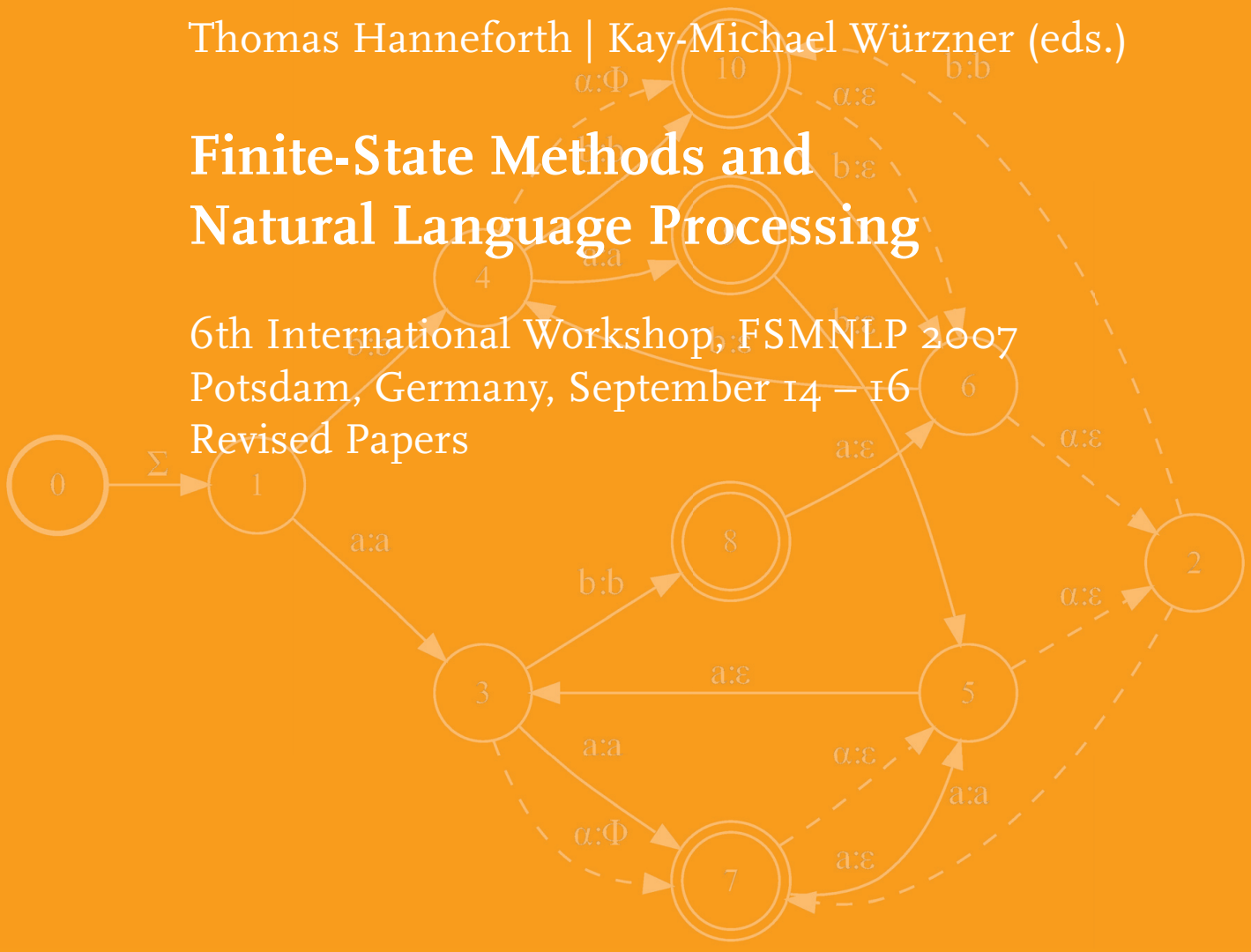
Universität Potsdam

Thomas Hanneforth | Kay-Michael Würzner (eds.)

Finite-State Methods and Natural Language Processing

6th International Workshop, FSMNLP 2007
Potsdam, Germany, September 14 – 16
Revised Papers



Potsdam University Press

Finite-State Methods and Natural Language Processing

Thomas Hanneforth | Kay-Michael Würzner (eds.)

Finite-State Methods and Natural Language Processing

6th International Workshop, FSMNLP 2007
Potsdam, Germany, September 14 – 16
Revised Papers

Potsdam University Press

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Universitätsverlag Potsdam 2008
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 4625
E-Mail: verlag@uni-potsdam.de

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2008/2381/
URN urn:nbn:de:kobv:517-opus-23812
[http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-23812]

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-940793-57-7

Preface

These proceedings contain the revised versions of the paper presented at the 6th International Workshop on Finite-State Methods and Natural Language Processing, FSMNLP 2007. FSMNLP 2007 was held in Potsdam, Germany, on September 14 – 16, 2007 and covered a wide range of topics from morphology to temporal logics.

This volume contains 15 regular and 3 invited papers.

The editors would like to thank all members of the Programme Committees for their painstaking and precise work. Many thanks go to our invited speakers Lauri Karttunen, Bruce Watson and Anssi Yli-Jyrä. We would also like to thank our co-organizers Sina Zarrieß, Andrea Corradini and Jonas Kuhn for their energy and enthusiasm. Alexander Siebert created our conference web site and Jan Engel the conference poster, many thanks for that!

We are deeply grateful to our sponsor, the Zentrum für Allgemeine Sprachwissenschaft and its assistant director Hans-Martin Gärtner. Last but not least, without the students and our secretary Ines Mauer who helped us this event wouldn't have been possible.

Potsdam
July 2008

Thomas Hanneforth Kay-Michael Würzner

Organisation

FSMNLP 2007 was organised by the Linguistics Department, Potsdam University.

Invited Speakers

Lauri Karttunen http://www2.parc.com/istl/members/karttune/:
New Features in PARC Finite State Toolkits

Bruce Watson http://www.cs.up.ac.za/cs/bwatson/:
Advances in automata implementation techniques

Anssi Yli-Jyrä http://www.ling.helsinki.fi/~aylijyra/:
The Hidden Jewels of Double Negation

Steering Committee

Lauri Karttunen (Palo Alto Research Center, USA and Stanford University, USA)

Kimmo Koskenniemi (University of Helsinki, Finland)

Kemal Oflazer (Sabanci University, Turkey)

Anssi Yli Jyrä (University of Helsinki and CSC – Scientific Computing Ltd., Espoo)

Programme Committee

Cyril Allauzen (Courant Institute, NYU, USA)
http://cs.nyu.edu/~allauzen/

Francisco Casacuberta (Instituto Tecnológico De Informática, Valencia)
http://web.iti.upv.es/~fcn/

Damir Cavar (University of Zadar, Croatia)
http://eng.unizd.hr/~dcavar/

Jean-Marc Champarnaud (Université de Rouen, France)
http://jmc.feydakins.org/

Maxime Crochemore (Université de Marne-la-Vallée, France)
http://www-igm.univ-mlv.fr/~mac/

Colin de la Higuera (Jean Monnet University, Saint-Etienne, France)
http://eurise.univ-st-etienne.fr/~cdlh/

Jan Daciuk (Dept. of Knowledge Engineering, Gdansk University of Technology, Poland)
http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/

Karin Haenelt (Fraunhofer Gesellschaft & University of Heidelberg)
http://kontext.fraunhofer.de/haenelt/

Gerhard Jäger (Universität Bielefeld, Germany)
http://wwwhomes.uni-bielefeld.de/gjaeger/home.html

Lauri Karttunen (Palo Alto Research Center, USA)
http://www2.parc.com/istl/members/karttune/

André Kempe (Yahoo Search Technologies, Paris)
http://a.kempe.free.fr

András Kornai (Budapest Institute of Technology, Hungary)
http://www.kornai.com/

Marcus Kracht (Department of Linguistics, UCLA, USA)
http://www.linguistics.ucla.edu/people/Kracht/

Eric Laporte (Université de Marne-la-Vallée, France)
http://www-igm.univ-mlv.fr/~laporte/index_en.html

Krista Lagus (Helsinki University of Technology, Finland)
http://www.cis.hut.fi/krista/

Stoyan Mihov (Bulgarian Academy of Sciences, Sofia)
http://lml.bas.bg/~stoyan

Hermann Ney (RWTH Aachen)
http://www-i6.informatik.rwth-aachen.de/web/Staff/ney/index.html

Kemal Oflazer (Sabanci University, Turkey & Carnegie Mellon University, Pittsburgh, USA)
http://people.sabanciuniv.edu/~oflazer/

Max Silberztein (Université de Franche-Comté, France)
http://www.nooj4nlp.net/

Richard Sproat (University of Illinois at Urbana-Champaign, USA)
http://www.linguistics.uiuc.edu/rws/

Barbara Stiebels (Zentrum für Allgemeine Sprachwissenschaft, Berlin)
http://www.zas.gwz-berlin.de/mitarb/homepage/stiebels/

Enrique Vidal (Instituto Tecnológico De Informática, Valencia)
http://web.iti.upv.es/~evidal/

Bruce Watson (Dept. of Computer Science, University of Pretoria, South Africa)
http://www.cs.up.ac.za/cs/bwatson/

Shuly Wintner (University of Haifa, Israel)
http://cs.haifa.ac.il/~shuly/

Anssi Yli-Jyrä (University of Helsinki & CSC Scientific Computing Ltd., Finland)
http://www.ling.helsinki.fi/~aylijyra/

Local Organization Committee

Thomas Hanneforth **chair** (University of Potsdam, Germany)
mailto:tom@ling.uni-potsdam.de

Jonas Kuhn (University of Potsdam, Germany)
mailto:kuhn@ling.uni-potsdam.de

Kay-Michael Würzner (University of Potsdam, Germany)
mailto:wuerzner@ling.uni-potsdam.de

Sina Zarrieß (University of Potsdam, Germany)
mailto:zarriess@uni-potsdam.de

Andrea Corradini (University of Potsdam, Germany)
mailto:andrea@.ling.uni-potsdam.de

Table of Contents

Part I

Invited Papers

New Features in PARC Finite State Toolkits

Lauri Karttunen

Palo Alto Research Center

The xfst utility that accompanied the Beesley & Karttunen book on "Finite-State Morphology" in 2003 ([1]) has been updated. In this talk I will describe the new features and show how they can be deployed. The new features include:

- full UTF-8 support
- symbol ranges (e.g. "a-z")
- definitions for lists (sets of symbols)
- list flags (symbols for member of, not a member of)
- insert flags (symbols for an embedded language)
- built-in and user-definable functions

The publicly available xfst tool is the "little sister" of a more powerful utility called fst that is used in commercial applications at companies such as Inxight (recently acquired by Business Objects) and Powerset. One of the new features in the fst tool is a pattern matching algorithm that overcomes the limitations inherent in the approach introduced in the B&K book. As the book explains, in the (x)fst regular expression language it is possible to define pattern networks with expressions such as

Example 1. `(Det) Adj* Noun* @-> "<NP>" ... "</NP>"`

where `@->` is the left-to-right longest-match replace operator ([2]). Given appropriate definitions for Det, Adj, and N, the above expression compiles into a transducer inserts XML tags around noun phrases. For example, it maps

Example 2. `children are playing on a jungle gym`

into

Example 3. `<NP>children</N> are playing on <NP>a jungle gym</NP>.`

While this method works well for many purposes such as the recognition of dates, phone numbers, addresses, it becomes impractical when the number of words in the component expressions of the pattern increases beyond a few thousand words as the case would be in a real noun phrase recognizer.

The principal reason for the blow-up is that the `@->` operator encodes the longest-match constraint in the state and arc space of the resulting transducer. The new pattern matching method that is implemented in fst enforces the longest match constraint in the runtime algorithm and not in the physical network. I will outline the new pattern matching algorithm and illustrate its benefits.

References

1. Beesley, K.R., Karttunen, L.J.: Finite State Morphology. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, CA (2003)
2. Karttunen, L.J.: Directed Replacement. In Joshi, A.K., Palmer, M.S., eds.: Proceedings of the 34th annual meeting of the Association for Computational Linguistics. Annual Meeting of the ACL, Morristown, NJ, Association for Computational Linguistics (1996) 108–115

Advances in Automata Implementation Techniques (Abstract)

Bruce W. Watson

Department of Computer Science, University of Pretoria

1 Introduction

In this abstract, I give a brief overview of the latest issues and advances in automata (state machine) implementations. The full material (available from me by email) was originally presented in Potsdam at the FSMNLP Conference in September 2007. Contemporary automata toolkits[1] are applied in areas as diverse as computational linguistics, network security, text indexing, compression, and parallel/concurrent systems. Such implementations typically have three usage scenarios:

- *Compilation* from a regular expression to an automaton.
- *Minimization* of an automaton.
- *Execution* of the automaton on an input string.

The implementations can be improved in the following areas:

1. *Expressive power and succinctness.* Using the more exotic regular operators (such as intersection, negation, etc.) can make a regular expression exponentially more succinct.
2. *Memory consumption.*
3. *Running time.*
4. *Hardware utilization.* Contemporary CPU's include wide bit-wise operators, large memories and potentially FPGA's ('field-programmable gate arrays'), all of which can be reconfigured for highly parallel operations.

These particular opportunities are arising for several reasons: CPU's are not getting much faster (in terms of clock-speed), but they are getting *wider* (e.g. multi-core CPU's, very wide bit-wise operations); main-memory sizes are growing, but cache memory is not, meaning memory-access locality is as important as ever; reconfigurable hardware (e.g. complex multi-core graphics cards from NVidia and ATI, for FPGA's) are becoming the norm in high-end computers. All of these aspects are still underutilized in automata implementations.

Here, I pay particular attention to compilation, though minimization and execution are equally important phases. Depending on the length of the input being processed by an automaton, the execution phase consumes the majority of the time. As such, efficient implementations in hardware are within reach.

[1] Examples of such footnotes include FIRE Engine, Grail, Vaucanson, FST, etc.

2 Compiling (constructing) automata

Automata compilation algorithms fall into two categories:

- *Inductive* constructions build up an automaton based on the structure of the input regular expression: simple automata are used for the atomic regular expressions, and each expression operator gives rise to a constructive operator on automata. As a result, 'compiling' is simply constructing the homomorphic image of the input regular expression. Such constructions have a number of advantages:
  - The automaton's structure reflects the underlying expression's structure.
  - The subautomata (corresponding to subexpressions) can be constructed independently.
  - Shared subexpressions need only be constructed once.

