Applications of Diamonded Double Negation

Yli-Jyrä, Anssi

Potsdam University Press,
2008

Yli-Jyrä , A 2008 , Applications of Diamonded Double Negation . in T Hanneforth & K-M Würzner (eds) , Finite-state methods and natural language processing : 6th International Workshop, FSMNLP 2007, Potsdam, Germany, September 14-16, Revised Papers . Potsdam University Press, Potsdam , pp. 6-30 . <
http://books.google.fi/books?id=ZDPgXa2fK4UC&pg=PP1&lpg=PP1&dq=%22Potsdam+University+Press%22+FSMI
>

http://hdl.handle.net/10138/17540

acceptedVersion

Applications of Diamonded Double Negation

Anssi Yli-Jyrä

Department of General Linguistics, University of Helsinki, Finland

Abstract. Nested complementation plays an important role in expressing counter- *i.e.* star-free and first-order definable languages and their hierarchies. In addition, methods that compile phonological rules into finite-state networks use *double-nested complementation* or "double negation". This paper reviews how the double-nested complementation extends to a relatively new operation, *generalized restriction* (GR), coined by the author (Yli-Jyrä and Koskenniemi 2004). This operation encapsulates a double-nested complementation and elimination of a concatenation marker, *diamond*, whose finite occurrences align concatenations in the arguments of the operation. The paper demonstrates that the GR operation has an interesting potential in expressing regular languages, various kinds of grammars, bimorphisms and relations. This motivates a further study of optimized implementation of the operator.

1 Introduction

The goal of this paper[1] is to advocate implementation and optimization of a non-classical regular operation – *generalized restriction.* This operation augments a double-nested complementation in a very useful way.

Algorithms for complementation are not among the most famous operators in finite-state toolkits due to various reasons that that include (i) the need to define the universal language, (ii) the need to determinize the automaton with a possibly exponential construction and (iii) the absence of weights in the resulting automaton.

Complementation has an important role in generalized star-free expressions and hierarchies characterizing star-free languages. *Generalized star-free expressions* consist of finite languages, concatenations and the Boolean operators, including complementation that assumes the universal language. They describe the counter-free *i.e.* star-free languages [1,2]. These languages are also characterized with the *dot-depth hierarchy* [3] and *first order logic* with linear order ($FO[<]$) [4]. It is also known that typical regular string set expressions in language technology can be reduced to star-free expressions [5].

In addition, complementation is an important operation in finite-state morphology, where it is used to compile conditional rules into automata. In the early finite-state accounts of morphology, phonological rules were compiled manually into transducers. Johnson [6] sketched in 1972 how to obtain finite-state transducers (and bimachines) directly from rules. In 1981, Kaplan and Kay presented

[1] The original title of this invited paper was *The Hidden Jewels of Double Negation.*

another approach to compilation of generative phonological rules descriptions [7]. A bit later, rules of the classical Two Level (TWOL) model [8] and its early implementations were at first hand-compiled. In these circumstances, Kaplan discovered the relevance of *double-nested complementation* [7]. According to a recent interview (Kaplan, p.c., 2007), back then his eureka was: "Everything must be [a] double negation!"

Roughly two decades later the author discovered [9] how to combine double-nested complementation with special markers (*diamonds*) that occurred only a specified number of times (typically twice) in hidden strings. This extension is called *generalized restriction* (GR) because it generalizes the semantics of the context restriction rule of the TWOL model. The new operator has many applications in linguistic finite-state formalisms.

By surveying the applications of the operation, the paper argues that the operator is a versatile and practical tool for finite-state grammar construction.

The Structure of the Paper The notational preliminaries are in Section 2. Section 3 contains introduction to implication rules in finite-state phonology and morphology. Section 4 introduces diamonds and generalized restriction. Sections 5, 6, 7, 8, 9 tell about their applications in constraint systems, combinatorial systems, bracketed systems, bimorphisms, and optimality theoretic systems, respectively. Section 10 discusses a measure of locality in nested generalized restriction. A case example of the possible optimizations is elaborated in Section 11 and the paper is concluded by Section 12.

2 Preliminaries

We assume the reader is familiar with classical results on the connection between closure properties of deterministic and non-deterministic automata and those of regular languages. Apart from Section (7.2), all string sets in this paper are regular languages.

Let A_1, A_2 be sets of symbols. Let U and V be languages over A_1. We assume that the reader is familiar with regular languages and the basic regular operations: concatenation UV, intersection $U \cap V$, union $U \cup V$, asymmetric difference $U \backslash V$, complementation $\overline{U}$, Kleene's star U^*, and Kleene's plus U^+. Let $U^0 = U^{\leq 0} = \epsilon$ and let U^k and $U^{\leq k}$, where $k > 0$, denote respectively the languages $UU^{(k-1)}$ and $(\epsilon \cup U)U^{\leq (k-1)}$. The *local A_2-closure* of U is the relation $f_{A_2}{:}A_1^* \to A_1^*$ defined as $f_{A_2}(U) = \{f(a_0)f(a_1)\dots f(a_{m-1}) \mid a_0 a_1 \dots a_{m-1} {\in} U \wedge a_0, a_1, \dots, a_{m-1} {\in} A_1\}$ where $f(a) = a^*$ for every $a {\in} A_2$, and $f(a){=}a$ otherwise. The *elimination* of symbols A_2 in language U is the function $d_{A_2}(U) = f_{A_2}(U) \backslash A_1^* A_2 A_1^*$. If r is a binary relation, its inverse is denoted by r^{-1}.

Notation $A_1{:}A_2$ denotes alphabet $\{a_1{:}a_2 | a_1 \in A_1, a_2 \in A_2\}$. Set Π is called the *total pivot alphabet*. Its every element is a character pair $a{:}b$ and it is closed in such a way that $a{:}a, b{:}b \in \Pi$ for all $a{:}b \in \Pi$. The *diamond alphabet* M contains markers $\diamond_0{:}\diamond_0, \diamond_1{:}\diamond_1, \diamond_2{:}\diamond_2, \dots, \diamond_s{:}\diamond_s$ and it is disjoint from Π. An identity pair $a{:}a \in (\Pi \cup M)$ is often written simply as symbol a.

The null string is denoted by ϵ. We often denote set $\{u\}$, where $u \in A_1^*$, by u. The length of string u is denoted by $|u|$. A sequence $u = a_0{:}b_0 a_1{:}b_1 \dots a_{m-1}{:}b_{m-1} \subseteq (A_1{:}A_2)^*$ is called a *symbol-pair string* and analyzed alternatively as a string pair $(x_1, x_2) = (a_0 a_1 \dots a_{m-1}, b_0 b_1 \dots b_{m-1})$. Pair (x_1, x_2) can be denoted by $x_1{:}x_2$ if $|x_1| = |x_2|$. In such a pair, x_1 is called the *input string* and x_2 is called the *output string.*

Disjoint sets $B_L \subseteq \Pi$ and $B_R \subseteq \Pi$ have the same cardinality and they are called the *left* and the *right bracket alphabets*, respectively. Set B_L contains at least symbols $<_1, <_2, <_{\text{V}}, <_{\text{NP}}, <_{\text{VP}}, <_{\overleftarrow{\text{SUBJ}}}, <_{\overrightarrow{\text{OBJ}}}$, and set B_R contains at least symbols $>_1, >_2, >_{\text{V}}, >_{\text{NP}}, >_{\text{VP}}, >_{\overleftarrow{\text{SUBJ}}}, >_{\overrightarrow{\text{OBJ}}}$. Let $B = B_L \cup B_R$ and $B_i = \{<_i, >_i\}$.

Let $0{:}0 \in \Pi$ be a representative for the empty string ϵ. The *input and output projections* $\pi_1, \pi_2 : \Pi^* \to \Pi^*$ are defined as $\pi_1(X) = \{d_0(x_1){:}d_0(x_1) \mid x_1{:}x_2 {\in} X\}$ and $\pi_2(X) = \{d_0(x_2){:}d_0(x_2) \mid x_1{:}x_2 {\in} X\}$. Let $I = \pi_1(\Pi)$ and $\Sigma = I \backslash B$.

3 Two Historically Important Implication Operators

Variations of production, alternation and constraint rules in linguistics have a close relationship to logical implications. In propositional logic, *material implication* can be expressed with a disjunction with only one negation: $a \to b = (\neg a) \vee b$. A double negation occurs in *de Morgan's law*: $a \vee b = \neg(\neg a \wedge \neg b)$. By combining these equivalences, we obtain a double negation with conjunction:

$$a \to b = \neg((\neg\neg a) \wedge \neg b) = \neg(a \wedge \neg b). \tag{1}$$

3.1 Kaplan's *if-then* Operators

The introduction quoted Ronald Kaplan's spontaneous eureka "everything must be a double negation" (p.c., 2007). His words crystallize the following discovery: Choosing Equation (1) instead of equation $a \to b = (\neg a) \vee b$ is a very useful choice when implications in finite-state phonology are compiled into automata: Equation (1) can be varied by replacing the conjunction with concatenation – a conjunction of a prefix and a juxtaposed suffix in a string.

Equation (1) resembles Kaplan's *if-then* operators [7] that take two argument languages $P, S \subseteq \Pi^*$:

$$\textit{if-P-then-S}(P, S) \overset{\text{def}}{=} \overline{P\overline{S}} = (\Pi^* \backslash P(\Pi^* \backslash S)) \tag{2}$$

$$\textit{if-S-then-P}(P, S) \overset{\text{def}}{=} \overline{\overline{P}S} = (\Pi^* \backslash (\Pi^* \backslash P)S) \tag{3}$$

$$\textit{P-iff-S}(P, S) \overset{\text{def}}{=} \textit{if-P-then-S}(P, S) \cap \textit{if-S-then-P}(P, S). \tag{4}$$

3.2 Compound Context Restriction

Definition Koskenniemi [8] employs in his Two-Level Grammar a constraint rule called *context restriction.* This rule type involves two-way context conditions such as $\#L\_R\#$ whose alternative forms are related by the following equivalences:

$$\ldots\_\ldots \Leftrightarrow \ldots\epsilon\_\epsilon\ldots; \qquad \#\Pi^* L\_\ldots \Leftrightarrow L\_\ldots; \qquad \ldots\_R\Pi^*\# \Leftrightarrow \ldots\_R. \tag{5}$$

If the rule has several two-way contexts, it is called a *compound context restriction* (CCR) [7] and written as

$$X \Rightarrow \#L_1\_R_1\#, \ldots, \#L_n\_R_n\#. \tag{6}$$

Semantics In Rule 6, languages $X, L_i, R_i \subseteq \Pi^*$, for every $i \in \{1, 2, \ldots, n\}$ are called, respectively, the *center*, the *left context* i and the *right context* i. The rule denotes the largest subset of Π^* where every occurrence of factor $x \in X$ splits the whole string into three parts v, x, y in such a way that there is at least one i such that $v \in L_1$ and $y \in R_1$ [7].

