing (ICSLP-2002)*, pp. 109–112, Denver, CO, USA.
- A. GARDINER (1957), *Egyptian Grammar*, Griffith Institute, Ashmolean Museum, Oxford.
- R. HANNIG (1995), *Grosses Handwörterbuch Ägyptisch-Deutsch: die Sprache der Pharaonen (2800-950 v.Chr.)*, Verlag Philipp von Zabern, Mainz.
- G. IFRAH (1981), *Histoire Universelle des Chiffres*, Editions Seghers, Paris.
- S.M. KATZ (1987), Estimation of probabilities from sparse data for the language model component of a speech recognizer, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.
- K. KNIGHT and J. GRAEHL (1998), Machine Transliteration, *Computational Linguistics*, 24(4):599–612.
- A. LOPRIENO (1995), *Ancient Egyptian: a linguistic introduction*, Cambridge University Press.
- M.G.A. MALIK, C. BOITET, and P. BHATTACHARYYA (2008), Hindi Urdu machine transliteration using finite-state transducers, in *The 22nd International Conference on Computational Linguistics*, volume 1, pp. 537–544, Manchester, UK.
- M.-J. NEDERHOF (2008), Automatic Alignment of Hieroglyphs and Transliteration, in N. STRUDWICK, editor, *Information Technology and Egyptology in 2008, Proceedings of the meeting of the Computer Working Group of the International Association of Egyptologists*, pp. 71–92, Gorgias Press.
- S. POLIS, A.-C. HONNAY, and J. WINAND (2013), Building an Annotated Corpus of Late Egyptian, in S. POLIS and J. WINAND, editors, *Texts, Languages & Information Technology in Egyptology*, pp. 25–44, Presses Universitaires de Liège.
- S. POLIS and S. ROSMORDUC (2013), Réviser le codage de l'égyptien ancien. Vers un répertoire partagé des signes hiéroglyphiques, *Document Numérique*, 16(3):45–67.

A probabilistic model of Ancient Egyptian writing

- S. POLIS and S. ROSMORDUC (2015), The Hieroglyphic Sign Functions. Suggestions for a Revised Taxonomy, in H. AMSTUTZ, A. DORN, M. MÜLLER, M. RONSDORF, and S. ULJAS, editors, *Fuzzy Boundaries: Festschrift für Antonio Loprieno*, volume 1, pp. 149–174, Widmaier Verlag, Hamburg.
- L.R. RABINER (1989), A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, 77(2):257–286.
- S. ROSMORDUC (2002/3), Codage informatique des langues anciennes, *Document Numérique*, 6:211–224.
- S. ROSMORDUC (2008), Automated Transliteration of Egyptian Hieroglyphs, in N. STRUDWICK, editor, *Information Technology and Egyptology in 2008, Proceedings of the meeting of the Computer Working Group of the International Association of Egyptologists*, pp. 167–183, Gorgias Press.
- W. SCHENKEL (1971), Zur Struktur der Hieroglyphenschrift, *Mitteilungen des deutschen archäologischen Instituts, Abteilung Kairo*, 27:85–98.
- W. SCHENKEL (1984), *Aus der Arbeit an einer Konkordanz zu den altägyptischen Sargtexten*, volume 12 of *Göttinger Orientforschungen, IV. Reihe*, Harrassowitz.
- R. SPROAT (2000), *A Computational Theory of Writing Systems*, Cambridge University Press, Cambridge.
- A. TSUKAMOTO (1997), Automated Transcription of Egyptian Hieroglyphic Texts: via transliteration using computer, *Journal of the Faculty of Culture and Education*, 2(1):1–40, Saga-University.
- E. VIDAL, F. THOLLARD, C. DE LA HIGUERA, F. CASACUBERTA, and R.C. CARRASCO (2005), Probabilistic Finite-State Machines — Part II, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.

<http://creativecommons.org/licenses/by/3.0/>