  Similarly, they have some disadvantages:
  - Shared subexpressions do not lead to shared subautomata — often leading to nonminimal automata.
  - Dead (unreachable) states may be constructed for some regular expressions.
  - Exotic regular operators (negation, etc.) are extremely difficult to implement inductively.
- *Reachability* constructions being with a just a few states (usually only one — the start state), and use graph-reachability algorithms to construct the remainder of the automaton. Such constructions often use *derivatives* (also known as *continuations*). The advantages of reachability constructions are:
  - No dead states are constructed.
  - Shared subexpressions are handled only once, leading to automata that are smaller than with inductive constructions.
  - Exotic regular operators are handled extremely easily (indeed, all sixteen Boolean regular operators are handled 'for free' in derivative-based reachability constructions).

  The disadvantages are:
  - A constructed automaton displays virtually no structure which is recognizable from the input regular expression, especially when exotic operators are used.
  - Subautomaton sharing may occur, but is difficult to identify.

Regardless of which construction style is used, there are only a few algorithmic optimization that are applicable:

- *Incremental* algorithms involve doing minimal recomputation of the output when the input changes — usually accomplished by saving some of the intermediate computations. Numerous incremental minimization algorithms are already known, including algorithms which can be halted at any point yielding a partially-minimized automaton ([1]). Incremental inductive construction is easily implemented thanks to the homomorphic nature of the algorithm; by contrast, incremental reachability construction is an important area of future work.

– *Parallel* algorithms are a new imperative, based on trends in hardware. Recent work has yielded parallel automaton minimization algorithms. For example, the algorithm given in [2] checks equivalence of states in separate threads; with enough available threads, this algorithm can minimize in linear time. Parallel constructions are straightforward:
  - Parallel inductive construction can be done with separate threads dealing with the subexpressions. A linear speedup is possible, with enough threads.
  - Parallel reachability construction can be done with separate threads dealing with the out-transitions of each new state in the reachability graph. The performance improvement is somewhat sensitive to the structure of the transition graph (in the worst case, it leads to no speedup).

Acknowledgements: I am particularly thankful to the FSMNLP 2007 organizers who helped me tremendously in forming my thoughts on this topic.

References

1. Watson, B.W., Daciuk, J.: An Efficient Incremental DFA Minimization Algorithm. Natural Language Engineering **9** (2003) 49–64
2. Strauss, T., Kourie, D.G., Watson, B.W.: A Concurrent Specification of Brzozowski's DFA Construction Algorithm. International Journal of Foundations of Computer Science **19**(1) (2008) 125–135

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 $f_{A_2}{:}{A_1}^* \rightarrow {A_1}^*$ defined as $f_{A_2}(U) = \{f(a_0)f(a_1)\ldots f(a_{m-1}) \mid a_0a_1\ldots a_{m-1} \in U \wedge a_0, a_1, \ldots, a_{m-1} \in A_1\}$ 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 $\diamond_0{:}\diamond_0, \diamond_1{:}\diamond_1, \diamond_2{:}\diamond_2, \ldots, \diamond_s{:}\diamond_s$ 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 $u = a_0{:}b_0 a_1{:}b_1 \ldots a_{m-1}{:}b_{m-1} \subseteq (A_1{:}A_2)^*$ is called a *symbol-pair string* and analyzed alternatively as a string pair $(x_1, x_2) = (a_0 a_1 \ldots a_{m-1}, b_0 b_1 \ldots b_{m-1})$. 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* $\pi_1, \pi_2 : \Pi^* \rightarrow \Pi^*$ 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: $a \rightarrow b = (\neg a) \vee b$. 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:

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

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 $a \rightarrow b = (\neg a) \vee b$ 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 $P, S \subseteq \Pi^*$:

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

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

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

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:

$$...\_...\Leftrightarrow ...\epsilon\_\epsilon...; \qquad \#\Pi^*L\_...\Leftrightarrow L\_...; \qquad ...\_R\Pi^*\#\Leftrightarrow ...\_R. \quad (5)$$

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

$$X \Rightarrow \#L_1\_R_1\#, \ldots, \#L_n\_R_n\#. \quad (6)$$

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]:

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

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:

$$\begin{aligned} X \Rightarrow \#L_1\_R_1\#, \#L_2\_R_2\# \stackrel{\text{def}}{=} \\ \textit{if-P-then-S}(\Pi^* X, R_1 \cup R_2) \cap \textit{if-S-then-P}(L_1, X(R_1 \backslash R_2)) \ \cap \\ \textit{if-S-then-P}(L_2, X(R_2 \backslash R_1)) \cap \textit{if-S-then-P}(L_1 \cup L_2, X(R_1 \cap R_2)). \end{aligned} \tag{8}$$

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 $0 \leq i \leq j \leq m$. Variables $i, j, k, \cdots \in \mathbb{N}$ 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 $\phi_{\Pi^*}(b, e) = (\forall b \leq i \leq e{-}1) \vee_{c \in \Pi} \phi_c(i, i+1)$. 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 $\phi_{U \setminus V}(b, e) = \phi_U(b, e) \wedge \neg \phi_V(b, e)$.

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

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

With Range Variables The signature of $FO[<]$ can be extended with position range variables $\mathbf{v}, \mathbf{x}, \mathbf{y}, \cdots \in \mathbb{N} \times \mathbb{N}$ each of which is a pair of type $[b, e]$ where $b \leq e$. The variables denote substrings $w[b_\mathbf{v}, e_\mathbf{v}], w[b_\mathbf{x}, e_\mathbf{x}], w[b_\mathbf{y}, e_\mathbf{y}], \ldots$. 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

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

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 $\phi_X(b, e)$, $\phi_{L_1}(b, e), \phi_{R_1}(b, e), \ldots \phi_{L_n}(b, e), \phi_{R_n}(b, e)$.

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

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

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 $\overset{\Pi,k,M}{\Rightarrow}\colon 2^{(\Pi^* M \Pi^*)^{\leq k}} \times 2^{(\Pi^* M \Pi^*)^*} \to 2^{\Pi^*}$ 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':

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

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 $[U_1 \cup \ldots U_r \overset{\Pi,k,M}{\Rightarrow} U_{r+1} \cup \ldots U'_p]$ is definable in $FO[<]$. This holds because we have the equation

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

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 $d_M : (\Pi^* M \Pi^*)^* \rightarrow \Pi^*$ 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 $\mathit{nowhere}(X) \overset{\text{def}}{=} [\Pi^* X \Pi^* \overset{\Pi,0,M}{\Rightarrow} \emptyset]$.

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

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

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

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

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

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

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

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

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

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

5.2 Decomposing into Conjunctive Constraints

It is not always practical to evaluate a GR operation $W \stackrel{\Pi,k,M}{\Rightarrow} W'$ 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

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

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

$$[W_1 \overset{\Pi,k,M}{\Rightarrow} W'_1] \cap [W_2 \overset{\Pi,k,M}{\Rightarrow} W'_2] = [(W_1 \cup W_2) \overset{\Pi,k,M}{\Rightarrow} (W'_1 \cup W'_2)] \quad \text{when } W_1 \cap W'_2 \subseteq W_1{}' \text{ and } W_2 \cap W_1{}' \subseteq W_2{}'; \tag{24}$$

$$[W_1 \overset{\Pi,k,M}{\Rightarrow} W'_1] \barwedge [W_2 \overset{\Pi,k,M}{\Rightarrow} W'_2] \overset{\text{def}}{=} [(W_1 \cup W_2) \overset{\Pi,k,M}{\Rightarrow} ((W_1 \cap W'_1) \cup (W_2 \cap W'_2))]. \tag{25}$$

In (25), the new operator $\barwedge$ 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

$$[X \texttt{<=>} \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \#] \overset{\text{def}}{=} [C \overset{\Pi,2,M}{\Rightarrow} \cup_{i=1}{}^{n} L_i \diamond_1 X \diamond_1 R_i] \quad \text{where } C = (\cup_{i=1}{}^{n} L_i \diamond_1 \pi_1{}^{-1}(\pi_1(X)) \diamond_1 R_i) \cup (\Pi^* \diamond_1 X \diamond_1 \Pi^*). \tag{29}$$

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:

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

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

$$X{=}[\Pi^* \nu_{2,1}(\epsilon \cup \Pi)\Pi^* \overset{\Pi,2,M}{\Rightarrow} \nu_{\star,1}(X)] \text{ where } \nu_{2,j}(W){=}{d_{\{\diamond_1,\ldots,\diamond_j\}}}^{-1}(W) \tag{32}$$
$$\cap\, (\Pi^* \diamond_j \Pi^* \diamond_j \Pi^*)$$
$$\text{and } \nu_{\star,j}(W){=}{d_{\{\diamond_1,\ldots,\diamond_j\}}}^{-1}(W). \tag{33}$$

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

$$X' = [\nu_{1,1}'(\Pi^+) \overset{\Pi,1,M}{\Rightarrow} \Pi^* \nu_{\star,1}'(X)\Pi^*] {=} [\Pi^* \nu_{2,1}(\Pi)\Pi^* \overset{\Pi,2,M}{\Rightarrow} \Pi^* \nu_{\star,1}(X)\Pi^*]. \tag{34}$$

Every string in the result X' will now have a *string covering* that consists of possibly overlapping factors X. For example, if $X = \{\texttt{autom}, \texttt{mate}, \texttt{eria}\}$ 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

$$[X_1 \texttt{=>} \#L_1 \_ R_1 \#] \barwedge \ldots \barwedge [X_n \texttt{=>} \#L_n \_ R_n \#] = [W_1 \overset{\Pi,2,M}{\Rightarrow} S_1] \qquad (35)$$
$$\text{where } W_1 = \Pi^* {\nu_{1,1}}'(\Pi)\Pi^* \text{ and } S_1 = \cup_{i=1}^{n} L_i {\nu_{\star,1}}'(X_i) R_i.$$

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, (36) 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

$$W_1 = \Pi^* \nu_{2,1}(\Pi \cup \cup_{i=1}{}^{n} X_i)\Pi^*;\ S_1 = \cup_{i=1}{}^{n} L_i\, \nu_{\star,1}(X_i)\, R_i. \tag{36}$$

6.3 Coercions

A change such as `(reduce+ation):(reduc000tion)` 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

$$\begin{gathered}[X_1 \texttt{<=} \# L_1 \_ R_1 \#] \fatbslash \ldots \fatbslash [X_p \texttt{<=} \# L_p \_ R_p \#] \stackrel{\text{def}}{=} [W_2 \overset{\Pi,2,M}{\Rightarrow} S_2] \\ \text{where } W_2 = \cup_{i=1}{}^{p} L_i\, \nu_{2,2}(\pi_1{}^{-1}(\pi_1(X_i)))\, R_i;\ \ S_2 = \cup_{i=1}{}^{p} L_i\, \nu_{\star,2}(X_i)\, R_i.\end{gathered} \tag{37}$$

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 ∪ 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

$$W_2 = \cup_{i=1}{}^{p} L_i\, \nu_{2,l_i}(\pi_1{}^{-1}(\pi_1(X_i)))\, R_i; \quad S_2 = \cup_{i=1}{}^{p} L_i\, \nu_{\star,l_i}(X_i)\, R_i. \tag{38}$$

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

$$[W_1 \overset{\Pi,2,M}{\Rightarrow} S_1] \Cap [W_2 \overset{\Pi,2,M}{\Rightarrow} S_2]. \tag{39}$$

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

$$\textsc{Gen}_{\text{core}} = [\Pi \ \texttt{=>}\ ] \Cap [I^* \ \texttt{=>}\ \_] \tag{40}$$

$$[X' \ \texttt{(=>)}\ \#L_1 \_ R_1 \#, \ldots \#L_n \_ R_n \#] \overset{\text{def}}{=} [X' \ \texttt{=>}\ \#L_1{}' \_ R_1{}' \#, \ldots \#L_n{}' \_ R_n{}' \#] \text{ where}$$
$$X' \subseteq B_L X B_R;\ X, L_i, R_i \subseteq (\Pi \backslash B)^*, L_i{}' = d_B{}^{-1}(L_i), R_i{}' = d_B{}^{-1}(R_i) \text{ for every } i. \tag{41}$$

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 $\{<, >\}$ is the language D_1 generated by the context-free grammar with a single nonterminal symbol S, two terminal symbols $<, >$, and productions $S \to S<S>S$ 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 $<$ and $>$. 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 $\Delta = {d_{\Pi\backslash B}}^{-1}(D_1{}')$ that contains also freely occurring non-bracket symbols.