The *if-then* operators provide an easy solution to the exact compilation of *simple (i.e. 1-context) context restriction* [7,8]:

$$X \Rightarrow \#L_1\_R_1\# \stackrel{\text{def}}{=} \textit{if-S-then-P}(L_1, X\Pi^*) \cap \textit{if-P-then-S}(\Pi^* X, R_1). \tag{7}$$

Underlying Complexity In contrast to the simple context restriction, CCR is very combinatorial. Since there are n two-sided contexts, there are $2^n - 1$ potential ways in which at least one two-sided context $\#L_i\_R_i\#$ surrounds each occurrence of the factor $x \in X$. Each context can fail to be satisfied on its left, right or both sides, *i.e.* there are 3^n potential ways in which too many of the $2n$ context parts can be missing.

There are $(m + 1)(m + 2)/2$ factors in an m-character string. The center can properly embed to itself (*e.g.* $\mathtt{bcd} \cup \mathtt{c}$), embed to and be aligned with itself (*e.g.* $\mathtt{bc}^+$), or self-overlap (*e.g.* $\mathtt{bc} \cup \mathtt{cd}$). According to the given semantics of CCR, all instances of X must be surrounded by a context. Even if the occurrences of factors $x_1, x_2 \in X$ were embedded or overlapping, their contexts should independently satisfy the context restriction.

Approximate Formulas Traditionally, the correct semantics has only been *approximated* in the previously described implementations of the context restriction operator: Karttunen *et al.* [10] and Kaplan and Kay [7] mark the occurrences of center factors of CCR using a concatenation closure of constrained regions. That does not work, however, if center $X \subseteq \Pi^*$ is not a subset of Π. Grimley-Evans *et al.* [11] mark a contiguous sequence of center factors. The method assumes that the applications of the rule are in consecutive and non-overlapping ranges of positions. A method by Kempe (p.c., see the appendix of [9]) replaces in an auxiliary tape all center occurrences that have a valid context and verifies that there is no occurrence of center that has not been replaced.

Yli-Jyrä and Koskenniemi [9] survey a number of approximate formulas. In fact, there are real examples of rules where all known approximate methods fail [5]. Pasi Tapanainen's program *RuleCompile*, still in use in 1998, produced output that has not been shown incorrect, but it is possible that it is based on a very good approximate algorithm.

4 First-Order Logic and Diamonds

4.1 Context Restriction

With Star-Free Operators A correct formula for two-context context restriction was published independently by two authors: (i) Dale Gerdemann (see the appendix of [9]), and (ii) the current author [5,9]. The first formula was like this:

$$\begin{aligned} X \Rightarrow \#L_1\_\,R_1\#, \#L_2\_\,R_2\# &\overset{\text{def}}{=} \\ &\textit{if-P-then-S}(\Pi^* X, R_1 \cup R_2) \cap \textit{if-S-then-P}(L_1, X(R_1 \backslash R_2)) \;\cap \\ &\textit{if-S-then-P}(L_2, X(R_2 \backslash R_1)) \cap \textit{if-S-then-P}(L_1 \cup L_2, X(R_1 \cap R_2)). \qquad (8) \end{aligned}$$

The second solution [5] handles any number of contexts, but it involves a star-free expression whose size is exponential to n. This motivates the search for a better compilation method.

It is remarkable that these compilation methods for CCR are based on concatenation and Boolean operations, *i.e.* star-free operations. Star-free operations are closely related to finite model-theory and first-order logic in particular. To study this connection in depth, we can assume that languages $X, L_1, R_1, \ldots,$ L_n, R_n are star-free.

With Position Variables In the *first-order logic with linear order*, $FO[<]$, we can interpret formulas over a finite string $w = \langle c_0, c_1, \ldots, c_{m-1}\rangle$. Denote c_i with $w[i]$ and denote substring $c_i c_{i+1} \ldots c_{j-1}$ with $w[i, j]$, where $0{\le}i{\le}j{\le}m$. Variables $i, j, k, \dots \in \mathbb{N}$ specify string positions and they have to be bound in complete $FO[<]$ sentences.

The occurrence of symbol $c \in \Pi$ in position $i \in \mathbb{N}$ is described by predicate $\phi_c(i, i+1)$ that is true if and only if $i < m$ and $w[i] = c$. The *substring* $c_b c_{b+1} \ldots c_{e-1} \in \Pi^*$ is described by formula $\phi_v(b, e) = \phi_{c_b}(b, b{+}1) \wedge \phi_{c_{b+1}}(b{+}1, b{+}2) \wedge \cdots \wedge \phi_{c_{e-1}}(e{-}1, e)$. The *universal language* over alphabet Π is described by formula $\phi_{\Pi^*}(b, e) = (\forall b{\le}i{\le}e{-}1) \vee_{c\in\Pi} \phi_c(i, i+1)$. Let U and V be languages over Π. The *concatenation* of languages U and V is described by formula $\phi_{UV}(b, e){=}\exists i(\phi_U(b, i) \wedge \phi_V(i, e))$. The *union* of languages U and V is described by formula $\phi_{U\cup V}(b, e){=}\phi_U(b, e) \vee \phi_V(b, e)$. The *intersection* of sets U and V is described by formula $\phi_{U\cup V}(b, e){=}\phi_U(b, e) \wedge \phi_V(b, e)$. The *asymmetric difference* $U\backslash V$ is described by formula $\phi_{U\backslash V}(b, e){=}\phi_U(b, e) \wedge \neg\phi_V(b, e)$.

For all star-free expressions p over alphabet Π, a corresponding $FO[<]$ expression can be constructed by collecting it recursively from the expression tree of p. String w matches the expression p if sentence $\phi_p(0, |w|)$ is true.

If $\phi_X(b, e), \phi_{L_1}(b, e), \phi_{R_1}(b, e), \ldots \phi_{L_n}(b, e), \phi_{R_n}(b, e)$ are formulas describing star-free languages $X, L_1, R_1, \ldots, L_n, R_n$, then Rule 6 is described as set

$$\{w{\in}\Pi^* \mid \neg\exists 0{\le}i{\le}j{\le}|w|[\phi_X(i, j) \wedge \neg \vee_{i=1}^{n} (\phi_{L_i}(0, i) \wedge \phi_{R_i}(j, |w|))]\}. \qquad (9)$$

With Range Variables The signature of $FO[<]$ can be extended with position range variables $\mathbf{v}, \mathbf{x}, \mathbf{y}, \dots \in \mathbb{N} \times \mathbb{N}$ each of which is a pair of type $[b, e]$ where $b \leq e$. The variables denote substrings $w[b_\mathbf{v}, e_\mathbf{v}], w[b_\mathbf{x}, e_\mathbf{x}], w[b_\mathbf{y}, e_\mathbf{y}], \dots$. Expression $\mathbf{x} {\in} X$ denotes formula $\phi_X(b_\mathbf{x}, e_\mathbf{x})$. If $e_\mathbf{v}{=}b_\mathbf{x}$, the concatenation of substrings denoted by ranges $\mathbf{v}$ and $\mathbf{x}$ is denoted by range $[b_\mathbf{v}, e_\mathbf{x}]$ and expressed as $\mathbf{vx}$. Equivalence $\mathbf{v}{=}\mathbf{x}$ denotes formula $b_\mathbf{v}{=}b_\mathbf{x} \wedge e_\mathbf{v}{=}e_\mathbf{x}$. The language of expression 6 is described as set

$$\{w{\in}\Pi^* \mid \neg\exists \mathbf{v}, \mathbf{x}, \mathbf{y}[\mathbf{w}{=}\mathbf{vxy} \wedge \mathbf{x}{\in}X \wedge \neg \vee_{i=1}^{n} (\mathbf{v}{\in}L_i \wedge \mathbf{y}{\in}R_i)]\}. \qquad (10)$$

With MSO Logic The logic $FO[<]$ captures only star-free languages. In order to handle all regular operands, we should extend the logic with monadic second-order (MSO) quantifiers [12,13]. In contrast to Vaillette [14] who makes extensively use of MSO when describing various rules we could maintain the structure of the first-order formula (10) and employ MSO quantifiers only the internal structure of subformulas $\phi_X(b, e)$, $\phi_{L_1}(b, e), \phi_{R_1}(b, e), \dots \phi_{L_n}(b, e), \phi_{R_n}(b, e)$.

4.2 Diamond

As an alternative to an MSO-based semantics, we can use the closure properties of regular languages and transform Formula 10 into regular operations. Like MSO logic, a method based on regular operators is not limited to the case where the operators are star-free.

Our *ad hoc* transformation is based on an encoding that eliminates individual range variables. We observe that inside the scope of quantification $\exists \mathbf{v}, \mathbf{x}, \mathbf{y}$, the quantified variables are restricted by condition $\mathbf{w}{=}\mathbf{vxy}$. In other words, string w is a concatenation of three substrings $v = w[b_\mathbf{v}, e_\mathbf{v}]$, $x = w[b_\mathbf{x}, e_\mathbf{x}]$, and $y = w[b_\mathbf{y}, e_\mathbf{y}]$.

To indicate the parts of the string $w \in \Pi^*$, we use *marked concatenation* $v{\diamond_1}x{\diamond_1}y$ where $\diamond_1 \notin \Pi$. A membership test $x \in X$ will be implemented as condition $v{\diamond_1}x{\diamond_1}y \in W$ where $W = \Pi^*{\diamond_1}X{\diamond_1}\Pi^*$. Similarly, the condition $\mathbf{v}{\in}L_i \wedge \mathbf{y}{\in}R_i$ corresponds now to condition $v{\diamond_1}x{\diamond_1}y \in W_i'$ where $W_i' = L_i{\diamond_1}\Pi^*{\diamond_1}R_i$. Subformula $\phi(\mathbf{v}, \mathbf{x}, \mathbf{y}) = \mathbf{x}{\in}X \wedge \neg \vee_{i=1}^{n} (\mathbf{v}{\in}L_i \wedge \mathbf{y}{\in}R_i)$ thus becomes $v{\diamond_1}x{\diamond_1}y \in C$ where $C = W \setminus \cup_{i=1}^{n} W_i'$.

To obtain concatenation as usual, we just eliminate the markers from marked concatenations: $vxy = d_M(v{\diamond_1}x{\diamond_1}y)$. This corresponds to quantifier elimination. Accordingly, $d_M(C)$ denotes the set $\{w \in \Pi^* \mid \exists \mathbf{v}, \mathbf{x}, \mathbf{y}(\mathbf{w}{=}\mathbf{vxy} \wedge \phi(\mathbf{v}, \mathbf{x}, \mathbf{y}))\}$. It is now easy to see that Formula (10) corresponds to regular expression

$$\Pi^* \setminus d_M(\Pi^*{\diamond_1}X{\diamond_1}\Pi^* \setminus \cup_{i=1}^{n} L_i{\diamond_1}\Pi^*{\diamond_1}R_i). \qquad (11)$$

The most obvious advantage of (11) is its linear size according to the number of contexts. Instead of a Boolean combination of 2^n "double negations" as in (8), Formula (11) contains only one "double negation".

Expression (11) uses a marker symbol $\diamond_1$ that we call a *diamond*. If fact, we assume a marker alphabet M that contains $\diamond_1$ and other diamonds. Diamonds differ essentially from auxiliary markers used by Kaplan and Kay [7]: the number of their occurrences in strings of C is constrained by an integer k. They also bear resemblance to pebbles in Ehrenfeucht-Fraisse games.[2]

4.3 Generalized Restriction

Marked concatenations constitute the foundation for *generalized restriction operator* (GR), a new operation coined by the current author [9]. We define the GR operator now in a manner that is slightly more general in comparison to the original definition. The GR operator $\overset{\Pi,k,M}{\Rightarrow}: 2^{(\Pi^* M \Pi^*)^{\leq k}} \times 2^{(\Pi^* M \Pi^*)^*} \to 2^{\Pi^*}$ is defined by the equation

$$W \overset{\Pi,k,M}{\Rightarrow} W' \overset{\text{def}}{=} \Pi^* \backslash d_M(W \backslash W'). \tag{12}$$

Languages W and W' are called in [15] the *generalized precondition* and the *generalized postcondition*, respectively. When $M = \{\diamond\}$ and $W, W' \subseteq (\Pi^* M \Pi^*)^k$ for some k, the definition is essentially the same as originally.

First-Order Definability A regular expression describes a star-free language if (but not only if) its generalized star-height [16] is zero. The star-height is increased by such operators as Kleene's star. The universal language Π^* is only a short-hand for $\overline{\emptyset}$ and is therefore a neutral element for star-height. Compound context restriction preserves the star-height in regular expressions [5]. It is interesting to see whether the same property holds for generalized restriction.

Assume that the arguments of the GR operator are star-free. To prove that the operator does not increase the star-height, we first split its operands according to the number of diamonds:

$$\begin{gathered} [W \overset{\Pi,k,M}{\Rightarrow} W'] \;=\; \cap_{i=0}^{k} \, [W_i \overset{\Pi,k,M}{\Rightarrow} W'_i] \\ \text{where } W_i = W \cap (\Pi^* M \Pi^*)^i \text{ and } W'_i = W' \cap (\Pi^* M \Pi^*)^i. \end{gathered} \tag{13}$$

While it is possible that strings of W' contain more than k diamonds, it does not matter, because such strings does not affect the difference $W \backslash W'$. It now suffices to show that each sub-GR $W_i \overset{\Pi,k,M}{\Rightarrow} W'_i$ preserves the generalized star-height. For this purpose, we split each sub-GR according to the diamond types used:

$$\begin{gathered} [W_i \overset{\Pi,k,M}{\Rightarrow} W'_i] \;=\; \cap_{c_1 \in M} \cdots \cap_{c_k \in M} \; [U \overset{\Pi,k,M}{\Rightarrow} U'] \text{ where} \\ U{=}W_i{\cap}(\Pi^* c_i \Pi^* c_{i+1} \ldots \Pi^* c_k \Pi^*); U'{=}W'_i{\cap}(\Pi^* c_i \Pi^* c_{i+1} \ldots \Pi^* c_k \Pi^*). \end{gathered} \tag{14}$$

Because U and U' are obtained respectively from star-free languages W and W' by star-free operations, they are also star-free. Again, it suffices to show

[2] Recently, Måns Hulden (p.c., 2008) has elaborated this marker-variable connection and added the likeness of predicate logic to regular expressions using named diamond-like markers.

that every sub-GR $U \overset{\Pi,k,M}{\Rightarrow} U'$ in this decomposition preserves the generalized star-height. This time, we assume there are finite decompositions of U and U':

$$U \overset{\Pi,k,M}{\Rightarrow} U' = [U_1 \cup \dots U_r \overset{\Pi,k,M}{\Rightarrow} U_{r+1} \cup \dots U_p] \text{ where every } U_i \text{ is of the form } U_i = X_{i,0}c_1X_{i,1}c_1 \dots c_kX_{i,k} \text{ in which } c_1, c_2, \dots, c_k \in M. \quad (15)$$

In fact, there exists a decomposition like in Formula (15) because the strongly connected components in the automata recognizing U and U' do not contain diamond transitions and it is therefore easy to extract subautomata between the diamonds. All subautomata of the automata recognizing U and U' are counter-free because languages U and U' are star-free [2]. Accordingly, each component $X_{i,j}$ corresponds to a counter-free subautomaton and is, thus, star-free and definable by an $FO[<]$ formula $\phi_{X_{i,j}}(b, e)$.

Now, as we have split the original GR into sub-GRs, it suffices to show that every sub-GR $[U_1 \cup \dots U_r \overset{\Pi,k,M}{\Rightarrow} U_{r+1} \cup \dots U'_p]$ is definable in $FO[<]$. This holds because we have the equation

$$[U_1 \cup \dots U_r \overset{\Pi,k,M}{\Rightarrow} U_{r+1} \cup \dots U_p] = \{w{\in}\Pi^* \mid \neg\exists \mathbf{x}_0, \mathbf{x}_1, \dots \mathbf{x}_k[\mathbf{w}{=}\mathbf{x}_1\mathbf{x}_2 \dots \mathbf{x}_k \wedge (\vee_{i=1}^{r} \wedge_{j=0}^{k} \mathbf{x}_j{\in}X_{i,j}) \wedge \neg(\vee_{i=r+1}^{p} \wedge_{j=0}^{k} \mathbf{x}_j{\in}X_{i,j})]\}. \quad (16)$$

To conclude, the GR operation preserves the generalized star-height when the height of all its arguments is zero.

Problem 1. Is it possible to get a shorter proof by proving that deletion $d_M : (\Pi^*M\Pi^*)^* \to \Pi^*$ preserves the generalized star height when its domain is restricted to $(\Pi^*M\Pi^*)^k$?

5 Application: Constraint Systems

Often finite-state grammars and rule compilation methods involve constraint relations that are either intersected or composed with other regular relations. The GR operator can be used to construct constraint languages. In addition, it is possible to combine conjunctive constraints so that only one GR operator is needed, or decompose the operator into a conjunction of layers in order to reduce the total count of states in automata. In this section, we will review these three uses of the operator.

5.1 Simple Constraints

Let $X \subseteq \Pi^*$ be a language that acts as a *constraint* such as in finite-state intersection grammar [17]. A constraint language X can be turned into its complement through the equation $\overline{X} = [X \overset{\Pi,0,M}{\Rightarrow} \emptyset]$.

Let $X \subseteq \Pi^*$ be a local grammar that describes forbidden patterns [18]. It can be compiled into as a constraint language *nowhere*$(X) \overset{\text{def}}{=} [\Pi^*X\Pi^* \overset{\Pi,0,M}{\Rightarrow} \emptyset]$.

The *center prohibition* rule [15] (denoted by `/<=`) bears some similarity to the *nowhere* operation and is defined by the equation

$$[X\,\texttt{/<=}\#L_1\_R_1\#,\ldots,\#L_n\_R_n\#] \overset{\text{def}}{=} [\cup_{i=1}^{n} L_i\diamond_0 X\diamond_0 R_i \overset{\Pi,2,M}{\Rightarrow} \emptyset]. \tag{17}$$

Yli-Jyrä and Koskenniemi [9] document how Kaplan's *if-then* operators are reduced to generalized restrictions: *if-P-then-S*$(P,S) = [P\diamond_1\Pi^* \overset{\Pi,1,M}{\Rightarrow} \Pi^*\diamond_1 S]$; *if-S-then-P*$(P,S) = [\Pi^*\diamond_1 S \overset{\Pi,1,M}{\Rightarrow} P\diamond_1\Pi^*]$; *P-iff-S*$(P,S) = [(P\diamond_1\Pi^* \cup \Pi^*\diamond_1 S) \overset{\Pi,1,M}{\Rightarrow} P\diamond_1 S]$. Generalized restriction captures also *compound context restriction* (denoted by `=>`) and *coercion* (denoted here by `<<=` although [9] used `<=`), through the equations

$$[X\texttt{=>}\#L_1\_R_1\#,\ldots\#L_n\_R_n\#] \overset{\text{def}}{=} [\Pi^*\diamond_1 X\diamond_1\Pi^* \overset{\Pi,2,M}{\Rightarrow} \cup_{i=1}^{n} L_i\diamond_1\Pi^*\diamond_1 R_i]; \tag{18}$$

$$[X\texttt{<<=}\#L_1\_R_1\#,\ldots,\#L_n\_R_n\#] \overset{\text{def}}{=} [\cup_{i=1}^{n} L_i\diamond_1\Pi^*\diamond_1 R_i \overset{\Pi,2,M}{\Rightarrow} \Pi^*\diamond_1 X\diamond_1\Pi^*]. \tag{19}$$

Generalized Two-Level Grammar (GTWOL) [15] introduces *center presence requirement* (denoted by `<==`, actually the same as *coercion with (additional) preconditions* [9]), and includes *presence requirement* (denoted by `==>`) that provides a direct interface to the 2-diamond GRs. These are defined by

$$[X\texttt{<==}C] \overset{\text{def}}{=} [C \overset{\Pi,2,M}{\Rightarrow} \Pi^*\diamond_1 X\diamond_1\Pi^*] \qquad \text{where } C\subseteq\Pi^*\diamond_1\Pi^*\diamond_1\Pi^*; \tag{20}$$

$$[C\texttt{==>}C'] \overset{\text{def}}{=} [C \overset{\Pi,2,M}{\Rightarrow} C'] \qquad \text{where } C,C'\subseteq\Pi^*\diamond_1\Pi^*\diamond_1\Pi^*. \tag{21}$$