Limited Bracketing The approximation $\Delta_i \subset {d_{\Pi\backslash B}}^{-1}(D_1{}')$, where $i \in \mathbb{N}$, can be obtained by induction on the *bracketing depth* i using a nested GR as follows:

$$\Delta_0 = \textit{match-L-R}(\Pi^* B_L, \emptyset, B_R \Pi^*) = W \overset{\Pi,1,M}{\Rightarrow} \emptyset$$
$$\text{where } W = (\Pi^* B_L \diamond_1 \Pi^*) \cup (\Pi^* \diamond_2 B_R \Pi^*); \tag{42}$$

$$\Delta_i = \textit{match-L-R}(\Pi^* B_R, \Delta_{i-1}, B_R \Pi^*) = W \overset{\Pi,1,M}{\Rightarrow} f_M(\Pi^* B_L \diamond_1 \Delta_{i-1} \diamond_2 B_R \Pi^*)$$
$$\text{where } i > 0 \text{ and } W = (\Pi^* B_L \diamond_1 \Pi^*) \cup (\Pi^* \diamond_2 B_R \Pi^*). \tag{43}$$

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 (43) 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 → SUBJ ⋆ OBJ].

$$\textbf{T1A}: \quad \Pi^* \, \nu_{1,1}{}'(\Pi) \, \Pi^* \overset{\Pi,1,M}{\Rightarrow} \Pi^* \, \nu_{\star,1}{}'(<_{\mathrm{S}}<_{\mathrm{NP}} \cup >_{\mathrm{NP}}<_{\mathrm{VP}} \cup >_{\mathrm{VP}}<_{\mathrm{S}}) \, \Pi^* \tag{44}$$

$$\textbf{T1B}: \quad \Pi^* \, \nu_{1,1}{}'(\Pi) \, \Pi^* \overset{\Pi,1,M}{\Rightarrow} \Pi^* \, \nu_{\star,1}{}'(>_{\overline{\mathrm{SUBJ}}} \; hit \; <_{\overline{\mathrm{OBJ}}}) \, \Pi^* \tag{45}$$

$$\textbf{T2A}: \quad \Pi^*{\diamond_1}<_{\mathrm{NP}}\Delta>_{\mathrm{NP}}{\diamond_1}\Pi^* \overset{\Pi,2,M}{\Rightarrow} \Pi^*<_{\mathrm{S}}{\diamond_1}<_{\mathrm{NP}}\Delta>_{\mathrm{NP}}{\diamond_1}<_{\mathrm{VP}}\Delta>_{\mathrm{VP}}>_{\mathrm{S}}\Pi^* \tag{46}$$

$$\textbf{T2B}: \quad \Pi^*<_{\mathrm{S}}{\diamond_1}\Delta{\diamond_1}>_{\mathrm{S}}\Pi^* \overset{\Pi,2,M}{\Rightarrow} \Pi^*<_{\mathrm{S}}{\diamond_1}<_{\mathrm{NP}}\Delta>_{\mathrm{NP}}<_{\mathrm{VP}}\Delta>_{\mathrm{VP}}{\diamond_1}>_{\mathrm{S}}\Pi^* \tag{47}$$

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 $\psi_1 : \Pi^* \to {\Sigma_1}^*$ is the *input homomorphism*, $P \subseteq \Pi^*$ is the *pivot*, and $\psi_2 : \Pi^* \to {\Sigma_2}^*$ 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

$$\begin{aligned} X \texttt{ (->) } \#L_1 \_ R_1 \#, \ldots \#L_n \_ R_n \# &\stackrel{\text{def}}{=} \beta(\textsc{Gen}_G) \\ \text{where } \textsc{Gen}_G = \textsc{Gen}_{\text{core}} \pitchfork [\texttt{<}X\texttt{> (=>) } \#L_1 \_ R_1 \#, \ldots \#L_n \_ R_n \#]. \end{aligned} \tag{48}$$

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 $[\texttt{a:b (->) } \#\Pi^* {\pi_2}^{-1}(\texttt{c}) \_ {\pi_1}^{-1}(\texttt{d})\Pi^*\#]$.

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

$$T_{\text{most}+} \stackrel{\text{def}}{=} \{(\pi_1(w), \pi_2(w)) | w \in \left(B_L{:}0\, \Sigma^+ B_R{:}0 \cup \Sigma \cup B\right)^*\} \tag{49}$$

$$T_{\text{lest},B'} \stackrel{\text{def}}{=} \{(w, w') | w, w' \in I^*, d_B(w) = d_B(w'), w \notin I^* B' I^*, w' \in I^* B' I^*\} \tag{50}$$

$$T_{\text{lr}} \stackrel{\text{def}}{=} \{(vby, vau) | v, y, u \in I^*, a \in \Pi \setminus B, b \in B_L, d_B(y) = d_B(au)\} \tag{51}$$

$$T_{\text{lrlong}} \stackrel{\text{def}}{=} \{(vau, vby) | v, u, y \in I^*, a \in \Pi \setminus B, b \in B_R, d_B(y) = d_B(au)\}. \tag{52}$$

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

$$\textsc{Gen}_G \texttt{ r-glc } T \stackrel{\text{def}}{=} \{w \in \textsc{Gen}_G \mid \neg\exists w'(w' \in \textsc{Gen}_G \wedge (\pi_1(w), \pi_1(w')) \in T)\}. \quad (53)$$

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:

$$X \texttt{ -> } \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \# \stackrel{\text{def}}{=} \beta(\textsc{Gen}'_G \texttt{ r-glc } (T_{\text{most}+} \cup T_{\text{lest},B_2})) \quad (54)$$

$$X \texttt{ @-> } \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \# \stackrel{\text{def}}{=} \beta(\textsc{Gen}_G \texttt{ r-glc } (T_{\text{lr}} \cup T_{\text{lrlong}})) \quad (55)$$

$$\text{where } \textsc{Gen}_G = [\texttt{<}_1 \quad X \quad \texttt{>}_1 \ \texttt{(=>)} \ \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \#] \Cap \textsc{Gen}_{\text{core}}$$

$$\text{and } \textsc{Gen}'_G = [\texttt{<}_2 \pi_1(X) \texttt{>}_2 \ \texttt{(=>)} \ \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \#] \Cap \textsc{Gen}_G.$$

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 (54) and (55) is structured in a similar fashion as Optimality Theory [39]. The roles of the candidate generator language $\textsc{Gen}_G$, the constraint component Con and the transformation β are outlined as follows:

$$\beta(\textsc{Gen}_G \circ \textsc{Con}) = \beta(\textsc{Gen}_G \texttt{ glc}_1 \ \textsc{Con}_1 \ldots \texttt{ glc}_c \ \textsc{Con}_c) \quad (56)$$

$$\text{where } \texttt{glc}_i \in \{\texttt{glc}, \texttt{r-glc}, \texttt{b-glc}\} \text{ is left associative.}$$

It is interesting that the structure of (56) 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 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

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

very simple structure. Accordingly, the components in (56) are relations that are contrasted as follows:

$$\text{stateless } \beta \textit{ vs. } \text{star-free same-length } \textsc{Gen}_G \textit{ vs. } \text{regular } \textsc{Con}. \tag{57}$$

We believe that (56) 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:

$$\begin{aligned}
\Pi^* &= \overline{\{a\} - \{a\}} && : \mathbf{B}_0 \\
\Delta_0 &= \overline{\Pi^* \{<, >\} \Pi^*} && : \mathbf{B}_1 \\
\lambda_i &= B_L \Delta_{i-1} B_L \Delta_{i-2} B_L \dots \Delta_1 B_L \Delta_0 B_L && : \mathbf{M}_i \\
\rho_i &= B_R \Delta_0 B_R \Delta_1 B_R \dots \Delta_{i-2} B_R \Delta_{i-1} B_R && : \mathbf{M}_i \\
\Delta_i &= \overline{\Pi^* \lambda_i \Pi^*} \cap \overline{\Pi^* \rho_i \Pi^*} \cap \overline{\Delta_{i-1} B_R \Pi^*} \cap \overline{\Pi^* B_L \Delta_{i-1}} && : \mathbf{B}_{i+1}.
\end{aligned} \tag{58}$$

The GR-based construction (43) 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 (58). The upper bounds apply to the construction (43) as follows:

$$\begin{aligned}
\mathbf{B}_0 :&\quad \Pi^* = \overline{\{a\} - \{a\}} \\
\mathbf{B}_1 :&\quad W = (\Pi^* B_L \diamond_1 \Pi^*) \cup (\Pi^* \diamond_2 B_R \Pi^*);\ \ \Delta_0 = \Pi^* \backslash f_M(W) \\
\mathbf{M}_{i+1} :&\quad W'_i = \Pi^* B_L \diamond_1 \Delta_i \diamond_2 B_R \Pi^* \\
\mathbf{B}_{i+1} :&\quad \Delta_i = \Pi^* \backslash f_M(W \backslash f_M(W'_{i-1})).
\end{aligned} \tag{59}$$

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:

$$\begin{aligned}
L \circ [L' \cap (W \overset{\Pi,2,M}{\Rightarrow} W')] = L \circ [\diamond_0 \Pi^* \cup W \overset{\Pi,2,M}{\Rightarrow} \diamond_0 L' \cup W'] = L \circ [\Pi^* \backslash d_M(W'')] \\
\text{where } W'' = (\diamond_0 \Pi^* \cup W) \backslash (\diamond_0 L' \cup W').
\end{aligned} \tag{60}$$

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 $\overline{L'} = \Pi^* \backslash L'$) to the minuend ($\diamond_0 \Pi^* \cup W$). In addition, these bad strings can contain diamonds freely. This change can make W'' larger, but $d_M(W'')$ remains the same:

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

A non-deterministic automaton recognizing $(\diamond_0 \Pi^* \cup W \cup {d_0}^{-1}(\overline{L'}))$ reaches a final loop when it recognizes a marked string $w \in (\Pi \cup M)^*$ for which $w(\Pi \cup M)^* \cap {d_0}^{-1}(L') = \emptyset$. 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 $W' = \Pi^* W'$. we can, however, postpone the determinization of the subtrahend by using de Morgan's law, which allows us, in a way, *subtract during determinization*:

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

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 $\diamond_0 \Pi^* \cup W \cup \overline{L'}$ is a bottleneck, because typically $W = \Pi^* W$. To address this problem, we can specialize W and rewrite the sub-expression as $\diamond_0 \Pi^* \cup ({d_M}^{-1}(L') \cap W) \cup \overline{L'}$.

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

$$\Pi^* \backslash d_M(W'') = \Pi^* \backslash d_M(\overline{W'''})$$
$$\text{where } W''' = \overline{\diamond_0 \Pi^* \cup ({d_M}^{-1}(L') \cap W) \cup {d_0}^{-1}(\overline{L'})} \cup \diamond_0 L' \cup ({d_M}^{-1}(L') \cap W'). \tag{63}$$

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 (60) 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., Žďárek, 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 (forthcoming 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., Žďárek, 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)

Part II

Regular Papers

Asymmetric Term Alignment with Selective Contiguity Constraints by Multi-Tape Automata

Mădălina Barbaiani[1,2], Nicola Cancedda[1], Chris Dance[1], Szilárd Fazekas[1,2], Tamás Gaál[1], and Éric Gaussier[1,3]

[1] Xerox Research Centre Europe – Grenoble Laboratory, 6 chemin de Maupertuis, 38240 Meylan, France
[2] Department of Computer Science, Rovira i Virgili University, Tarragona, Spain
[3] Joseph Fourier University, Grenoble, France

Abstract. This article describes a HMM-based word-alignment method that can selectively enforce a *contiguity constraint*. This method has a direct application in the extraction of a bilingual terminological lexicon from a parallel corpus, but can also be used as a preliminary step for the extraction of phrase pairs in a Phrase-Based Statistical Machine Translation system. Contiguous *source* words composing terms are aligned to contiguous *target* language words. The HMM is transformed into a Weighted Finite State Transducer (WFST) and contiguity constraints are enforced by specific multi-tape WFSTs. The proposed method is especially suited when basic linguistic resources (morphological analyzer, part-of-speech taggers and term extractors) are available for the source language only.

1 Introduction

Specialized bilingual terminologies are essential to technical translators for ensuring correctness and consistency in large translation projects. This is attested by the presence, in professional translation environments, of tools to collect and navigate terminologies. Several methods for extracting multilingual terminologies from parallel document collections have been proposed [1–6]. Unlike these methods, the method described here does not require the availability of a morphological analyzer and a POS tagger for both languages.

Besides lexicon and terminology extraction, word alignment is also an essential step in most Statistical Machine Translation approaches, as well as in the projection of linguistic resources across parallel corpora. Like existing methods for performing word alignment based on Hidden Markov Models (HMMs) [7], ours builds an HMM with one state for each words in a source language emitting words in the target language, and associates alignments with paths through the HMM. Our method allows the enforcement of the constraint that target words aligned to a same source term should be contiguous (*contiguity constraint*), thus restricting the alignment search space according to a generally acknowledged linguistic principle, and leading to improved lexica. This is done without imposing that alignments be *monotonic*, i.e. that word order is fully preserved in

the two languages. The method uses an implementation of weighted multi-tape finite state automata [8–10].

This paper focuses on the task of word alignment as it is critical in performing automatic terminology extraction. Existing methods to align documents at the sentence level [11–14], to identify candidate terms [15] and to extract a terminological dictionary from the word-aligned corpus can be used in conjunction with what we present here to complete the overall task.

2 Word Alignment, Sentence-Pair HMM

Consider a sentence pair $({e_1}^I, {f_1}^J)$, ${e_1}^I$ being the English sentence and ${f_1}^J$ its foreign translation consisting of I and J words, respectively[4]. Elements in ${e_1}^I$ can be either (possibly multi-word) terms or individual words occurring outside term boundaries. For reasons that will become clear soon, we will call such elements *states*. Let ${a_1}^J$ be the sequence of alignment variables, with $a_j = i$ if and only if the foreign word f_j is aligned to the English word/term e_i. We will restrict our attention to alignments in which a foreign word must be assigned to one and only one English state. Our objective is to determine ${a_1}^{J*}$ such that:

$$ {a_1}^{J*} = \text{argmax}_{{a_1}^J}\{P({a_1}^J|{f_1}^J, {e_1}^I)\} \tag{1} $$

Applying Bayes' rule and making some conditional independence assumptions, this becomes:

$$ {a_1}^{J*} = \text{argmax}_{{a_1}^J}\{\prod_{j=1}^{J} p(f_j|e_{a_j})p(a_j|a_{j-1})\} \tag{2} $$

We can model our conditional probability distribution by means of a Hidden Markov Model (HMM) with states $e_i, i = 1 \ldots I$ and emitting in the foreign alphabet Σ_F.[5]

The method needs estimates of word translation probabilities in the form $p(f|e)$, that is, for each English word e, the probability that it will be translated with word f. These can be achieved, for example, by the use of an Expectation-Maximization algorithm [16] using a translation model like in [17]. Further preprocessing may include lowercasing, lemmatisation, stemming, and the decomposition of compound words.

Emission probabilities can be estimated from the word translation probabilities. For transition probabilities, we would prefer to favor monotonic alignments but allow other arrangements, too: a two-component discrete Gaussian mixture is appropriate to model this. The states of the HMM are either contiguous multi-word terms or out-of-term words. Figure 1 shows an example.

[4] Following a convention similar to the standard one in the MT community, in the following we will assume that English is the source language, for which resources are available, and a *foreign* language is what we previously referred to as the target language. Needless to say, the method is, in its elements described here, independent of the actual language pair.

[5] We use throughout the term *alphabet* in the language theory sense of "set of symbols". In our case symbols will be words, not individual characters.

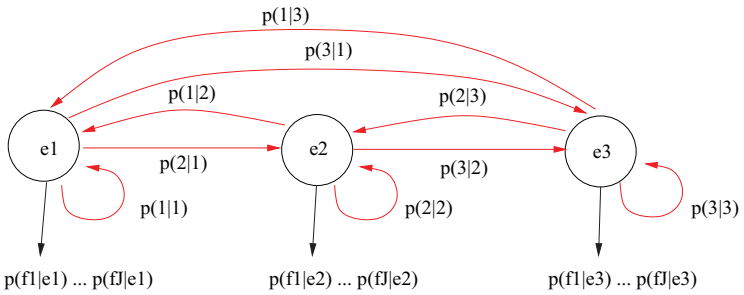


Fig. 1. HMM for three English terms; there is a single state for each term and the transition probabilities are modeled by two-component discrete Gaussian mixtures. I (3 here) and J are source- and target-language term indices, respectively.

The transformation of the HMM into a WFST of tropical semiring (W_{HMM}) is as follows:

- For every state of the HMM we create a state in the WFST
- Between any two states e_i and $e_{i'}$ (including $i = i'$) we add J transitions labeled by the foreign words on the input tape and i' on the output tape with weight $-\log p(i'|i) - \log p(f_k|e_{i'})$
- We create an initial state from which there will be a transition to every state e_i labeled f_1 with weight $-\log p(f_1|e_i)$
- Every state except the initial one is final.

A part of such an automaton is illustrated in Figure 2.

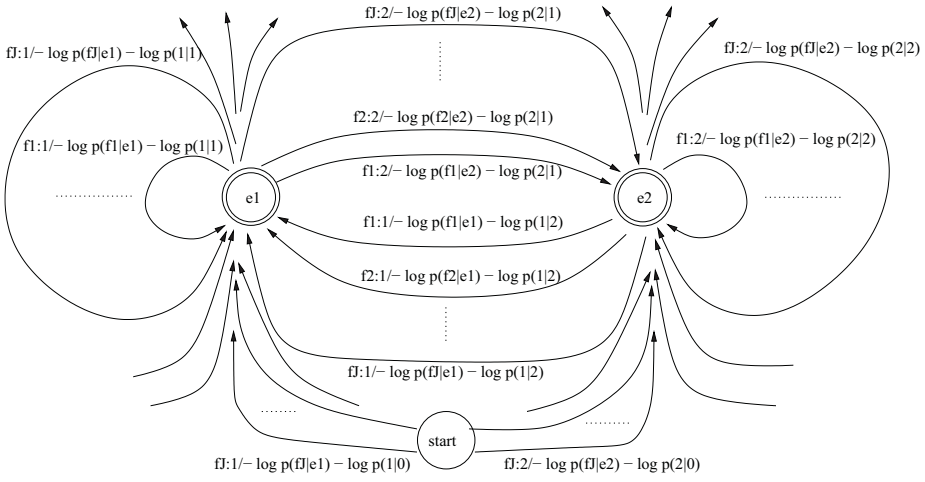


Fig. 2. Two states of the WFST built from the HMM, and their connection to the initial state.

3 Enforcing Contiguity

After turning our HMM into a weighted transducer, we can obtain the most probable alignment by applying Dijkstra's or Viterbi's shortest path algorithm. If we do so directly, with the weighted transducer, however, we are not guaranteed to obtain alignments respecting the contiguity constraint for terms. Formally:

An alignment ${a_1}^J$ *is contiguous if and only if there do not exist integers* i, j, k*, with* $1 \leq i < j < k \leq J$*, such that* $a_i = a_k = t$ *and* $a_j \neq t$*, for any* t *such that* e_t *is a term.*

The method we propose to solve this problem is inspired by [18]. In [18] the authors considered the problem of modeling a probability distribution over all alternative permutations of the chunks in a sequence: in our case, however, the foreign and the English sequence do not necessary have the same length, and the constraint we need to enforce is somewhat different.

Given a WFST encoding an HMM with I states, we build an automaton PE with one state for each pair of a subset of states (term states visited before the last) and a state (the last visited state if it is a term state, a wildcard symbol otherwise). The path-set in this automaton will be such that only sequences over $\{1 \dots I\}$ that do not contain discontinuous repetitions of term identifiers will be accepted:

$$PE = (Q_{PE}, \Sigma_{PE}, E_{PE}, I_{PE}, F_{PE})$$

Q_{PE}: $\{\text{start}\} \cup \{\langle A, \sigma \rangle \in 2^{\Sigma_t} \times \Sigma | \sigma_{PE} \notin A\}$

Σ_{PE}: $\{1, \dots, I\}$

$$\Sigma_{PE} = \Sigma_t \cup \Sigma_n,\ \Sigma_t \cap \Sigma_n = \emptyset$$

E_{PE}: $(\text{start}, \sigma, \langle \emptyset, \sigma \rangle)$

$$\forall \sigma \in \Sigma_{PE}$$

$(\langle A, \sigma \rangle, \sigma', \langle A \cup \{\sigma\}, \sigma' \rangle)$

$$\forall A \subseteq \Sigma_t, \forall \sigma \in \Sigma_t, \sigma \notin A, \forall \sigma' \in \Sigma_{PE}, \sigma' \notin A \cup \{\sigma\}$$

$(\langle A, \sigma \rangle, \sigma, \langle A, \sigma \rangle)$

$$\forall A \subseteq \Sigma_t, \forall \sigma \in \Sigma_{PE}, \sigma \notin A$$

$(\langle A, \sigma \rangle, \sigma', \langle A, \sigma' \rangle)$

$$\forall A \subseteq \Sigma_t, \forall \sigma \in \Sigma_n, \forall \sigma' \in \Sigma_{PE}, \sigma' \notin A \cup \{\sigma\}$$

I_{PE}: $\{\text{start}\}$

F_{PE}: $Q_{PE} \setminus \{\text{start}\}$

Edges in the first set (first line for E_{PE}) are used to match the first symbol in the input sentence. Edges in the second set are followed whenever a new symbol is aligned to a new term or out-of-term word, and the previous was aligned to a term. The third set covers the case when a symbol is aligned to the same term or out-of-term word as the symbol just before it. Finally, the fourth set covers transitions such that the current symbol is aligned to a new term or out-of-term word, and the previous was aligned to an out-of-term word.

By making every non-start state final and by introducing a special initial state, we obtain the automaton that accepts all and only the alignments satis-

fying the contiguity constraint. In the example of Fig. 3, the alignment 1122143 is accepted, while 12324 is not, since 2 is repeated twice separated by a 3. Notice that constructing the contiguity enforcing automaton can be done once, in advance, for all possible combinations of $|\Sigma_t|$ and $|\Sigma_n|$ present in the corpus, renaming states for specific source sentences.

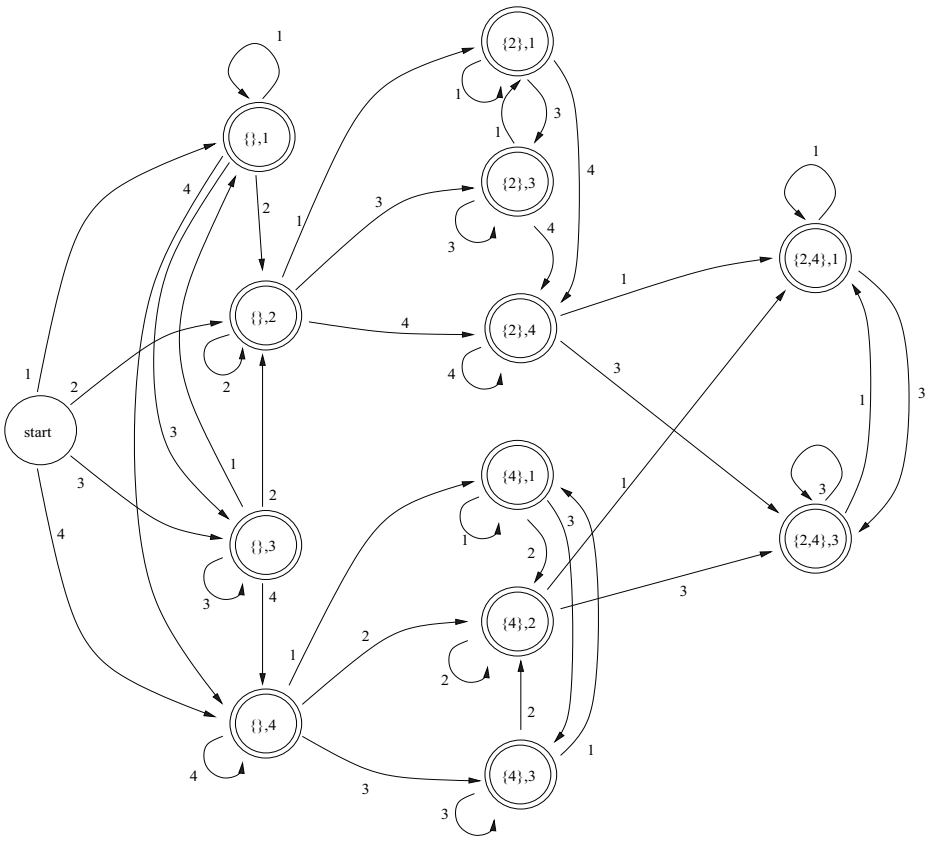


Fig. 3. Contiguity enforcing automaton for an English sentence with four states, two of which (e_2 and e_4) correspond to multi-word terms and two of which (e_1 and e_3) correspond to words occurring outside terms.

The composition of the contiguity enforcing automaton PE and the original WFST built from the HMM can be constructed directly by expanding every transition on symbol $\sigma \in \Sigma_{PE}$ in the contiguity enforcing automaton with $|\Sigma_F|$ transitions, one for each possible foreign word and assigning weights according to the appropriate transition and emission log-probabilities:

$$(\text{start}, \sigma, \langle \emptyset, \sigma \rangle)$$
$$\rightarrow (\text{start}, f : \sigma / -logP(f|e_\sigma) - logP(\sigma|0), \langle \emptyset, \sigma \rangle)$$
$$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_{PE}$$

$$(\langle A, \sigma \rangle, \sigma', \langle A \cup \{\sigma\}, \sigma' \rangle)$$
$$\rightarrow (\langle A, \sigma \rangle, f : \sigma' / -logP(f|e_{\sigma'}) - logP(\sigma'|\sigma), \langle A \cup \{\sigma\}, \sigma' \rangle)$$
$$\forall A \subseteq \Sigma_t, \forall f \in \Sigma_F, \forall \sigma \in \Sigma_t, \sigma \notin A, \forall \sigma' \in \Sigma_{PE}, \sigma' \notin A \cup \{\sigma\}$$

$$(\langle A, \sigma \rangle, \sigma, \langle A, \sigma \rangle)$$
$$\rightarrow (\langle A, \sigma \rangle, f : \sigma / -logP(f|e_\sigma) - logP(\sigma|\sigma), \langle A, \sigma \rangle)$$
$$\forall A \subseteq \Sigma_t, \forall f \in \Sigma_F, \forall \sigma \in \Sigma_{PE}, \sigma \notin A$$

$$(\langle A, \sigma \rangle, \sigma', \langle A, \sigma' \rangle)$$
$$\rightarrow (\langle A, \sigma \rangle, f : \sigma' / -logP(f|e_{\sigma'}) - logP(\sigma'|\sigma), \langle A, \sigma' \rangle)$$
$$\forall A \subseteq \Sigma_t, \forall f \in \Sigma_F, \forall \sigma \in \Sigma_n, \forall \sigma' \in \Sigma_{PE}, \sigma' \notin A \cup \{\sigma\}$$

where Σ_F denotes the set of distinct words in the foreign sentence to be aligned.

Overall computational complexity is $O(I^2 2^{|\Sigma_t|} J)$. The result, again, is the best path of the directly constructed $PE' = W_{HMM} \diamond PE$.

3.1 Lazy contiguity enforcing

In the previous section, the whole contiguity enforcing automaton is composed *a priori* with the HMM-derived WFST before the best-path algorithm is run. While this time-consuming operation ensures that the selected alignment will respect the contiguity constraint, it is actually pointless in the cases where the best path in the original HMM-derived WFST does not violate the constraint. A variation of the previous method thus consists in first running a best-path algorithm on the initial WFST, checking whether the constraint is violated for any term and, if it is, compose the WFST with a reduced contiguity-enforcing automaton limited to the terms for which contiguity was violated, and iterate (Algorithm 1). It is easily verified that the automaton in Fig.4 enforces the contiguity constraint for the term $\sigma \in \Sigma_t$

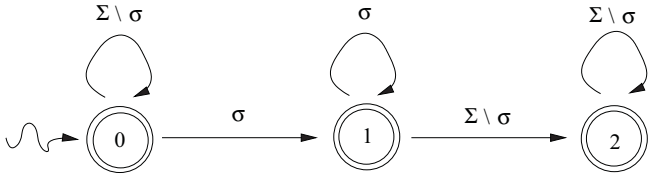


Fig. 4: The three-state automaton enforcing the constraint that whenever the state σ in the HMM-derived WFST is left, it is never entered again. For efficiency [10], every single arc with label $\Sigma \setminus \sigma$ represents a set of arcs, one for each $\sigma' \neq \sigma$.

Algorithm 1 The iterative algorithm that only composes the HMM-derived WFST with single-term contiguity checker for terms for which contiguity was violated.

$W \leftarrow W_{HMM}$
repeat
  $\pi \leftarrow$ Viterbi(F, W)
  $Z \leftarrow$ violate-contiguity(π, W)
  for all z in Z **do**
    $W \leftarrow W \diamond A_z$
until $Z = \emptyset$
return π

Each composition has the effect of doubling the number of states and edges. In the worst case, the algorithm is forced to compose again and again for enforcing contiguity one term at a time, and the finally resulting automaton is the same as what would be obtained by composing W_{HMM} directly with P. In this case there is no asymptotic penalty in the repeated composition itself, but we executed in vain $|\Sigma_t| - 1$ times the best-path algorithm, on WFSTs of exponentially increasing size. Notice, though, that the global asymptotic complexity for all repetitions of the best-path algorithm is the same as the complexity of the last iteration, because $\sum_{i=0}{}^{|\Sigma_t|} 2^i IJ = IJ(2^{|\Sigma_t|+1} - 1)$, and $\mathrm{O}(IJ(2^{|\Sigma_t|+1} - 1)) = \mathrm{O}(IJ2^{|\Sigma_t|})$. In other words, by performing composition lazily, we pay at most a constant factor in time, and nothing in space.