Two-level Grammar [8] contains an operation called *surface coercion* (traditionally denoted by `<=`). It is defined by the equation

$$[X\texttt{<=}\#L_1\_R_1\#,...,\#L_n\_R_n\#] \overset{\text{def}}{=} [X\texttt{<==}(\cup_i L_i\diamond_1\pi_1^{-1}(\pi_1(X))\diamond_1 R_i)]. \tag{22}$$

5.2 Decomposing into Conjunctive Constraints

It is not always practical to evaluate a GR operation $W \overset{\Pi,k,M}{\Rightarrow} W'$ as a whole. It may sometimes be better to decompose the operands W using additional preconditions [9]. For this purpose, we need a number of *layer preconditions* $P_1, P_2, \ldots, P_m \in (\Pi^* M \Pi^*)^*$ that are defined in such a way that $W \subseteq \cup_{i=1}^{m}(P_i \cap W)$. The decomposition corresponds then to intersection

$$\cap_{i=1}^{m}(W \cap P_i \overset{\Pi,k,M}{\Rightarrow} W'). \tag{23}$$

5.3 Systems of Conjunctive Constraints

In contrast to decompositions, We can also reduce intersection of two GRs into one GR. The possible scenarios include the equations

$$[W_1 \overset{\Pi,k,M}{\Rightarrow} W_1'] \cap [W_2 \overset{\Pi,k,M}{\Rightarrow} W_2'] = [(W_1 \cup W_2) \overset{\Pi,k,M}{\Rightarrow} (W_1' \cup W_2')]$$
$$\text{when } W_1 \cap W_2' \subseteq W_1' \text{ and } W_2 \cap W_1' \subseteq W_2'; \tag{24}$$

$$[W_1 \overset{\Pi,k,M}{\Rightarrow} W_1'] \mathbin{\cap\!\!\!\!+} [W_2 \overset{\Pi,k,M}{\Rightarrow} W_2'] \overset{\text{def}}{=} [(W_1 \cup W_2) \overset{\Pi,k,M}{\Rightarrow} ((W_1 \cap W_1') \cup (W_2 \cap W_2'))]. \tag{25}$$

In (25), the new operator $\mathbin{\cap\!\!\!\!+}$ is called *coherent intersection* since it resolves various implication conflicts between two conjunctive generalized restrictions. In the following, we illustrate the applications of the conjunctive GR operation in (24) with three examples.

Enforcing Balanced Structure A new operator *match-L-R*(L, D, R) is defined by the equation

$$\textit{match-L-R}(L, D, R) \overset{\text{def}}{=} [L\diamond_1\Pi^* \overset{\Pi,1,\diamond_1}{\Rightarrow} L\diamond_1 DR] \cap [\Pi^*\diamond_2 R \overset{\Pi,1,\diamond_2}{\Rightarrow} LD\diamond_2 R]. \tag{26}$$

According to the operator, a left side L (or right side R) must always be paired with a right side R (or left side L), and separated from that with a string that belongs to D. The operator is useful in enforcing balanced structures such as bracketing. Thanks to Equation (24), the operator can also be defined using only one GR operator as the equation

$$\textit{match-L-R}(L, D, R) = [(L\diamond_1\Pi^* \cup \Pi^*\diamond_2 R) \overset{\Pi,1,M}{\Rightarrow} (L\diamond_1 DR \cup LD\diamond_2 R)]. \tag{27}$$

Moreover, a feasible superset of $L\diamond_1 DR \cup LD\diamond_2 R$ can be obtained from $L\diamond_1 D\diamond_2 R$ with the local closure operator f_M that acts as a "metarule" in the equation

$$\textit{match-L-R}(L, D, R) = [(L\diamond_1\Pi^* \cup \Pi^*\diamond_2 R) \overset{\Pi,1,M}{\Rightarrow} f_M(L\diamond_1 D\diamond_2 R)]. \tag{28}$$

Double Arrow Rules The double arrow operator (context restriction plus surface coercion) in Two-Level Grammar [8] is a conjunction of two simpler rules. Each such rule reduces to a single GR operation defined by the equation

$$\begin{gathered}[][X\texttt{<=>}\#L_1\_R_1\#, \ldots, \#L_n\_R_n\#] \overset{\text{def}}{=} [C \overset{\Pi,2,M}{\Rightarrow} \cup_{i=1}^n L_i\diamond_1 X\diamond_1 R_i] \\ \text{where } C = (\cup_{i=1}^n L_i\diamond_1 \pi_1^{-1}(\pi_1(X))\diamond_1 R_i) \cup (\Pi^*\diamond_1 X\diamond_1 \Pi^*).\end{gathered} \tag{29}$$

Coarsely Interpreted Two-Level Grammar Yli-Jyrä and Koskenniemi [15] compile rules of Generalized Two-level Grammar like in Formulas (17), (18), (20), (21), (22), and (29). All rule types reduce to a common form $W_i \overset{\Pi,2,M}{\Rightarrow} W_i'$.

If the rules are loose enough to avoid mutual conflicts, they can be compiled easily in separation. The semantics of the whole grammar is traditionally obtained as an intersection of the individual rules [8], provided that a compound context restriction is counted as a single rule.

In theory, it is also possible to compile all rules at once as if their intersection were computed [15]. The uniform rules are combined straightforwardly on the basis of (24). One possibility to do this is presented in the equation

$$\cap_i(W_i \overset{\Pi,2,M}{\Rightarrow} W_i') = [(\cup_i\diamond_i W_i) \overset{\Pi,3,M}{\Rightarrow} \cup_i\diamond_i W_i']. \tag{30}$$

6 Application: Combinatorial Systems

We now turn our attention from the constraining power of the GR operator to its ability to *generate* a language through its second operand. If the first operand of the GR operator is the universal language, the second operand specifies strings that remain. In comparison, if a logical formula ϕ_1 is a tautology, the truth value of the material implication $\phi_1 \rightarrow \phi_2$ depends completely on the right-hand side ϕ_2. Accordingly, any language $X \subseteq \Pi^*$ can be passed through the GR operation unchanged as follows: $X = [\Pi^* \overset{\Pi,0,M}{\Rightarrow} X]$.

6.1 String Coverings with a Lexicon

The things get very interesting when we modify the first operand by adding into it a diamond that occurs before an *arbitrary* character position. This changes a lot: a string in the second operand would now be passed through the GR operation only if its *every* character is disjunctively preceded by a hidden diamond:

$$X=[\Pi^*\nu'_{1,1}(\Pi)\Pi^* \overset{\Pi,1,M}{\Rightarrow} \nu'_{\star,1}(X)] \text{ where } \nu'_{1,j}(W)=d^{-1}_{\{\diamond_1,\ldots,\diamond_j\}}(W)\cap(\epsilon\cup(\Pi^*M)\Pi^+) \quad (31)$$
$$\text{and } \nu'_{\star,j}(W)=d^{-1}_{\{\diamond_1,\ldots,\diamond_j\}}(W)\cap(\epsilon\cup(\Pi^*M)^*\Pi^+).$$

The same effect is captured by adding two diamonds that surround each character and empty string on both operand languages:

$$X=[\Pi^*\nu_{2,1}(\Pi)\Pi^* \overset{\Pi,2,M}{\Rightarrow} \nu_{\star,1}(X)] \text{ where } \nu_{2,j}(W)=d^{-1}_{\{\diamond_1,\ldots,\diamond_j\}}(W) \cap (\Pi^*\diamond_j\Pi^*\diamond_j\Pi^*)$$
$$\text{and } \nu_{\star,j}(W)=d^{-1}_{\{\diamond_1,\ldots,\diamond_j\}}(W). \quad (32)$$

Free Contexts We can now elaborate the right-hand side of the GR and add there left and right contexts $\#\Pi^*\_\Pi^*\#$:

$$X' = [\Pi^*\nu'_{1,1}(\Pi)\Pi^* \overset{\Pi,1,M}{\Rightarrow} \Pi^*\nu'_{\star,1}(X)\Pi^*] = [\Pi^*\nu_{2,1}(\Pi)\Pi^* \overset{\Pi,2,M}{\Rightarrow} \Pi^*\nu_{\star,1}(X)\Pi^*]. \quad (33)$$

Every string in the result X' will now have a *string covering* that consists of possibly overlapping factors X. For example, if $X = \{$`autom`, `mate`, `eria`$\}$ then X' contains such strings as `automate`, `materia` and `automateriautom`. Accordingly, we have defined a simple combinatorial system. If we want, we can concatenate a unique *sentinel* symbol $\sigma \in \Pi$ to the end of the lexicon $X \subseteq (\Pi \backslash \sigma)^*$ in order to generate $X' = (X\sigma)^*$ *i.e.* a set of strings covered with non-overlapping factors taken from set $X\sigma$.

Problem 2. Can coverings be used to describe allomorph selection, nonconcatenative morphotactics, interdigitation and multi-component rewriting?

Problem 3. Can $\nu'_{\star,j}$ or $\nu_{\star,j}$ be used on the left hand side of an extended GR operator that would still preserve star-freeness? How the change interacts with star-freeness, automata size and applications?

6.2 Optional Changes

Rules as Permissions The string coverings help us to understand GTWOL [15]. In GTWOL, any CCR rule is equivalent to a coherent intersection of simple context restrictions. The set of all n context restrictions are combined under the coherent intersection operator, which corresponds to automatic right-arrow conflict resolution [19,20]. This interpretation is reflected by the equation

$$[X_1\texttt{=>}\#L_1\_R_1\#] \Cap \ldots \Cap [X_n\texttt{=>}\#L_n\_R_n\#] = [W_1 \overset{\Pi,2,M}{\Rightarrow} S_1] \qquad (34)$$
$$\text{where } W_1{=}\Pi^*\nu'_{1,1}(\Pi)\Pi^* \text{ and } S_1{=}\cup_{i=1}^{n} L_i\nu'_{\star,1}(X_i)R_i.$$

Default Correspondences GTWOL [15] assumes that the *identity pairs* in the string covering do not need permissions. This is now captured in every GTWOL by a default context restriction rule $[I^*\texttt{=>}\_]$ that permits substrings consisting of identity pairs I to occur unconditionally. This rule corresponds to default correspondence pairs in the classical TWOL [8].