3.2 Local Reordering

While the permutation automaton is, in many cases, an appropriate solution, it has the drawback of growing exponentially in size with the number of source terms. The present section describes a method for enforcing the contiguity constraint which is not exponential in size in the length of the input and consists in a single best-search path on a multi-tape weighted finite-state transducer. The reduction in complexity is obtained at the price of allowing only *local reorderings* between the source and the target sequence. A permutation π is a local reordering with *jump length* m_j and *window size* m_w if and only if every element is at most m_j steps away from its position in the identity permutation (i.e. $|\pi_i - i| \leq m_j$) and every deviation from the identity permutation occurs within a subsequence of size at most m_w (i.e. the corresponding permutation matrix is block-diagonal with blocks of size at most m_w). For example, for maximal jump length $m_j = 2$ and window size $m_w = 3$ an acceptable permutation would be $(3, 2, 1, 4)$ but not $(3, 4, 1, 2)$. It is possible to write automatically automata to recognise sequences of jumps (i.e. $\pi_1 - 1, \pi_2 - 2, \ldots, \pi_I - I$) corresponding to local reorderings for arbitrary values of m_j and m_w. We will first describe how such a local reordering automata can be automatically generated, and then describe a method for compiling the original HMM into a new (multi-tape) WFST [8, 10] that represents the same probability distribution but outputs a sequence

of jumps as well as a sequence of visited states, and that can thus be joined with the local reordering automaton.

Algorithm 2 builds a local reordering automaton given the parameters m_j and m_w. Although the construction takes $O(m_w!m_w\log(m_w!m_w))$ time, where

Algorithm 2 *ReorderingAutomata*$(m_j, m_w)(R)$

```
Create state s_0, which will be the initial state and the only final state
Add a loop edge to s_0 labeled with 0
Generate the list Perm of all possible permutations of m_w many elements
Create empty list JumpSeq
for all π in Perm do
    tmp=""
    for all j in {1, 2, .., m_w} do
        if |π_j − j| ≤ m_j then
            Append π_j − j to tmp
    if length(tmp) = m_w then
        Add tmp to JumpSeq
for all i in JumpSeq do
    Strip leading and trailing 0s from i
    Create state s_{i,1}
    Add an arc going from s_0 to s_{i,1} labeled by i[1]
    for all j in {2, .., m_w − 1} do
        Create state s_{i,j}
        Add an arc going from s_{i,j−1} to s_{i,j} labeled by i[j]
    Add an arc going from s_{i,m_w−1} to s_0 labeled by i[m_w]
Minimize the automaton
```

complexity is dominated by the minimization operation, it needs be done once only for the desired (max. jump, window size) pair and does not need be repeated for each sentence. This algorithm was implemented and used in the preliminary experiments reported in Section 4. Figure 5 shows such an automaton. We note here that we have a more direct way of building the local reordering automaton, too.

In order to be able to use this automaton, we need to generate suitable input: the WFST built from HMM should output jumps between term positions instead of plain word/term identifiers like with the permutation automaton. Moreover, the reordering automaton only accepts (appropriately constrained) *permutations* of the state identifiers. There are two issues then that must be solved in order to use an approach based on the reordering automaton. The first is that we want to allow repeated visits to a same term, provided they are contiguous. The jump sequence will thus need to be generated from a term sequence from which contiguous repetitions are replaced by a single occurrence of a special term identifier. A second issue is that we might or might not be willing to accept the additional constraint that all English terms must be aligned to some foreign word. We will first present a solution for the case in which we assume that all

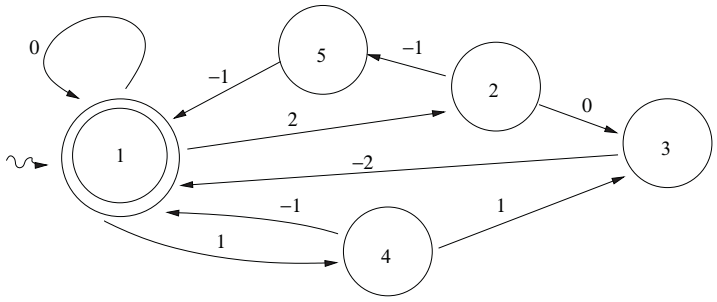


Fig. 5. The automaton produced by Algorithm 2 with parameters $m_j = 2$ and $m_w = 3$.

occurrences of terms on the English side have a counterpart on the foreign side, and then propose a modification that relaxes this assumption at the cost of introducing some complexity in the process.

Cascaded Automata with All English Terms Aligned In order to produce appropriate input for the reordering automaton, we add to the WFST derived from the original HMM an additional tape, besides the first reading the input sequence of foreign words and the second writing the sequence of English word/term identifiers corresponding to the alignment. A special term identifier ranging from 1 to the number of terms in the sentence (1 for e_2 and 2 for e_4 in the example) is written on the third tape every time a term-state is entered from a different state (Fig. 6). Formally:

$$H = (Q_H, \Sigma_H, E_H, I_H, F_H, A)$$

Q_H: $\{\text{start}\} \cup \{q_\sigma | \sigma \in \Sigma_H\}$
Σ_H : $\Sigma_{H1} \times \Sigma_{H2} \times \Sigma_{H3}$
Σ_{H1}: Σ_F
Σ_{H2}: $\{1, \ldots, I\}$ $\Sigma_{H2} = \Sigma_t \cup \Sigma_n$, $\Sigma_t \cap \Sigma_n = \emptyset$
Σ_{H3}: $\{1, \ldots, |\Sigma_t|\}$
E_H: $(\text{start}, f : \sigma : \epsilon, q_\sigma)/ - logP(f|e_\sigma) - logP(\sigma|0)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_n$
$(\text{start}, f : \sigma : t_\sigma, q_\sigma)/ - logP(f|e_\sigma) - logP(\sigma|0)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_t$
$(q_\sigma, f : \sigma' : \epsilon, q_{\sigma'})/ - logP(f|e_{\sigma'}) - logP(\sigma'|\sigma)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_{H2}, \forall \sigma' \in \Sigma_n$
$(q_\sigma, f : \sigma' : t_{\sigma'}, q_{\sigma'})/ - logP(f|e_{\sigma'}) - logP(\sigma'|\sigma)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_{H2}, \forall \sigma' \in \Sigma_t, \sigma \neq \sigma'$
$(q_\sigma, f : \sigma : \epsilon, q_\sigma)/ - logP(f|e_\sigma) - logP(\sigma|\sigma)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_t$
I_H: $\{\text{start}\}$
F_H: $Q_H \setminus \{\text{start}\}$

where $t_\sigma, \sigma \in \Sigma_t$ is a special term identifier in $\{1, \ldots, |\Sigma_t|\}$ and $A = \langle \mathbb{R} \cup \{+\infty\}, min, +, 0, +\infty \rangle$ is the tropical semiring.

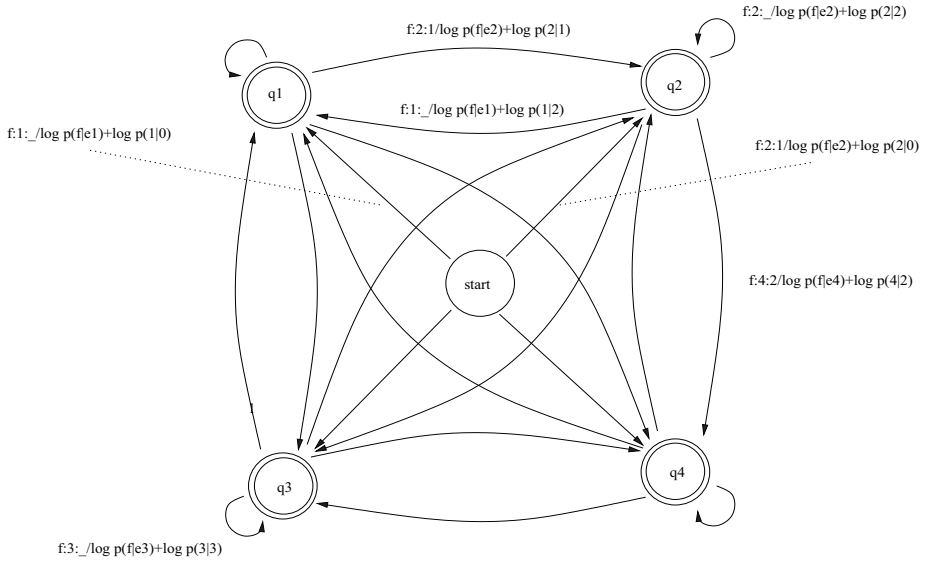


Fig. 6. An example of a 3-tape WFST encoding an HMM. e_2 and e_4 are terms, e_1 and e_3 are out-of-term words. Only one transition for every state pair is shown, while in the actual automaton there is one for every word f in the foreign sentence. Also, only some labels/weights are shown.

We introduced two simple auxiliary finite-state machines: the *identical permutation* automaton I and the *difference* automaton D: the former just encodes an increasing integer sequence $1, 2, \ldots, |\Sigma_t|$, while the latter has three tapes and puts on the third the difference of the first two.

By appropriately joining[6] the transducers defined so far we obtain a transducer accepting only alignments corresponding to local reorderings of the English terms,

$$G = ((H \bowtie_{\{3=1\}} D) \bowtie_{\{4=1\}} I) \bowtie_{\{5=1\}} R, \tag{3}$$

where the result G has five tapes: $H_1 : H_2 : H_3 = D_1 : D_2 = I_1 : D_3 = R_1$. The best alignment satisfying the contiguity constraint and representing a local reordering (with parameters m_j and m_w) is then simply obtained through a best-path search on G, on the tape G_2.

Cascaded Automata with Empty Alignments for Some English Terms The method described above imposes that all English terms are aligned. If we do

[6] Algorithms for joining and composing automata are described in [9].

not want to impose this additional constraint, we can still enforce the contiguity constraint with local reorderings, but at the price of making the method more complicated.

The reordering automaton recognises the language of all jump sequences corresponding to local reordering permutations. We generated the jump sequence by computing a symbol-wise difference between the sequence of term identifiers in the order they were entered and their identical permutations. We could do that simply through the D transducer because these two sequences have the same length. If now we want to relax the constraint that all English terms are aligned we need a way to "visit" all states of the HMM other than by aligning a foreign word to it. We can do this by introducing special transitions going into term states that do not "use" any input symbol. When assessing the probability of an alignment, this will alter the contribution of the transition probabilities. To account for this effect, we will thus create two separate WFSTs from the original HMM: one (E) will account for the emission probabilities only, and one will account for the transition probabilities (T). The former will accept the input sequence and will output on one of its tapes a sequence (with no contiguous repetitions) of identifiers of those visited states which are aligned to real foreign words only:

$$E = (Q_E, \Sigma_E, E_E, I_E, F_E, A)$$

Q_E: $\{\text{start}\} \cup \{q_\sigma | \sigma \in \Sigma_E\}$
Σ_E : $\Sigma_{E1} \times \Sigma_{E2} \times \Sigma_{E3}$
Σ_{E1}: Σ_F
Σ_{E2}: $\{1, \ldots, I\}$
$\Sigma_{E2} = \Sigma_t \cup \Sigma_n$, $\Sigma_t \cap \Sigma_n = \emptyset$
Σ_{E3}: $\{1, \ldots, |\Sigma_t|\}$
E_E: $(\text{start}, f : \sigma : \epsilon, q_\sigma)/ - logP(f|e_\sigma)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_n$
$(\text{start}, f : \sigma : t_\sigma, q_\sigma)/ - logP(f|e_\sigma)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_t$
$(\text{start}, \epsilon : \epsilon : t_\sigma, q_\sigma)/0$
$\forall \sigma \in \Sigma_t$
$(q_\sigma, f : \sigma' : \epsilon, q_{\sigma'})/ - logP(f|e_{\sigma'})$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_{E2}, \forall \sigma' \in \Sigma_n$
$(q_\sigma, f : \sigma' : t_{\sigma'}, q_{\sigma'})/ - logP(f|e_{\sigma'})$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_{E2}, \forall \sigma' \in \Sigma_t, \sigma \neq \sigma'$
$(q_\sigma, f : \sigma : \epsilon, q_\sigma)/ - logP(f|e_\sigma)$
$\forall f \in \Sigma_F, \forall \sigma \in \Sigma_t$
$(q_\sigma, \epsilon : \epsilon : t_\sigma, q_\sigma)/0$
$\forall \sigma \in \Sigma_t$
I_E: $\{\text{start}\}$
F_E: $Q_E \setminus \{\text{start}\}$

where $t_\sigma, \sigma \in \Sigma_t$ is a special term identifier in $\{1, \ldots, |\Sigma_t|\}$ and $A = \langle \mathbb{R} \cup \{+\infty\}, min, +, 0, +\infty \rangle$ is the tropical semiring.

The latter will compute the contribution of transition probabilities based on this state sequence:

$$T = (Q_T, \Sigma_T, E_T, I_T, F_T, A)$$

Q_T: $\{\text{start}\} \cup \{q_\sigma | \sigma \in \Sigma_T\}$
Σ_T : $\{1, \ldots, I\}$
E_T: $(\text{start}, \sigma, q_\sigma)/ - logP(\sigma|0)\ \forall \sigma \in \Sigma_T$
$(q_\sigma, \sigma', q_{\sigma'})/ - logP(\sigma'|\sigma)\ \forall \sigma, \sigma' \in \Sigma_T$
I_T: $\{\text{start}\}$
F_T: $Q_T \setminus \{\text{start}\}$

The result of joining these transducers with the D, I and R transducers (see before) is a transducer accepting only alignments corresponding to local reorderings of the English terms where some English terms can remain unaligned:

$$G' = (((E \bowtie_{\{2=1\}} T) \bowtie_{\{3=1\}} D) \bowtie_{\{4=1\}} I) \bowtie_{\{5=1\}} R \tag{4}$$

where the result G' has five tapes: $E_1 : E_2 = T1 : E_3 = D_1 : D_2 = I_1 : D_3 = R_1$. The best local reordering alignment (with m_j and m_w) satisfying the contiguity constraint is then obtained through a best-path search on G', on the tape G'_2.

4 Experiments

Some experiments applying the described word-alignment methods to multilingual terminology extraction were performed. 576 sentence pairs coming from Xerox manuals in English-German were annotated with 897 term boundaries and alignments by a native speaker of German. As a base for our projection, we first took manually identified English terms, then NPs extracted using patterns of parts-of-speech. For both cases, we aligned German words to English terms and out-of-term words according to some early variants of the methods described above:

- SELECT: lazy enforcement of the contiguity constraint to terms only;
- REORD: local reordering method with $m_j = 2$ and $m_w = 3$
- SYMMETRIC: the standard symmetric method by which term candidates are first identified in German using POS patterns and are then aligned, but without a heuristic that greedily extends term boundaries to improve boundary detection.

For each method we measured the number of exact matches, together with precision, recall and F-score at the word level according to the following definitions:

$$P = \frac{\sum_{i=1}^{n} p_i}{n}; \quad p_i = \frac{a_i}{c_i}; \quad R = \frac{\sum_{i=1}^{n} r_i}{n}; \quad r_i = \frac{a_i}{w_i}; \quad F = \frac{2PR}{P+R}$$

where c_i is the number of words identified as part of terms in sentence i, w_i is the number of words in terms in the reference for sentence i, and a_i is the number of correctly aligned in-term tokens in sentence i.

Test results are in table 1, where terms are all German gold standard terms for which a translation was provided or not in the candidate sentences.

method	**manual annotation**			
	exact matches	**P**	**R**	**F**
SELECT	291 (32.44%)	86.39%	64.51%	73.87%
REORD	258 (28.76%)	83.65%	67.83%	74.91%
SYMMETRIC	282 (31.44%)	97.41%	98.12%	97.76%
method	**automated annotation**			
	exact matches	**P**	**R**	**F**
SELECT	275 (30.66%)	86.69%	67.07%	75.63%
REORD	235 (26.20%)	83.38%	70.53%	76.42%
SYMMETRIC	205 (22.85%)	89.93%	96.04%	92.88%

Table 1. Preliminary experiment results.Total number of terms: 897, total number of sentences: 576.

From such preliminary experiments, it would seem that in the automated case, performance is comparable with those obtained with the symmetric method, recall is smaller, more resource-intensive, although this is somewhat underestimated due to the fact that an effective recall-oriented heuristic (although possibly precision-degrading) was not used.

We aligned 576 sentences in 6 hours when enforcing local reordering contiguity constraint and in 10 minutes for the selective lazy one. The standard symmetric alignment is much faster.

After these experiments, a Norwegian-English terminology containing 4211 candidate term pairs was extracted from a parallel corpus of 38487 sentence pairs from Xerox manuals using an early version of the lazy contiguity enforcement method.

5 Summary

We have shown that our method can be used to extract bilingual terminologies in cases when neither a morphological analyzer nor a POS tagger are available for the target language. This new result is based on a powerful probabilistic model that explicitly models distortion in alignments and ensures that the produced

alignment is optimal respecting the contiguity constraint according to the probabilistic model. The statistical model is advantageously exploited by the use of weighted multi-tape calculus. Test results confirm the claimed advantages.

Extensive previous literature on the problem of word alignment, from [17] then [7, 19] and [20, 21, 18] to [22] either does not cover the contiguity issue at all or can not enforce it as a constraint.

As far as bilingual terminology from parallel corpora is concerned, most proposed methods [1–6] rely on target-source matching sequences of Part-of-Speech requiring reliable POS taggers for both. We need only one, on the source side. [23] is asymmetric but can not guarantee optimality with respect to the underlying model [7].

Acknowledgments

This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

1. van der Eijk, P.: Automating the acquisition of bilingual terminology. In: Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (1993) 113–119
2. Dagan, I., Church, K.: Termight: identifying and translating technical terminology. In: Proceedings of the fourth conference on Applied natural language processing, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1994) 34–40
3. Gaussier, É., Hull, D., Ait-Mokhtar, S.: Term alignment in use: Machine-aided human translation. In Veronis, J., ed.: Parallel text processing: Alignment and use of translation corpora. Kluwer Academic Publishers (2000) 253–274
4. Déjean, H., Gaussier, E., Sadat, F.: An approach based on multilingual thesauri and model combination for bilingual lexicon extraction. In: Proceedings of the 19th international conference on Computational linguistics, Taipei, Taiwan (2002)
5. Hull, D.: (2005) US patent US6885985.
6. Hull, D.: Automating the construction of bilingual terminology lexicons. Terminology (1997)
7. Vogel, S., Ney, H., Tillmann, C.: HMM-based word alignment in statistical translation. In: COLING 96, Copenhagen (1996)
8. Kempe, A., Baeijs, C., Gaál, T., Guingne, F., Nicart, F.: WFSC – a new weighted finite state compiler. In Dang, Z., Ibarra, O.H., eds.: Proc. CIAA 2003. Volume 2759 of LNCS., Heidelberg, Springer-Verlag (2003) 108–119
9. Kempe, A., Champarnaud, J.M., Guigne, F., Nicart, F.: Wfsm auto-intersection and join algorithms. In: FSMNLP 2005, Helsinki, Finland (2005)
10. Nicart, F., Champarnaud, J.M., Csáki, T., Gaál, T., Kempe, A.: Multitape automata with symbol classes. In Ibarra, O.H., Yen, H.C., eds.: Proc. CIAA 2006. Volume 4094 of LNCS., Berlin, Heidelberg, Springer-Verlag (2006) 126–136

[7] The method in [23] is protected by IBM patent US6236958.

11. Gale, W.A., Church, K.W., Yarowsky, D.: Using bilingual materials to develop word sense disambiguation methods. In: Fourth International Conference on Theoretical and Methodological Issues in Machine Translation, Montreal (1992)
12. Simard, M., Plamondon, P.: Bilingual sentence alignment: Balancing robustness and accuracy. Machine Translation **13**(11) (March 1998)
13. Aswani, N., Gaizauskas, R.: A hybrid approach to align sentences and words in English-Hindi parallel corpora. In: ACL workshop on Building and Using Parallel Text. (2005)
14. Kumar Singh, A., Husain, S.: Comparison, selection and use of sentence alignment algorithms for new language pairs. In: ACL workshop on Building and Using Parallel Text. (2005)
15. Jacquemin, C., Bourigault, D.: Term extraction and automatic indexing. In Mitkov, R., ed.: Handbook of Computational Linguistics. Oxford University Press (2000)
16. Och, F.J., Ney, H.: A systematic comparison of various statistical alignment models. Computational Linguistics **29**(1) (2003) 19–51
17. Brown, P.F., Pietra, V.J.D., Pietra, S.A.D., Mercer, R.L.: The mathematics of statistical machine translation: parameter estimation. Comput. Linguist. **19**(2) (1993) 263–311
18. Kumar, S., Byrne, W.: A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In: HLT-NAACL 2003, Edmonton, Canada (2003)
19. Moore, R.: A discriminative framework for bilingual word alignment. In: Proc. of HLT/EMNLP 2005, Vancouver, BC, Canada (2005)
20. Och, F., Tillmann, C., Ney, H.: Improved alignment models for statistical machine translation. In: Proc. of EMNLP/VLC 1999, University of Maryland, College Park, MD, USA (1999)
21. Koehn, P., Och, F., Marcu, D.: Statistical phrase-based translation. In: Proc. of HLT-NAACL 2003, Edmonton, Canada (2003)
22. Goutte, C., Yamada, K., Gaussier, E.: Aligning words using matrix factorisation. In: Proc. of ACL'04., Barcelona, Spain (2004)
23. Gaussier, E.: Flow network models for word alignment and terminology extraction from bilingual corpora. In Boitet, C., Whitelock, P., eds.: Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics, San Francisco, California, Morgan Kaufmann Publishers (1998) 444–450

Finite-State Compilation of Feature Structures for Two-Level Morphology

François Barthélemy

CNAM (Cédric), Paris, France
INRIA (Atoll), Rocquencourt, France

Abstract. This paper describes a two-level formalism where feature structures are used in contextual rules. Whereas usual two-level grammars describe rational sets over symbol pairs, this new formalism uses tree structured regular expressions. They allow an explicit and precise definition of the scope of feature structures. A given surface form may be described using several feature structures. Feature unification is expressed in contextual rules using variables, like in a unification grammar. Grammars are compiled in finite state multi-tape transducers.

1 Introduction

Feature Structures are a convenient way of representing partial information. They have been broadly used for many purposes in Natural Language Processing.

Finite-State Morphology is an approach of computational morphology where the morphology of a natural language is described using contextual rules which denote a rational relation. These rules are simultaneous or sequential constraints. Each rule is compiled into a rational relation and all the relations are intersected or composed to obtain a unique relation implementing the grammar.

The use of feature structures for morphological computational descriptions is now very widespread. Many systems including Pc-Kimmo 2, Mmorph, Xerox Finite State Tool, have feature structures. Feature structure processing is usually performed by a separate engine, but there were some attempts to incorporate features into finite state machines.

Rémi Zajac [1] proposed a two-level formalism where the lexical level consists in a feature structure. This formalism is compiled into an extended finite-state transducer. Instead of concatenating the features of the lexical level, the extended finite-state machine unifies them. The surface representations are concatenated as usual. There is no restriction on the power of feature structure used, as far as the unification is decidable. The main drawback of the approach is that there is no other operation on feature structure than unification. The values may become more and more precise, but they can't change in a success path of the transducer. It is therefore impossible to encode informations such as the grammatical category that is changed by affix composition (e.g. `realization` where the form is a noun but the category of the root `realize` is verb).

More recently, Amtrup [2] proposed to use weighted finite state-machines, the weight being a feature structure. The idea is that feature structures with

union and unification are a semiring, which is the only property required on weights. Here again, there is no restriction on the feature structures, but the feature computation has to be monotonous. An important benefit with respect to Zajac is that there is no modification of the underlying finite-state machines.

Another approach is used by the Xerox tools [3]. There are no feature structure but *flag diacritics* which may be seen as independent features. There is a set of different operations on features: unification, but also positive setting, which gives a value to a flag, regardless of its previous value, negative setting which excludes a value for a given flag, positive and negative value test. All the operations act on a single feature, not a feature structure.

The flag operations are present in the finite state machines as special symbols concatenated to ordinary symbols. Operations are not performed by finite-state machines: they are performed at runtime, with an enumeration of the solutions and using a memory to store a single feature structure. Karttunen and Beesley propose to use features to decrease the size of finite-state machines. Finite-state machines over-generate, the over-generation being fixed at runtime. The system provides a command which transforms automatically a machine having flag diacritics into an equivalent machine without such flags, so the user may choose between run-time evaluation of these flags or compile-time evaluation, which may cause an explosion of the size of the machine.

Kiraz [4] proposed to compile features into finite-state automata. They are represented by strings of special symbols which are concatenated to the strings of grapheme/phoneme of the lexical level. In this framework, features are used only for two-level rule filtering: two-level rule application involves the unification of a feature structure associated to the rule with the feature structure associated to the lexical part of the feasible pair of the rule. Feature structures are always local to one morpheme. Features can't govern morpheme composition and no structure for the complete form is computed.

In this paper, we generalize the work of Kiraz in such a way that feature structures may be used not only for rule filtering, but also for affix concatenation. Instead of concatenating lexical representations and feature structure on the same level, they will be separated in two different levels. The rule application will involve possibly several feature structures. Two-level grammars become a kind of unification grammars. Such a formalism will allow the use of a feature structure as abstract representation of a form, following the proposition of Zajac.

Like Kiraz and the Xerox flag diacritics, some restrictions will apply on feature structures to allow their compilation as strings. Furthermore, some structural restrictions will apply on grammars in order to remain finite-state.

2 Compiling feature structures

In this section, we consider the compilation of a restricted kind of feature structures in strings. There is nothing new in this part of the paper: the techniques come from previous work about compilation of feature structures either in finite-state machines or in Prolog terms (see for instance [5]).

We first restrict ourselves to flat feature structures taking values in finite domains (small ones). Such structures are a convenient way of describing partial information, which is possibly incrementally enriched, by unification with other structures or by applying some rules.

A feature structure is a set of features, each feature being identified by its name and taking a value in a small domain. Each pair (name, value) may be implemented by a special symbol written <name=value>, and a feature structure by the concatenation of the symbols corresponding to its features. For instance, the features `pers` and `num` representing respectively the person and the number of an English verbal form, take respectively the values 1, 2, 3 and `sg` (singular), `pl` (plural). The alphabet used for implementing them are: <pers=1>, <pers=2>, <pers=3>, <num=sg> and <num=pl>. A structure `[num=sg,pers=1]` is represented by the string <num=sg><pers=1>.

To obtain the uniqueness of the representation, one has to use a fixed order between features such as for instance the lexicographic order between feature names. If one knows the set of features which may enrich a feature structure along the computations, a feature structure may be compiled into a regular expression implementing all these features. For instance, the structure `[pers=3]` is compiled into (<num=sg>|<num=pl>)<pers=3>.

Unifying two structures is equivalent to intersecting the strings representing them.

The compilation technique extends to embedded acyclic structures. The notion of feature name is just replaced by the notion of path. For instance:

$$\begin{bmatrix} \text{cat} & \text{name} \\ \text{agr} & \begin{bmatrix} \text{gender} & \text{masc} \\ \text{number} & \text{plural} \end{bmatrix} \end{bmatrix}$$

<agr.gender=masc><agr.number=plural><cat=name>.

Such an imbrication is convenient when several structures share the same substructure. This may be denoted using a single variable. In the compiled form, there will be no difference with respect to a flattened structure.

The disjunction and difference over regular expressions give support for feature structures with disjunctive and negative specification. For instance:

$$\begin{bmatrix} \text{person} & 1|2 \\ \text{tense} & \leftarrow \text{past} \end{bmatrix}$$

(<person=1>|<person=2>)(dom(tense)–<tense=past>)

3 Relating feature structures and strings

In the propositions of Zajac and Amtrup, a single feature structure is associated to respectively a surface form and a pair lexical and surface forms. For Kaplan

and Kay [6], there is a feature structure for each symbol, describing its phonological properties using binary features. Kiraz gives an example where a feature structure is associated to each lexical entry (typically, a morpheme), several such entries being concatenated to obtain a surface form. This means that morphological descriptions may use feature structures having different scopes with respect to the symbols of surface form. And why not using several types of structures with different scopes in the same description? For instance, one feature structure for each affix and another one for the complete form.