Multi-Character Changes In context restriction rules of GTWOL, center X can contain strings longer than one character. These multi-character changes can be described also in the classical TWOL through a combined effect of several rules. Regardless of the description, multi-character changes introduce a so-called *embedded-center conflict.* The conflict is more difficult to detect in TWOL that uses several partial rules.

Consider GTWOL rules [`a:i => m:m _`] and [`a:i b:j c:k => l:l _ r:r`]. In a conjunctive system, the first rule would reject string `l:l a:i b:j c:k r:r` although it is accepted by the second, more specific rule. Meanwhile, the second one would reject string `m:m a:i b:j c:k m:m` although it is accepted by the first one. Because the second rule is more specific, this violates expectations based on the *principle of longest application* [15]. To observe this principle, (35) includes the constraining aspects of context restriction rules. Every application of the rule involves a position range whose indication requires at least two diamonds. The relation $\nu_{\star,j}(W)$ is used to produce all entailed (shorter) rules when we redefine

$$W_1 = \Pi^*\nu_{2,1}(\Pi \cup \cup_{i=1}^{n} X_i)\Pi^*;\ S_1 = \cup_{i=1}^{n} L_i\, \nu_{\star,1}(X_i)\, R_i. \qquad (35)$$

6.3 Coercions

A change such as `(reduce+ation):(reduc0O0tion)` requires three double-arrow rules in the classical Two-Level Grammar [8,19]. Black *et al.* [21] make the point that these rules *depend on each other* and if one is missing, the failure caused by the *broken interaction* may not be easy to recognize. In GTWOL, the support for multi-character centers considerably alleviates the danger of broken interaction.

In GTWOL, an *embedded-center conflict* occurs between surface coercion rules [`a:o <= m:m _`] and [`a:i b:j c:k <= _ r:r`]. The first rule would reject string `m:m a:i b:j c:k r:r` but the second would accept it. This conflict is solved again through the principle of the longest application. Yet GTWOL does not automatically collect rules that interact as parts of long changes, if such changes are described with multiple rules.

All multi-context surface coercions can be split into rules that have only one context, because such rules have the common X that is the result of the coercion. The set of p simple surface coercions are again combined under coherent intersection. The semantics of a set of surface coercions is, thus, defined by

$$[X_1 \texttt{<=} \# L_1 \_ R_1 \#] \mathbin{\boxplus} \ldots \mathbin{\boxplus} [X_p \texttt{<=} \# L_p \_ R_p \#] \overset{\text{def}}{=} [W_2 \overset{\Pi,2,M}{\Rightarrow} S_2]$$
$$\text{where } W_2 = \cup_{i=1}^{p} L_i \, \nu_{2,2}(\pi_1^{-1}(\pi_1(X_i))) \, R_i; \quad S_2 = \cup_{i=1}^{p} L_i \, \nu_{\star,2}(X_i) \, R_i. \tag{36}$$

Partial-overlap conflicts [22] are difficult to detect and solve. Such a conflict occurs, for example, when conjunction of surface coercion rules [`1:a 2:b <= _`] and [`2:p 3:c <= _`] do not associate any surface form to lexical string `123`. These could be solved by adding a combined "super-rule" that has a strictly wider center. *E.g.*, [`1:a 2:b 3:3` $\cup$ `1:1 2:p 3:c <= _`] would override the original conflicting rules.

Disjunctive Ordering In Generative Phonology, alternative rewriting rules for the same phoneme are *disjunctively ordered* in such a way that a rule with the most specific environment condition is preferred over rules that apply elsewhere. Karttunen [19] presented a similar approach to *left-arrow conflicts* where two surface coercions claim opposite surface forms. The conflicting left-arrow rules can often, but not always, be ordered according to their specificity.

A given disjunctive order can be implemented easily. Define, for the coercion rule, an extended syntax $[l_i :: X_i \texttt{<=} \# L_i \_ R_i \#]$ that indicates the level of the rule. The rule belongs to l_i levels $1, 2, \ldots, l_i$. The rules at each level are put together under coherent intersection that resolves all left-arrow conflicts at that level. Accordingly, a coercion rule at level l will override a coercion rule at a level $j, 1 \leq j \leq l$ provided that the center of the rule at level j is not strictly longer. A rule with strictly longer center cannot be overriden by another rule, which is a potential problem in the current GTWOL.

We now update the definitions of W_2 and S_2 in such a way that both disjunctive ordering and the principle of longest applications are observed. The relation $\nu_{\star,j}$ is used to produce all entailed rules where diamonds mark the center and its substrings. This change is reflected by the redefinitions

$$W_2 = \cup_{i=1}^{p} L_i \, \nu_{2,l_i}(\pi_1^{-1}(\pi_1(X_i))) \, R_i; \quad S_2 = \cup_{i=1}^{p} L_i \, \nu_{\star,l_i}(X_i) \, R_i. \tag{37}$$

A *left-right arrow conflict* [15] is not very common but it occurs *e.g.* between surface coercion [1 :: `a:a <= c:c _`] and context restriction [1 :: `a:o => c:c _`]. The classical formalism guides the user to use double-arrow rules such as [1 :: `a:a <=> c:c _`]. Using double arrow rules is no longer necessary in GTWOL [15], because it is based on coherent intersection rather than intersection of rules. Accordingly, a successful rule application of one kind of rule overrides a corresponding failing application. Effectively, rules [1 :: `a:a => c:c _`] and [1 :: `a:o <= c:c _`] are thus implicitly added when one is specified.

However, coercion is considered stronger than restriction. At level 2 they can override and take precedence over conflicting context restriction rules that are at level 1. We can now combine context restrictions and surface coercions under coherent intersection, in a similar fashion as [15], by computing

$$[W_1 \overset{\Pi,2,M}{\Rightarrow} S_1] \Cap [W_2 \overset{\Pi,2,M}{\Rightarrow} S_2]. \tag{38}$$

Problem 4. How the disjunctive order of rules is determined most efficiently? How the remaining interaction, conflicts and overriding should be addressed?

7 Application: Bracketed Systems

We will now study applications of the GR operation in systems that use brackets to represent (i) overlap-free rewriting [7,11,23] or (ii) limited tree structures of context-free, dependency and mildly context-sensitive grammars [24,17,5,25,26].

7.1 Segmentation

Bracketed Generalized Two-Level Grammar (BGTWOL) [27] contains a rules whose centers are bracketed. In addition, every BGTWOL includes a default core $\textsc{Gen}_{\text{core}}$. This and *bracketed context restriction* (denoted by `(=>)`) are defined by equations

$$\textsc{Gen}_{\text{core}} = [\Pi \texttt{ => }] \Cap [I^* \texttt{ => } \_] \tag{39}$$

$$[X' \texttt{ (=>) } \#L_1\_R_1\#, \ldots \#L_n\_R_n\#] \overset{\text{def}}{=} [X' \texttt{ => } \#L'_1\_R'_1\#, \ldots \#L'_n\_R'_n\#] \text{ where}$$
$$X' \subseteq B_L X B_R;\ X, L_i, R_i \subseteq (\Pi \backslash B)^*, L'_i = d_B^{-1}(L_i), R'_i = d_B^{-1}(R_i) \text{ for every } i. \tag{40}$$

Rewriting or replace rules are usually represented in the literature with an arrow operator that relates a replaced string (or strings set) and its replacement [7,28]. Gerdemann and van Noord [29] argue that this kind of rule format fails to capture *backreferences* to the replaced in the replacement. In the current presentation, we assume that the center X and the contexts $L_1, R_2, \ldots L_n, R_n$ are regular subsets of $(\Pi \backslash B)^*$. This helps us to capture the maximally flexible definition of replace rules, including the backreferences and oriented contexts.

7.2 Tree Structures

According to a classical theorem due to Chomsky and Schützenberger [30], every context-free language is a homomorphic image of the intersection of a semi-Dyck language D_m [31] and a regular language R. This representation of context-free languages is varied in a few recent representations [5,25,9,26] whose regular approximations provide an excellent application for the GR operation.

In all these representations, the context-freeness of the system comes from a semi-Dyck language D_m whose strings have balanced bracketing. For example, the semi-Dyck language over alphabet $\{\texttt{<},\texttt{>}\}$ is the language D_1 generated by the context-free grammar with a single nonterminal symbol S, two terminal symbols $\texttt{<},\texttt{>}$, and productions $S \to S\texttt{<}S\texttt{>}S$ and $S \to \epsilon$.

Language $D'_1 \subset B^*$ is obtained from D_1 by substituting B_L and B_R respectively for the terminals $\texttt{<}$ and $\texttt{>}$. Such semi-Dyck languages that use m different kinds of labeled brackets are obtained from D'_1 with concatenation and Boolean operations [32,26]. Often language D'_1 is extended to language $\Delta = d^{-1}_{\Pi \setminus B}(D'_1)$ that contains also freely occurring non-bracket symbols.

Limited Bracketing The approximation $\Delta_i \subset d^{-1}_{\Pi\setminus B}(D'_1)$, where $i \in \mathbb{N}$, can be obtained by induction on the *bracketing depth* i using a nested GR as follows:

$$\begin{aligned}\Delta_0 &= \mathit{match\text{-}L\text{-}R}(\Pi^* B_L, \emptyset, B_R \Pi^*) = W \overset{\Pi,1,M}{\Rightarrow} \emptyset \\ &\quad \text{where } W = (\Pi^* B_L \diamond_1 \Pi^*) \cup (\Pi^* \diamond_2 B_R \Pi^*); \qquad (41)\\ \Delta_i &= \mathit{match\text{-}L\text{-}R}(\Pi^* B_R, \Delta_{i-1}, B_R \Pi^*) = W \overset{\Pi,1,M}{\Rightarrow} f_M(\Pi^* B_L \diamond_1 \Delta_{i-1} \diamond_2 B_R \Pi^*) \\ &\quad \text{where } i > 0 \text{ and } W = (\Pi^* B_L \diamond_1 \Pi^*) \cup (\Pi^* \diamond_2 B_R \Pi^*). \qquad (42)\end{aligned}$$

Language Δ_i is the largest set of strings over Π where the unlabeled bracketing is balanced and the depth of bracketing is limited by integer i, $i \geq 0$. This set is clearly star-free, because the generalized star-height of expression is zero.

If we add more brackets to B_L and B_R, the approximation Δ_i can be used in certain grammar representations in a useful way although language Δ_i itself does not check bracket labels. It can be used in approximations in such a way that labels get checked on a label-by-label basis [32,26]. Layerization (see below) provides an even better way to check labels without large constraint automata.

The context-free sets of parse trees exhibit tree locality that is lost in string-based representations and their star-free approximations. The inductive star-free definition (42) of limited bracketing Δ_i demonstrates that the approximation exhibits another form of locality [33] although the tree locality of context-free grammars is lost in approximation.

Subtypes of Representations There are two main types of grammar encodings that resemble the Chomsky-Schützenberger representations: **T1** is based on *local patterns* and **T2** is based on *sparse rules*. In **T1**, the semi-Dyck language is combined with the grammar by intersection. In **T2**, it is woven into the grammar rules with several regular operations.

Types **T1** and **T2** have subtypes: In [30], local patterns are based on constituent boundaries (**T1A**). In [26], local patterns describe argument structures (**T1B**). Sparse rules are based on either context restrictions (**T2A**) [17] or bracketing restriction constraints (**T2B**) [9]. Interestingly, all these subtypes are easily captured by the GR operation.

For each subtype **T1A**, ..., **T2B**, a typical GR rule is given in the following. The following examples are based on the local constituency tree [S → NP VP] or the local dependency tree [hit $\rightarrow$ SUBJ $\star$ OBJ].

$$\mathbf{T1A}: \quad \Pi^* \ \nu'_{1,1}(\Pi) \ \Pi^* \overset{\Pi,1,M}{\Rightarrow} \Pi^* \ \nu'_{\star,1}(<_{\mathrm{S}}<_{\mathrm{NP}} \cup >_{\mathrm{NP}}<_{\mathrm{VP}} \cup >_{\mathrm{VP}}<_{\mathrm{S}}) \ \Pi^* \tag{43}$$

$$\mathbf{T1B}: \quad \Pi^* \ \nu'_{1,1}(\Pi) \ \Pi^* \overset{\Pi,1,M}{\Rightarrow} \Pi^* \ \nu'_{\star,1}(>_{\overleftarrow{\mathrm{SUBJ}}} \ hit \ <_{\overrightarrow{\mathrm{OBJ}}}) \ \Pi^* \tag{44}$$

$$\mathbf{T2A}: \quad \Pi^* \diamond_1 <_{\mathrm{NP}} \Delta >_{\mathrm{NP}} \diamond_1 \Pi^* \overset{\Pi,2,M}{\Rightarrow} \Pi^* <_{\mathrm{S}} \diamond_1 <_{\mathrm{NP}} \Delta >_{\mathrm{NP}} \diamond_1 <_{\mathrm{VP}} \Delta >_{\mathrm{VP}} >_{\mathrm{S}} \Pi^* \tag{45}$$

$$\mathbf{T2B}: \quad \Pi^* <_{\mathrm{S}} \diamond_1 \Delta \diamond_1 >_{\mathrm{S}} \Pi^* \overset{\Pi,2,M}{\Rightarrow} \Pi^* <_{\mathrm{S}} \diamond_1 <_{\mathrm{NP}} \Delta >_{\mathrm{NP}} <_{\mathrm{VP}} \Delta >_{\mathrm{VP}} \diamond_1 >_{\mathrm{S}} \Pi^* \tag{46}$$

Problem 5. What other grammar systems could be approximated through this kind of representations?

Layerization In the approximated **T2** representations, the compiled grammar rules tend to grow undesirably when the depth of bracketing grows. Such grammars could be represented, however, much more compactly through *layerization.*

The layerization technique (Section 5.2) is an additional example of the flexibility of the GR operation. Each layer can correspond to a grammar that constraints the labeled bracketing at a given depth [9,34]. Additional preconditions added to generalized restrictions split the rules into layers that are easy to combine. A similar approach optimizes **T1** representations.

8 Application: Bimorphisms

The notion of bimorphism has been introduced in connection to tree transformations [35]. However, because strings are a special case of trees, it is possible to restrict tree bimorphisms to string bimorphisms. Let Σ_1, Σ_2 and Π be alphabets. A *bimorphism* is a triple (ψ_1, P, ψ_2) where $\psi_1 : \Pi^* \rightarrow \Sigma_1^*$ is the *input homomorphism*, $P \subseteq \Pi^*$ is the *pivot*, and $\psi_2 : \Pi^* \rightarrow \Sigma_2^*$ is the *output homomorphism.* The transformation relation $\beta(P) \subseteq \Sigma_1^* \times \Sigma_2^*$ computed by bimorphism is defined as $\beta(P) = \{(\psi_1(w), \psi_2(w)) \mid w \in P\}$.

We give two examples on how GR can be used to describe regular relations through a bimorphism.

Relations Defined by Two-Level Grammars The rule component of every two-Level grammar G [8,15] describes the language $\textsc{Gen}_G \subseteq \Pi^*$. Let $\psi_1(w) = \pi_1(w)$, $\psi_2(w) = \pi_2(w)$, $\Sigma_1 = \psi_1(\Pi)$ and $\Sigma_2 = \psi_2(\Pi)$. The grammar G defines bimorphism $(\psi_1, \textsc{Gen}_G, \psi_2)$.

Relations Defined by Conditional Optional Replace *Conditional optional replace (without overlaps)* [36,27], denoted by `(->)`, can be implemented with a bracketed context restriction P and bimorphism (ψ_1, P, ψ_2) where $\psi_1(w) = \pi_1(d_B(w))$ and $\psi_2(w) = \pi_2(d_B(w))$. This is defined by the equation

$$X \ \texttt{(->)} \ \#L_1\_R_1\#, \ldots \#L_n\_R_n\# \stackrel{\mathrm{def}}{=} \beta(\textsc{Gen}_G)$$
$$\text{where } \textsc{Gen}_G = \textsc{Gen}_{\mathrm{core}} \cap\!\!\!\!\cdot\, [\texttt{<}X\texttt{>} \ \texttt{(=>)} \ \#L_1\_R_1\#, \ldots \#L_n\_R_n\#]. \tag{47}$$

Note that the presented new syntax [27] for the replace operator is inspired by Two-Level Grammar [8]. When Beesley and Karttunen [20] write a replace rule as [`a (->) b //c _ d`], we write the same as [`a:b (->)` $\#\Pi^*\pi_2^{-1}(\texttt{c})$ _ $\pi_1^{-1}(\texttt{d})\Pi^*\#$].

Problem 6. Could a bimorphism be used to relate two GR-based grammars? Such an arrangement could be useful in machine translation.

Problem 7. Could the GR operation be used to describe properties of tree languages? Recall that the spine language of recognizable tree languages is regular and thus closed under the Boolean operations and diamond elimination.

9 Application: Optimality Theoretic Systems

Strict Preference Relations An interesting application of the GR operator suggests itself when the pivot language P of a bimorphism (ψ_1, P, ψ_2) is bracketed. Yli-Jyrä [27] derives different kinds of replace rules from optional replace rules using *strict preference relations* $T \subseteq I^* \times I^*$ such as

$$T_{\text{most+}} \overset{\text{def}}{=} \{(\pi_1(w), \pi_2(w)) | w \in \left(B_L{:}0\, \Sigma^+ B_R{:}0 \cup \Sigma \cup B\right)^*\} \tag{48}$$

$$T_{\text{lest},B'} \overset{\text{def}}{=} \{(w, w') | w, w' \in I^*, d_B(w) = d_B(w'), w \notin I^* B' I^*, w' \in I^* B' I^*\} \tag{49}$$

$$T_{\text{lr}} \overset{\text{def}}{=} \{(vby, vau) | v, y, u \in I^*, a \in \Pi \backslash B, b \in B_L, d_B(y) = d_B(au)\} \tag{50}$$

$$T_{\text{lrlong}} \overset{\text{def}}{=} \{(vau, vby) | v, u, y \in I^*, a \in \Pi \backslash B, b \in B_R, d_B(y) = d_B(au)\}. \tag{51}$$

Jäger's Composition Operation The preference relations are used as an optimality-theoretic (OT) constraint component (Con) that ranks of the candidates. The composition of the pivot $\textsc{Gen}_G \subseteq \Pi^*$ with the constraints in Con is implemented with a left-associative operator `r-glc` that is defined by

$$\textsc{Gen}_G \ \texttt{r-glc}\ T \overset{\text{def}}{=} \{w \in \textsc{Gen}_G \,|\, \neg\exists w'(w' \in \textsc{Gen}_G \wedge (\pi_1(w), \pi_1(w')) \in T)\}. \tag{52}$$

The operator `r-glc` is a variant of `glc`, an operator that Jäger [37] controversially coined *generalized lenient composition (GLC)*.[3]

9.1 Examples

The *conditional obligatory replace* rule (denoted by `->`) and the *conditional left-to-right longest replace* rule (denoted by `@->`) are compiled as follows:

$$X \ \texttt{->}\ \#L_1\_R_1\#, \ldots, \#L_n\_R_n\# \overset{\text{def}}{=} \beta(\textsc{Gen}'_G\ \texttt{r-glc}\ (T_{\text{most+}} \cup T_{\text{lest},B_2})) \tag{53}$$

$$X \ \texttt{@->}\ \#L_1\_R_1\#, \ldots, \#L_n\_R_n\# \overset{\text{def}}{=} \beta(\textsc{Gen}_G\ \texttt{r-glc}\ (T_{\text{lr}} \cup T_{\text{lrlong}})) \tag{54}$$

$$\text{where } \textsc{Gen}_G = [\texttt{<}_1 \quad X\ \texttt{>}_1\ \texttt{(=>)}\ \#L_1\_R_1\#, \ldots, \#L_n\_R_n\#] \mathbin{\text{⋒}} \textsc{Gen}_{\text{core}}$$

$$\text{and } \textsc{Gen}'_G = [\texttt{<}_2\pi_1(X)\texttt{>}_2\ \texttt{(=>)}\ \#L_1\_R_1\#, \ldots, \#L_n\_R_n\#] \mathbin{\text{⋒}} \textsc{Gen}_G.$$

[3] According to Dale Gerdemann (p.c., 2008), the GLC operator rather addresses a crucial problem with ordinary lenient composition than generalizes the operator.

In order to compile parallel conditional obligatory replacement, Kempe and Karttunen [28] employ a large number of brackets. Skut *et al.* [38] presents a rule compiler for ranked replace rules. Such ranking can be implemented by combining a bracketed context restriction and a GLC-based parse ranking [27].