Partition-based morphology gives a way to implement this notion of scope. It is a variant of two-level morphology first defined by [7] and further improved by [8], [9] and [10]. Instead of describing a length preserving relation using symbol pairs, it uses pairs of strings of possibly different length. For instance an affix-based description of the form *impossibly* is (im:in)(possibil:possible)(ity:ity). In such a system, the pairing is not distributive with respect to concatenation, so the above string is considered different from (i:i)(mpossibil:npossible)(ity:ity), for instance. In other words, the splitting of strings in substrings is significant. In the implementations, the boundaries of substrings are represented using a special symbol. We will use the symbol `w`.

Such a segmentation of surface form is useful for feature structures. For instance, ([cat=name],spi)([number=plural],es). It is possible to use n-ary relations instead of binary relations, so the feature structures may be added to the two classical levels (lexical and surface) as a third level.

The compilation method proposed in [10] consists in compiling n-ary regular expressions in n-tape transducers synchronized on substrings terminators, inserted at the end of each pair – or tuple in the case of n-ary expressions. The other symbols are read independently, ordered according to the level they belong to. For instance, a string (aaa:xx)(b:yy) is compiled in the same-length expression a:0 a:0 a:0 0:x 0:x w:w b:0 0:y 0:y w:w, and then in the corresponding letter transducer.

The join operation is a way to merge relations (resp. transducers) which share some common components (resp. tapes). [10] shows that this operation is defined if the two operands have exactly one common level. This property holds even when this common level is split in two different ways in the two relations. A different substring terminator is used for each way.

We propose a multi-level formalism where regular expressions and contextual rules are extended to describe tree-structured relations. Each level in the tree is an n-tuple with n greater or equal to 1. The syntax `<i|` and `|i>` is used to respectively open and close a tuple at a depth `i` in the tree. `<0|` and `|0>` open and close the tuple at the root of the tree. Each member of the relation is composed of exactly one such tuple. `<0|` and `|0>` are used in the description as the string boundaries classically needed and sometimes written #.

Such structured representations are compiled using terminators, i.e. the tuple openings and commas separating their components disappears and the tuple closings are compiled into a special symbol ω_i read on all the relevant tapes.

[cat=name,num=pl]				
[type=root]				[type=suffix]
s	p	y		s
s	p	i	e	s

```
<0| [cat=name,num=pl],
    <1| [type=root],
        <2| s, s |2>
        <2| p, p |2>
        <2| y, i |2>
        <2| epsilon, e |2> |1>
    <1| [type=suffix],
        <2| s, s |2> |1> |0>
```

Fig. 1. An example of the tuple notation

In order to remain finite-state, tree-structures must be restricted. The syntax that we propose here refers explicitly to the depth of trees, so it describes depth-bounded trees, which are finite-state. A discussion of the tractable tree structures will take place in the last section of this paper.

We use a simplified version of the *generalized restriction rules* by Yli-Jyrä and Koskenniemi [11]. Let Π be a finite alphabet and $\diamond$ a symbol not in Π. A rule is written $W \Rightarrow W'$ where $W \subseteq \Pi^* \diamond \Pi^* \diamond \Pi^*$ and $W' \subseteq \Pi^* \diamond \Pi^* \diamond \Pi^*$. W is called the precondition, W' the postcondition. The diamonds are used to split strings in three parts: the left context, the center and the right context. Let $d_\diamond$ be the operator which deletes all the occurrences of the symbols $\diamond$ in a language. It may be formally defined as the composition with a finite transducer followed by a projection. The rule $W \Rightarrow W'$ denotes the language $\Pi^* - d_\diamond(W - W')$.

Informally speaking, if the precondition holds, then the postcondition has to be verified. The diamonds are markers inserted in regular expressions to define the center of the rule in such a way that precondition and postcondition apply on the same part of the strings. The context restriction and surface coercion rules from previous versions of two-level morphology may be written using this unique kind of rules.

In our system, the patterns W and W' of a rule $W \Rightarrow W'$ must be valid tree-structured regular expressions where the center is any part of the expression.

Feature structure types are declared as a set of names associated to finite domains, each value being a string. In the expressions, the features are explicitly typed. The type is given first, then the pairs (name, value). For instance, `[verb:pers=3,gen=m]` is a feature structure of type `verb`.

Variables may be used to represent a value shared by several features in an expression or in a rule. An expression with such a variable is equivalent to the disjunction of the expressions where the variable is replaced by a given value. Variables will be written with an identifier beginning with the symbol $.

4 Examples

In the first example, there is a unique feature structure associated to each form. This example consists in a partial description of the imperfective of the Arabic verb. The information about gender, number and person is given by prefix and suffix added to a core.

The description begins with some declarations. The type of feature structures is given as a list of feature names and for each name, the domain of values of the feature. In this first example, there are two levels of structure: there is a grouping of letters into affixes, and then a grouping of affixes into a form to which a feature structure is associated.

The morphotactics is defined using regular expressions. The construction `REGEXP` gives a name to the disjunction of regular expressions it contains. Each such expression is terminated by a semi-colon. The underscore stands for any adequately structured string (wildcard). It has different actual meanings depending on its context. The construction `LET` allows to define a regular expression by applying algebraic operations on previously defined expressions.

```
FEATURE TYPES
  verb: gen in {m,f}, pers in {1,2,3}, num in {sg,pl,du};
END TYPES
REGEXP prefix IS
  <0| [verb:pers=1,num=sg], <1| a |1> _ |0>;
  <0| [verb:pers=2], <1| t a |1> _ |0>;
  <0| [verb:pers=3,gen=m], <1| y a |1> _ |0>;
  ...
END
REGEXP core IS
  <0| [verb:_], <1|_|1> <1| k t u b |1> <1|_|1> |0>;
  ...
END
REGEXP suffix IS
  <0| [verb:pers=1|3], _ <1| epsilon |1> |0>;
  <0| [verb:pers=2,gen=f], _ <1| i i n a |1> |0>;
  ...
END
LET form=intersect(prefix,core,suffix);
```

The relation `form` obtained by intersection of the three descriptions of prefix, core and suffix, contains verbal forms such as for instance:

```
<0| [verb:pers=1,num=sg],
        <1| a |1><1| k t u b |1><1| epsilon |1> |0>
```

It is a structured representation of the form `aktub` (I write). The notation `epsilon` stands for the empty string. The description uses an empty suffix to describe cases where nothing is suffixed to the core.

In this example, the notion of circumfix is probably more relevant than prefix and suffix. In the proposed implementation, the coordination of prefix and suffix is obtained through feature structure unification. A more explicit alternative is to define directly the circumfixes using expressions such as:

```
<0| [verb:pers=2,num=pl,gen=f],
                <1| t a |1> _ <1| n a |1> |0>;
```

The above description gives the lexical form of a verb. To obtain the surface form, some phonological and graphemical rules are to be applied. It is possible to express them using a classical set of contextual rules which associates pairs of symbols from lexical and surface level, and then join this two-level system with the rational relation `form` defined here, identifying the lexical level of the two systems. The result of this join is a ternary relation.

The second example shows that it is possible to use feature structures not only for the morphotactics but in the contextual rules too. It consists in a partial description of French conjugation. There is a feature structure for each affix, having different types according to the affix. The sharp symbol in contextual rules is used to identify the center of the rule (instead of the $\diamond$ symbol used in the presentation of generalized restriction rules).

```
FEATURE TYPES
  root: conj in {1,2,3};
  suff1: tense in {pres, fut, past, cond, imp};
  suff2: conj, tense, pers in {1,2,3}, num in {sg, pl};
END TYPES
ABBREVIATION infix : for tuples depth 2;
REGEXP affix IS
  <1| [root:conj=1], a:_ i:_ m:_ |1>;
  <1| [root:conj=3], c:_ o:_ u:_ s:_ |1>;
  ...
  <1| [suff1:tense=pres], epsilon:_ |1>;
  <1| [suff1:tense=fut], R:_ |1>;
  ...
  <1| [suff2:tense=fut,pers=1,num=sg], a:_ i:_ |1>;
  <1| [suff2:tense=!passe,pers=1,num=pl], o:_ n:_ s:_|1>;
  ...
END
REGEXP morphotactics IS
  <0| <1| [root:conj=$C], _ |1><1| [suff1:tense=$T], _|1>
      <1| [suff2:conj=$C,tense=$T], _ |1> |0>;
END
LET affix_star=star(affix);
LET verbal_forms=intersect(morphotactics,affix_star);
RULES
  <0| <1| [root:conj=3], _ o:_ u:_ #s:_# |1>
    <1| [suff1:tense=fut], R:_ |1> _ |0> =>
```

```
  _ #s:d# _;
  <1| [root:conj=1], _ |1> <1| _, #epsilon:_ R:_# |1> _
            =>
  _ #epsilon:e R:r# _;
  ...
END
```

`$C` and `$T` are variables which represent any value for respectively the conjugation and the tense of a form. They are used to ensure that two feature structures have the same value, whatever it is.

Examples of forms are *aimerai* (I will love) and *coudrons* (we will sew):

```
 <0|   <1| [root:conj=1], a:a i:i m:m |1>
       <1| [suff1:tense=fut], epsilon:e R:r |1>
       <1| [suff2:tense=fut,pers=1,num=sg,conj=1],
                                      a:a i:i |1>
|0>;
<0|    <1| [root:conj=3], c:c o:o u:u s:d |1>
       <1| [suff1:tense=fut], R:r |1>
       <1| [suff2:tense=fut,pers=1,num=pl,conj=3],
                             o:o n:n s:s |1>  |0>;
```

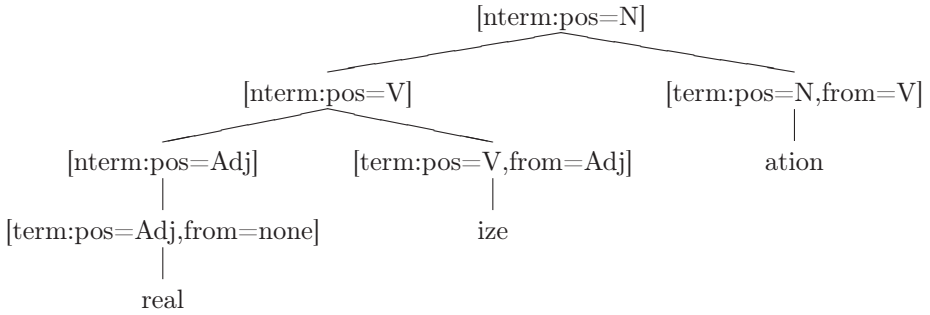
This example shows that no new construction is needed for using feature structures in contextual rules. The usual notions of context and center are sufficient. Features may be seen as a syntactic facility to express regular strings through a macro-expansion. The result of this macro-expansion is a set of regular generalized restriction rules which are compiled using the algorithm by Yli-Jyrä and Koskenniemi [11].

The third example illustrates how two different kinds of feature structures may be used in the same grammar: one kind will be devoted to the description of affixes; the other one to the composition of such affixes. They may be viewed as the terminal and non-terminal nodes of a unification grammar, respectively.

```
FEATURE TYPES
  term: pos in {N,Adj,V}, from in {N,Adj,V,none};
  nterm: pos;
END TYPES
CLASSES
  <letter>: a,b,c ...
END CLASSES
TUPLE TYPES
  <1| [nterm_], [term:_], <letter>* |1>;
END
REGEXP affix IS
  <1| _,[term:pos=Adj,from=none], real |1>;
  <1| _,[term:pos=V,from=none], move |1>;
  ...
```

```
  <1| _,[term:pos=V,from=Adj], ize |1>;
  <1| _,[term:pos=N,from=V], ation |1>;
  ...
END
LET affix_star=star(affix);
RULES suffixation ARE
<0| #<1| [nterm:pos=$P],_,_ |1># _ |0> =>
  _ #<1| [nterm:pos=$P],[term:pos=$P,from=none],_ |1># _;
_ #<1| [nterm:pos=$P1],_,_ |1># _ =>
  _ <1| [nterm:pos=$P2],_,_ |1> #<1| [nterm:pos=$P1],
                        [term:pos=$P1,from=$P2],_ |1># _;
END
LET stem=intersect(affix_star,suffixation);
```

This piece of code describes structures such as the following:



This syntax tree is encoded into a sequence of triples where each internal node is aligned with its rightmost child:

```
<0| <1| [nterm:pos=Adj],[term:pos=Adj,from=none],real |1>
    <1| [nterm:pos=V],[term:pos=V,from=Adj], ize |1>
    <1| [nterm:pos=N],[term:pos=N,from=V], ation |1>
|0>
```

The two generalized restriction rules are an encoding of the unification grammar:

$$\begin{bmatrix} \text{nterm} & \\ \text{pos} & \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{term} & \\ \text{pos} & \boxed{1} \\ \text{from} & \text{none} \end{bmatrix}$$

$$\begin{bmatrix} \text{nterm} & \\ \text{pos} & \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{nterm} & \\ \text{pos} & \boxed{2} \end{bmatrix} \begin{bmatrix} \text{term} & \\ \text{pos} & \boxed{1} \\ \text{from} & \boxed{2} \end{bmatrix}$$

Let us detail the compilation of the simplest of the two rules:

```
<0| #<1| [nterm:pos=$P],_,_ |1># _ |0> =>
  _ #<1| [nterm:pos=$P],[term:pos=$P,from=none],_ |1># _;
```

The first step consists in replacing the variable appearing in the center of the rule, namely `$P`, by its possible values, resulting in a set of three rules:

```
<0| #<1| [nterm:pos=N],_,_ |1># _ |0> =>
  _ #<1| [nterm:pos=N],[term:pos=N,from=none],_ |1># _;
<0| #<1| [nterm:pos=Adj],_,_ |1># _ |0> =>
  _ #<1| [nterm:pos=Adj],[term:pos=Adj,from=none],_ |1>#
  _;
<0| #<1| [nterm:pos=V],_,_ |1># _ |0> =>
  _ #<1| [nterm:pos=V],[term:pos=V,from=none],_ |1># _;
```