9.2 The Principled Design for Constrained Bimorphisms

The extended bimorphism in (53) and (54) is structured in a similar fashion as Optimality Theory [39]. The roles of the candidate generator language $\textsc{Gen}_G$, the constraint component $\textsc{Con}$ and the transformation β are outlined as follows:

$$\beta(\textsc{Gen}_G \circ \textsc{Con}) = \beta(\textsc{Gen}_G\ \texttt{glc}_1\ \textsc{Con}_1 \ldots\ \texttt{glc}_c\ \textsc{Con}_c) \tag{55}$$
$$\text{where } \texttt{glc}_i \in \{\texttt{glc}, \texttt{r-glc}, \texttt{b-glc}\} \text{ is left associative.}$$

It is interesting that the structure of (55) separates tasks according to their descriptive complexity. Because the candidate language $\textsc{Gen}_G$ is described with a generalized restriction whose arguments are, almost without question, star-free, $\textsc{Gen}_G$ is typically star-free and thus captured by $FO[<]$. The $\textsc{Con}$ constraints that compare candidate strings are not same-length relations [7] but they are regular relations that could be themselves described with bimorphisms. Finally, homomorphisms inside β are just stateless mappings and they have therefore very simple structure. Accordingly, the components in (55) are relations that are contrasted as follows:

$$\text{stateless } \beta \textit{ vs.} \text{ star-free same-length } \textsc{Gen}_G \textit{ vs.} \text{ regular } \textsc{Con}. \tag{56}$$

We believe that (55) is a design pattern that applies to numberless situations and helps us to develop algorithms that are designed to address different kinds of tasks efficiently.

10 Dot-Depth

Non-determinism and locality are related concepts. Mráz *et al.* [40] use the amount of encoded structural information as a measure for the degree of non-determinism of context-free grammars. If enough information on categories is added to the strings of a context-free language, the language becomes a deterministic context-free language. In a similar fashion, the star-freeness of a regular language means essentially that there is enough information to make the language star-free.

The Dot-Depth Hierarchy The amount of locality in star-free languages can be measured using the forbidden pattern hierarchy, the group hierarchy and the concatenation hierarchies such as the dot-depth hierarchy [4,41,42]. The dot-depth hierarchy corresponds in a very natural way to the classical hierarchy of first-order logic based on the alternation of existential and universal quantifiers in the prenex normal-form [4].

The *dot-depth hierarchy* was introduced by Cohen and Brzozowski [3]. In the dot-depth hierarchy, the first level, $\mathbf{B}_0$, is the Boolean closure of trivial languages $\{a\}$, $a \in \Pi$. An intermediate family of languages, $\mathbf{M}_i$, $i \geq 0$, contains the concatenations of zero or more languages from $\mathbf{B}_{i-1}$. The next level, $\mathbf{B}_i$, $i > 0$, consists of the Boolean combinations of the languages in $\mathbf{M}_{i-1}$.

Although an upper bound for the dot-depth of a language can be computed from a corresponding star-free expression, we do not know if there is a general decision procedure for the exact dot-depth [42].

The Measuring Problem Star-free constructions such as in [43,5] allows for proving by induction on i that the dot-depth of language Δ_i is actually not larger than $i+1$, because Δ_i could be contained to $\mathbf{B}_{i+1}$. This is done as follows:

$$\begin{aligned}
\Pi^* &= \overline{\{a\} - \{a\}} && : \mathbf{B}_0 \\
\Delta_0 &= \overline{\Pi^* \{<,>\} \Pi^*} && : \mathbf{B}_1 \\
\lambda_i &= B_L \Delta_{i-1} B_L \Delta_{i-2} B_L \ldots \Delta_1 B_L \Delta_0 B_L && : \mathbf{M}_i \\
\rho_i &= B_R \Delta_0 B_R \Delta_1 B_R \ldots \Delta_{i-2} B_R \Delta_{i-1} B_R && : \mathbf{M}_i \\
\Delta_i &= \overline{\Pi^* \lambda_i \Pi^*} \cap \overline{\Pi^* \rho_i \Pi^*} \cap \overline{\Delta_{i-1} B_R \Pi^*} \cap \overline{\Pi^* B_L \Delta_{i-1}} && : \mathbf{B}_{i+1}.
\end{aligned} \tag{57}$$

The GR-based construction (42) involves about $2i$ nested applications of complementation and local closure f_M. It is, however, a surprising fact that these do not seem to contribute to the dot-depth of the *languages* $\mathbf{B}_i$ more than one level per an induction step because the dot-depth of these languages cannot be bigger than the upper-bounds that are computed in (57). The upper bounds apply to the construction (42) as follows:

$$\begin{aligned}
\mathbf{B}_0 : &\quad \Pi^* = \overline{\{a\} - \{a\}} \\
\mathbf{B}_1 : &\quad W = (\Pi^* B_L \diamond_1 \Pi^*) \cup (\Pi^* \diamond_2 B_R \Pi^*); \;\; \Delta_0 = \Pi^* \backslash f_M(W) \\
\mathbf{M}_{i+1} : &\quad W'_i = \Pi^* B_L \diamond_1 \Delta_i \diamond_2 B_R \Pi^* \\
\mathbf{B}_{i+1} : &\quad \Delta_i = \Pi^* \backslash f_M(W \backslash f_M(W'_{i-1})).
\end{aligned} \tag{58}$$

Problem 8. Can we include generalized restriction to the operations used to build the dot-depth hierarchy? What is the contribution of diamond elimination to the dot-depth of the language?

11 Optimized Implementation

In Section 1, we mentioned a few potential objections against the GR operation. Some of the issues remain to be addressed. For example, we would like to compile grammars on the fly and apply them efficiently to surface syntactic parsing or to construction of lexical transducers.

Guided Intersection The intersection of Two-Level rules would normally be too large [44], which causes difficulties if we try to compile the grammar component in separation. Karttunen [44] addresses this problem with a high-arity operation: *intersecting composition.* Under this operation in expression $L\ (\circ\cap)\ (R_1, R_2, \ldots, R_r)$, the intersection of the phonological constraints R_1, $R_2, \ldots, R_r \subseteq \Pi^*$ is simultaneously restricted under composition with a regular relation L that represents the pairs of analyses and lexical forms.

A comparable approach can be used when the grammar is compiled using the GR operation. This time, however, a constraint language $L' = \pi_1^{-1}(\{\ x_2 \mid (x_1, x_2) \in L\})$ based on the set of lexical strings in lexicon L should be added to the postcondition as a conjunctive:

$$L \circ [L' \cap (W \overset{\Pi,2,M}{\Rightarrow} W')] = L \circ [\diamond_0 \Pi^* \cup W \overset{\Pi,2,M}{\Rightarrow} \diamond_0 L' \cup W'] = L \circ [\Pi^* \backslash d_M(W'')]$$
$$\text{where } W'' = (\diamond_0 \Pi^* \cup W) \backslash (\diamond_0 L' \cup W'). \tag{59}$$

A new diamond, $\diamond_0$, is concatenated to L' because it is not guaranteed that $W' \cap \Pi^* \subseteq L'$. We see now that the set of lexical forms and the grammar can both be combined under the GR operation, but we do not yet obtain an efficient compilation method without further optimizations. Languages W and W' are typically similar to local grammar languages such as $\Pi^* G \Pi^*$ of Mohri (2005), since it often happens that $W = \Pi^* W \Pi^*$ or $W' = \Pi^* W' \Pi^*$. A slightly improved compilation method would take advantage of the sparseness and locality of the grammar constraints.

Non-Deterministic Failure Automata Because W and W' are obtained as unions from individual grammar rules, it is natural and space efficient to represent these languages with *non-deterministic automata.* It might be also a good idea to compress these automata by optimizing their transition relations using *failure transitions* [18].

Optimized State Subsets The classical subset construction algorithm [45] constructs a deterministic automaton by creating all subsets of the state set that are reachable from the initial state with some common strings. The failure representation [18] optimizes also determinization, because it makes earlier states more *popular subset elements* than the latter states. In addition, many subsets could be merged easily by a trick that we call *final loop optimization*: if the subset contains an element state q that recognizes the universal language over the alphabet of the remaining suffixes, it is of no use to add any other element states to the subset.

Guided Determinization Suppose that we want to determinize the automaton recognizing W'' before the diamonds are removed from it. In order to take better advantage of the final loop optimization, we would like to ensure that W'' is as large as possible. Accordingly, we add *all strings that do not occur in* *lexicon* (*i.e.* the strings in $\overline{L'} = \mathit{\Pi}^* \backslash L'$) to the minuend $(\diamond_0 \mathit{\Pi}^* \cup W)$. In addition, these bad strings can contain diamonds freely. This change can make W'' larger, but $d_M(W'')$ remains the same:

$$d_M(W'') = d_M((\diamond_0 \mathit{\Pi}^* \cup W \cup d_0^{-1}(\overline{L'}))\backslash(\diamond_0 L' \cup W')). \tag{60}$$

A non-deterministic automaton recognizing $(\diamond_0 \mathit{\Pi}^* \cup W \cup d_0^{-1}(\overline{L'}))$ reaches a final loop when it recognizes a marked string $w \in (\mathit{\Pi} \cup M)^*$ for which $w(\mathit{\Pi} \cup M)^* \cap d_0^{-1}(L') = \emptyset$. This optimizes the subset construction considerably. The resulting method would apply the grammar to the lexicon in a very much similar way as intersecting composition [44], *i.e.* by avoiding paths that are not supported by the lexicon.

Subtracting Determinization In the above, the subtrahend $(\diamond_0 L' \cup W')$ has to be determinized. This can be a bottleneck, because typically $W' = \mathit{\Pi}^* W'$. we can, however, postpone the determinization of the subtrahend by using de Morgan's law, which allows us, in a way, *subtract during determinization*:

$$(\diamond_0 \mathit{\Pi}^* \cup W \cup d_0^{-1}(\overline{L'}))\backslash(\diamond_0 L' \cup W') = \overline{\overline{\diamond_0 \mathit{\Pi}^* \cup W \cup d_0^{-1}(\overline{L'})} \cup \diamond_0 L' \cup W}. \tag{61}$$

Specialization We can *specialize* the generalized postcondition W' of the grammar by intersecting it with lexicon L' because only the strings in L' need permissions. The subtrahend can, thus, be replaced with $(\diamond_0 L' \cup (d_M^{-1}(L') \cap W'))$. It is also possible that the determinization of sub-expression $\diamond_0 \mathit{\Pi}^* \cup W \cup \overline{L'}$ is a bottleneck, because typically $W = \mathit{\Pi}^* W$. To address this problem, we can specialize W and rewrite the sub-expression as $\diamond_0 \mathit{\Pi}^* \cup (d_M^{-1}(L') \cap W) \cup \overline{L'}$.

In comparison to W, intersection $d_M^{-1}(L') \cap W$ is easier to determinize as a part of the subexpression: it looks like its correlate $d_0^{-1}(\overline{L'})$ is already in the union. Moreover, if very few marked strings in W applied to the lexicon, the intersection would result into a small or empty language, which reduces the burden of determinization. The same could happen also with the strings in W', which suggests that both W and W' should be specialized as in the equation

$$\mathit{\Pi}^* \backslash d_M(W'') = \mathit{\Pi}^* \backslash d_M(\overline{W'''})$$
$$\text{where } W''' = \overline{\diamond_0 \mathit{\Pi}^* \cup (d_M^{-1}(L') \cap W) \cup d_0^{-1}(\overline{L'})} \cup \diamond_0 L' \cup (d_M^{-1}(L') \cap W'). \tag{62}$$

In sum, a deterministic automaton recognizing the marked language can be constructed, in most cases, without much effort on useless paths. While this language still contains diamonds, it is a significant step in computing Formula (59) efficiently.

We look forward to experiments that compare the GR-based compilation method for two-level grammars with Karttunen's intersecting composition [44].

Problem 9. Is there a lazy algorithm that would (1) determinize, (2) complement, (3) remove diamonds and (4) determinize using dynamic programming? Can it compute $d_M(\overline{W'''})$ more efficiently than the step-by-step approach?

Problem 10. Are there real cases where the presented optimization is not sufficient? Can the implementation of the GR operator be optimized for them? Can the evaluation of GR take advantage of layerization?

Problem 11. Can we define weighted generalized restriction and optimize it in different applications?

Problem 12. Can we define the GR operator even more generally without loosing its good properties? Study the use of $\nu_\star$ with coherent intersection.

12 Conclusion

Generalized restriction is a new and lesser-known star-free operation. It takes advantage of special-purpose marker symbols, diamonds, when combining the Boolean operators with concatenation. It increases the succinctness of star-free expressions and can be used with other regular operators. The operator has several important applications. It expresses a large family of constraints, rules and grammars as languages whose strings contain diamonds. An elegant representation for transducers is obtained by defining transductions via bimorphisms where generalized restriction describes the pivot. Inside bimorphisms, the operator can generate a set of candidates for a system of violable constraints.

We discussed many properties and applications of generalized restriction and identified twelve open problems. In addition, we sketched an optimized compilation method for two-level grammars.

Acknowledgements

In 2002, Lauri Carlson provided invaluable intellectual feedback and support during my first explorations of star-freeness [5] and first-order definability in finite-state grammars. This article was improved through some critical comments by Dale Gerdemann, Måns Hulden and Kimmo Koskenniemi in 2008.

References

1. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Information and Control **8**(2) (1965) 190–194
2. McNaughton, R., Papert, S.: Counter-free Automata. Research Monograph No. 65. MIT Press (1971)
3. Cohen, R.S., Brzozowski, J.A.: Dot-depth of star-free events. Journal of Computer and System Sciences **5** (1971) 1–15
4. Thomas, W.: Classifying regular events in symbolic logic. Journal Comput. System Sci. **25** (1982) 360–376
5. Yli-Jyrä, A.: Describing syntax with star-free regular expressions. In: 11th EACL 2003, Proceedings of the Conference, Budapest, Hungary (2003) 379–386
6. Johnson, C.D.: Formal aspects of phonological description. Number 3 in Monographs on linguistic analysis. Mouton, The Hague (1972)

7. Kaplan, R.M., Kay, M.: Regular models of phonological rule systems. Computational Linguistics **20**(3) (1994) 331–378
8. Koskenniemi, K.: Two-level morphology: a general computational model for word-form recognition and production. Number 11 in Publications. Department of General Linguistics, University of Helsinki, Helsinki (1983)
9. Yli-Jyrä, A., Koskenniemi, K.: Compiling contextual restrictions on strings into finite-state automata. In Cleophas, L., Watson, B.W., eds.: The Eindhoven FASTAR Days, Proceedings. Number 04/40 in Computer Science Reports, Eindhoven, The Netherlands, Technische Universiteit Eindhoven (2004) Also available at `www.ling.helsinki.fi/~aylijyra/dissertation/7.pdf`.
10. Karttunen, L., Koskenniemi, K., Kaplan, R.M.: A compiler for two-level phonological rules. Report CSLI-87-108, Center for Study of Language and Information, Stanford University, CA (1987)
11. Grimley-Evans, E., Kiraz, G.A., Pulman, S.G.: Compiling a partition-based two-level formalism. In: 16th COLING 1996, Proc. Conference. Volume 1., Copenhagen, Denmark (1996) 454–459
12. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Transactions of the American Mathematical Society **98**(1) (1961) 21–51
13. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **6** (1960) 66–92
14. Vaillette, N.: Logical Specification of Finite-State Transductions for Natural Language Processing. PhD thesis, Ohio State University (2004)
15. Yli-Jyrä, A., Koskenniemi, K.: Compiling generalized two-level rules and grammars. In: Proceedings of FinTAL 2006. LNAI (2006)
16. Pin, J.E., Straubing, H., Thérien, D.: Some results on the generalized star-height problem. Information and Computation **101**(2) (1992) 219–250
17. Koskenniemi, K., Tapanainen, P., Voutilainen, A.: Compiling and using finite-state syntactic rules. In: Proc. COLING'92. Volume I., Nantes, France (1992) 156–162
18. Mohri, M.: Local grammar algorithms. In Arppe *et al.*, A., ed.: Inquiries into Words, Constraints, and Contexts. Festschrift in Honour of Kimmo Koskenniemi on his 60th Birthday. CSLI Publications (2005) 84–93
19. Karttunen, L.: Finite-state constraints. In: Proceedings of the International Conference on Current Issues in Computational Linguistics, Universiti Sains Malaysia, Penang, Malaysia (1991)
20. Beesley, K., Karttunen, L.: Finite state morphology. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, CA, USA (2003)
21. Black, A., Ritchie, G., Pulman, S., Russell, G.: Formalisms for morphographemic description. In: 3rd EACL 1985, Proceedings of the Conference, Copenhagen, Denmark (1987) 11–18
22. Yli-Jyrä, A., Koskenniemi, K.: A new method for compiling parallel replacement rules. In Holub, J., Žd'árek, J., eds.: Implementation and Application of Automata, 12th International Conference, CIAA 2007, Revised Selected Papers. Volume 4783 of LNCS., Springer (2007) 320–321
23. Barthélemy, F.: Partitioning multitape transducers. In: Pre-proceedings of FSMNLP 2005. (2005) 3–12 The post-proceedings will appear in the LNAI series of Springer-Verlag.
24. Krauwer, S., des Tombe, L.: The finite state transducer as a theory of language. Utrect Working Papers in Linguistics, UWPL **9** (1980) 1–86
25. Yli-Jyrä, A.: Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In Kruijff, G.J.M.,

Duchier, D., eds.: Proc. Workshop of Recent Advances in Dependency Grammar, Geneva, Switzerland (2004) 33–40

26. Yli-Jyrä, A.: Approximating dependency grammars through intersection of star-free regular languages. International Journal of Foundations of Computer Science **16**(3) (2005) 565 – 579
27. Yli-Jyrä, A.: Transducers from parallel replacement rules and modes with generalized lenient composition. In: Proceedings of FSMNLP 2007, Potsdam, Germany (2008)
28. Kempe, A., Karttunen, L.: Parallel replacement in finite state calculus. In: 16th COLING 1996, Proc. Conference, Copenhagen, Denmark (1996) 622–627
29. Gerdemann, D., van Noord, G.: Transducers from rewrite rules with backreferences. In: 9th EACL 1999, Proceedings of the Conference. (1999) 126–133
30. Schützenberger, M.P.: On context-free languages and push-down automata. Information and Computation (Information and Control) **6** (1963) 246–264
31. Harrison, M.A.: Introduction to Formal Language Theory. Reading, Massachusetts, Addison-Wesley (1978)
32. Wrathall, C.: Characterizations of the Dyck sets. RAIRO – Informatique Théorique **11**(1) (1977) 53–62
33. Hella, L., Libkin, L., Nurmonen, J.: Notions of locality and their logical characterizations over finite models. Journal of Symb. Logic **64**(4) (1999) 1751–1773
34. Yli-Jyrä, A.: Contributions to the theory of finite-state based grammars. Number 38 in Publications. Department of General Linguistics, University of Helsinki (2005)
35. Arnold, A., Dauchet, M.: Morphismes et bimorphismes d'arbres. Theoretical Computer Science **20** (1982) 33–93
36. Karttunen, L.: The replace operator. In: 33th ACL 1995, Proceedings of the Conference, Cambridge, MA, USA (1995) 16–23
37. Jäger, G.: Gradient constraints in finite state OT: The unidirectional and the bidirectional case. In: Finite State Methods in Natural Language Processing 2001 (FSMNLP 2001), ESSLLI Workshop, Helsinki (2001) (35–40)
38. Skut, W., Ulrich, S., Hammervold, K.: A flexible rule compiler for speech synthesis. In: Proc. Intelligent Information Systems 2004, Zakopane, Poland (2004)
39. Prince, A., Smolensky, P.: Optimality Theory: Constraint interaction in generative grammar. Technical Report RuCCS TR-2, Rutgers University Center for Cognitive Science, New Brunswick, NJ (1993)
40. Mráz, F., Plátek, M., Otto, F.: A measure for the degree of nondeterminism of context-free languages. In Holub, J., Žd'árek, J., eds.: Implementation and Application of Automata, 12th International Conference, CIAA 2007, Revised Selected Papers. Volume 4783 of LNCS., Springer (2007) 192–202
41. Glaßer, C.: Forbidden-patterns and word extensions for concatenation hierarchies. PhD thesis, Bayerischen Julius-Maximilians-Universität Würzburg (2001)
42. Pin, J.E.: Algebraic tools for the concatenation product. Theoretical Computer Science **292**(1) (2003) 317–342
43. Thomas, W.: A concatenation game and the dot-depth hierarchy. In: Computation Theory and Logic, In Memory of Dieter Rödding. Volume 270 of Lecture Notes In Computer Science. Springer (1987) 415–426
44. Karttunen, L.: Constructing lexical transducers. In: 15th COLING 1994, Proceedings of the Conference. Volume 1., Kyoto, Japan (1994) 406–411
45. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: principles, techniques, and tools. Addison-Wesley Publishing Company, Reading, Massachusetts (1986)