The second step replaces each feature structure by its compiled form as a symbol string. The wildcard symbol is replaced by the relevant expression where the symbol `<any>` stands for any regular symbol in the alphabet (all the symbols except the end of tuple `<wi>` and #). For the first of the three rules from the previous step, it gives:

```
<0| #<1| <nterm><pos=N>,<any>*, <any>* |1>#
    <1| <any>*,<any>*,<any>*  |1>* |0> =>
 <0| <1| <any>*,<any>*,<any>*  |1>*
     #<1| <nterm><pos=N>,<term><pos=N><from=none>,<any>*
      |1>#
     <1| <any>*,<any>*,<any>* |1>* |0>;
```

The third step consists in compiling the tuples using the techniques presented in the section 3. Like in classical Two-Level morphology [12], 0 is a special symbol inserted to obtain same-length relations. It is treated alternatively as an ordinary symbol (for intersection) or as the empty string (for composition).

```
#:#:# <nterm>:0:0 <pos=N>:0:0 (0:<any>:0)* (0:0:<any>)*
<w1>:<w1>:<w1> #:#:#
((<any>:0:0)* (0:<any>:0)* (0:0:<any>)* <w1>:<w1>:<w1>)*
          =>
#:#:# <nterm>:0:0 <pos=N>:0:0 0:<term>:0 0:<pos=N>:0
      0:<from=none>:0 (0:0:<any>)* <w1>:<w1>:<w1> #:#:#
((<any>:0:0)* (0:<any>:0)* (0:0:<any>)* <w1>:<w1>:<w1>)*
```

Finally, the rule is compiled using the formula from [11], namely $\Pi^* - d_\diamond(W - W')$ where W and W' are respectively the left-hand side and the right-hand side of the rule and Π^* the support of the relation. In our example,
Π^* is a sequence of triples `<1|_,_,_|1>*`. Thanks to the type declaration of the tuples, the compiler knows that it is more precisely:
`<1| [nterm_], [term:_], <letter>* |1>*`, which compiles into the following:

```
(<nterm>:0:0 (<pos=V>:0:0|<pos=N>:0:0|<pos=Adj:0:0>)
0:<term>:0 (0:<pos=V>:0|0:<pos=N>:0|0:<pos=Adj:0>)
(0:<from=none>:0|0:<from=V>:0|0:<from=N>:0|
                                    0:<from=Adj:0>)
(0:0:<letter>)* <w1>:<w1>:<w1>)*
```

The result of the compilation of the rule is given in the figure 2. The notation `<name=_>` is used as an abbreviation which stands for any symbol associating a value to the feature `name`.

5 Theoretical and practical limits

There are two kinds of limits to the compilation of feature structures using tree-structured relations: theoretical limits due to the kind of tree structures which can be represented in finite-state machines; practical limits due to the size of the finite-state machines.

Not all structure are implementable as finite-state machines. It is well-known, for instance, that context-free parsing is not finite-state. Chomsky in [13] gives a characterization of grammars which are regular. A grammar is said *self embedding* if there exists a derivation $A \stackrel{*}{\rightarrow} \alpha A \beta$ where A is a non-terminal and α and β are non-empty strings. A grammar is regular if and only if it is not self-embedding. This includes finite, right-linear and left-linear grammars.

Note that our examples use implicitly linear structures although it seemingly describes only finite structures because sequences of tuples of a given level are allowed within a tuple of higher level. For example in
`<0| [verb:_], <1|_|1> <1| k t u b |1> <1|_|1> |0>`, there is a sequence of three tuples of depth 1 as second component of the tuple of depth 0.

Linear structures are sufficient to express some morphologies, such as for, instance, Turkish morphology or French flexion which use mostly suffixes. They are not sufficient to represent English or French derivation which use both prefixes and suffixes. The solution in these cases are to restrict to use only finite grammars, for instance by limiting the depth of recursion for self-embedding non-terminals.

From a practical point of view, descriptions involving tree-structured relations may be too large to be compiled and executed. Feature structures may describe long-distance dependencies like in the example of circumfixation of Arabic verbal forms. We have implemented a prototype which converts the formalism presented in this paper into genuine finite-state automata and uses the FSM toolkit [14] to compile and execute them. We have written a number of sample grammars for French and Turkish verbs and a medium-size grammar of the Akkadian verb (about 50 rules). During these experiments, we sometimes encountered size explosion that we resolved by a careful writing of grammars and ordering of algebraic operations

Feature structures must be limited to a small number of features having small domains. Like feature diacritics in Xerox Tools, feature structures in our system could be evaluated at run-time, when a composition with a surface or abstract form drastically decreases the size of the machine. Instead of performing unification at compile-time, equations giving values to features should be concatenated within each scope, i.e. in each tuple.

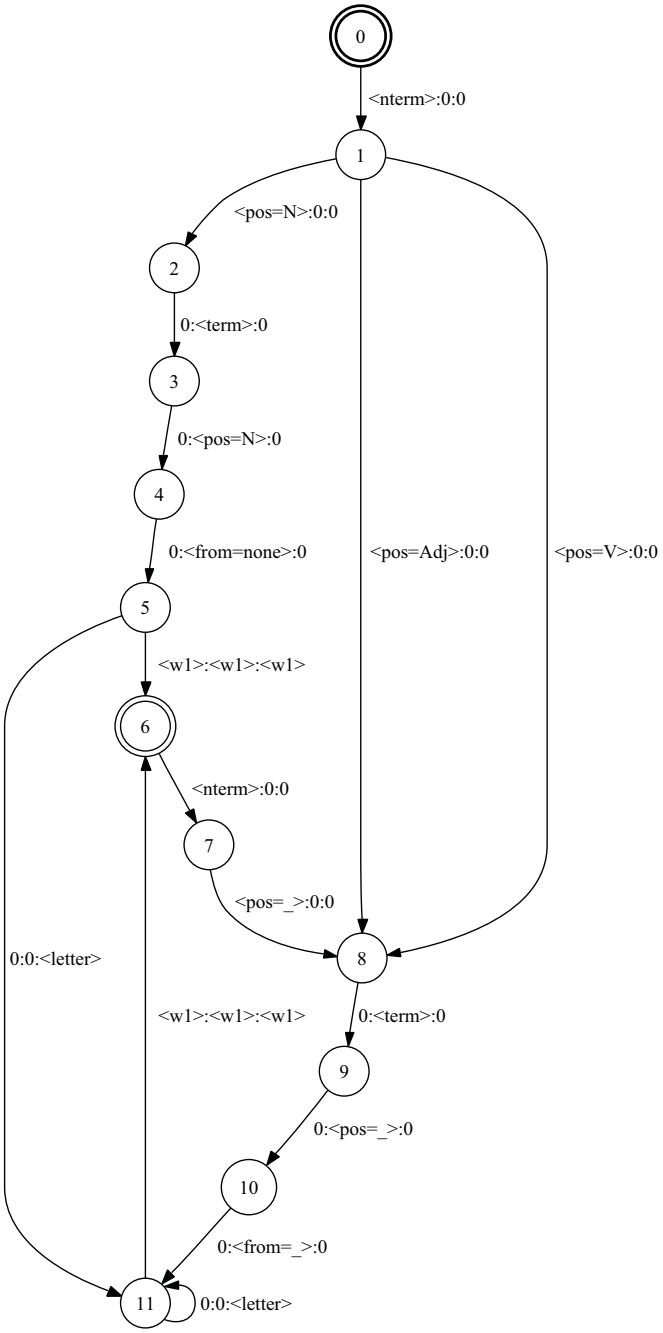


Fig. 2. Result of the rule compilation

6 Conclusion

The technique proposed in this paper is an improvement of the proposition by Kiraz, namely compiling feature structures into regular expressions which are part of a n-ary relation. The benefits of our approach are a more flexible use of the features and the possible simultaneous use of several kinds of scope for feature structures. There is also a better integration to contextual rules: the feature structures are part of the contexts and centers, and not a side condition.

With respect to the propositions by Amtrup and Zajac, the compilation in regular expressions offers a better integration into the two-level formalism. On the other hand, there are restrictions on the kind of structures and a risk of explosion of the size of the machines.

Ideally, a smart compiler should analyze grammars using unrestricted feature structures and automatically separate them in three parts: a small number of features or features approximations which are statically compiled, a second set of features which are computed at runtime as a finite-state operation (e.g. transducer composition), after the composition with a surface (or abstract) form and finally the features which are not computable using finite-state machines, and which would be evaluated separately for each solution by an external device. There is still a lot of work to perform such a statical analysis of grammars and to improve compilation techniques for the first two subsets of features.

References

1. Zajac, R.: Feature structures, unification and finite-state transducers. In: FSMNLP'98: International Workshop, on Finite State Methods in Natural Language Processing. (1998)
2. Amtrup, J.W.: Morphology in machine translation systems: Efficient integration of finite state transducers and feature structure descriptions. Machine Translation **18**(3) (2003) 217–238
3. Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI Publications (2003)
4. Kiraz, G.A.: Compiling regular formalisms with rule features into finite-state automata. In: ACL, Madrid, Spain (1997)
5. Shöter, A.: Compiling feature structures into terms: an empirical study in prolog. Technical Report EUCCS-RP-1993-1, ICCS, Edinburgh, Scotland (1993)
6. Kaplan, R.M., Kay, M.: Regular models of phonological rule systems. Computational Linguistics **20:3** (1994) 331–378
7. Black, A., Ritchie, G., Pulman, S., Russell, G.: Formalisms for morphographemic description. In: Proceedings of the third conference on European chapter of the Association for Computational Linguistics (EACL). (1987) 11–18
8. Pulman, S.G., Hepple, M.R.: A feature-based formalism for two-level phonology. Computer Speech and Language **7** (1993) 333–358
9. Grimley-Evans, E., Kiraz, G., Pulman, S.: Compiling a partition-based two-level formalism. In: COLING, Copenhagen, Denmark (1996) 454–459
10. Barthélemy, F.: Using Mazurkiewicz trace languages for partition-based morphology. In: ACL, Prague (Czech Republic) (2007)

11. Yli-Jyrä, A., Koskenniemi, K.: Compiling contextual restrictions on strings into finite-state automata. In Watson, B., Cleophas, L., eds.: Proc. Eindhoven FASTAR Days, Eindhoven, Netherlands (2004)
12. Koskenniemi, K.: Two-level morphology: a general computational model for word-form recognition and production. Technical Report 11, Department of General Linguistics, University of Helsinki (1983)
13. Chomsky, N.: On certain formal properties of grammars. Information and Control **2**(2) (1959) 137–167
14. Mohri, M., Pereira, F.C.N., Riley, M.: Weighted finite-state transducers in speech recognition. Computer Speech and Language **16**(1) (2002) 69–88

Segmentation in Super-Chunks with a Finite-State Approach

Olivier Blanc[1], Matthieu Constant[2], and Patrick Watrin[2]

[1] University of Munich, CIS, Germany
[2] University of Marne-la-Vallée, IGM, France

Abstract. Since Harris' parser in the late 50s, multiword units have been progressively integrated in parsers. Nevertheless, in the most part, they are still restricted to compound words, that are more stable and less numerous. Actually, language is full of semi-fixed expressions that also form basic semantic units: semi-fixed adverbial expressions (*e.g.* time), collocations. Like compounds, the identification of these structures limits the combinatorial complexity induced by lexical ambiguity. In this paper, we detail an experiment that largely integrates these notions in a finite-state procedure of segmentation into super-chunks, preliminary to a parser. We show that the chunker, developped for French, reaches 92.9% precision and 98.7% recall. Moreover, multiword units realize 36.6% of the attachments within nominal and prepositional phrases.

1 Introduction

Since Harris' parser in the late 50s [1], multiword units have been slowly integrated in parsers [2]. Nevertheless, in the most part, they are still restricted to compound words, that are more stable and less numerous. Actually, language is full of semi-fixed expressions that also form basic semantic units: semi-fixed adverbial expressions (*e.g.* time), nominal collocations. Like compounds, the identification of these structures limits the combinatorial complexity induced by lexical ambiguity.

To study this phenomenon, we implemented an incremental finite-state *chunker* for French[3] based on the notion of *super-chunk*. Super-chunks are different from the notion traditionally associated with chunks [4–7], because adjectival and prepositional attachment has been integrated. From a formal point of view, non-recursivity is verified. Like chunks, a super-chunk stops at its head (*e.g.* the noun in a nominal chunk). Nevertheless, by taking account of multiword units (MWUs), the notion of head is extended to complex structures. For instance, *marge d'exploitation* (trading margin) and *chiffre d'affaires brut* (gross sales turnover) are tagged as a noun at the lexical analysis stage[4] and therefore

[3] Tools presented in this paper are in the most part based on the programs of the Outilex platform [3].

[4] Note that morphosyntactic information is inherited from the lexical head of the MWU (*i.e. marge* and *chiffre*). In addition, that information is augmented with the

are computed as simple words. In that case, the ambiguity reduction is obvious. By analysing the sequence *chiffres d'affaires brut* (gross sales turnover) in a compositional manner, the procedure leads to 24 analyses, that are reduced to one if the collocation is considered as a whole. Moreover, this sole lexical entry permits the resolution of a double attachment (a prepositional one and an adjectival one), which facilitates the identification of the syntactic constituents.

The chunker presented in this paper is part of a larger project of developing a complete parser for French, directly usable by real applications such as information extraction. Our system is composed of three successive stages : (1) lexical segmentation into simple and MWUs ; (2) identification and tagging of super-chunks ; (3) attachment in constituents. An illustration of this incremental procedure is given in the table 1. In this paper, we will only focus on the two first stages that are both based on finite-state resources.

We will first describe the lexical segmentation module with the description of the lexical resources used; we will show how part of them has been automatically learnt and how they have been applied to texts. Then, we will present the super-chunk segmentation module inspired by [4] and next, the disambiguation process. Finally, an evaluation of the performances of our chunker will be made and its interest for resolving lexical attachments will be shown.

2 Lexical segmentation

The lexical segmentation is a key part of our chunker. It takes as an input a text segmented in sentences and in tokens. It is entirely based on Lexical Resources (LRs) either developed by linguists or automatically learnt from raw texts. These resources are either in the form of morpho-syntactic dictionaries or in the form of lexicalized local grammars.

2.1 Manually constructed lexical resources

The lexical module includes a large-coverage morpho-syntactic dictionary of inflected French forms. This dictionary has been developed between the mid-80's and the mid-90's by linguists at the University of Paris 7 [8, 9]. It is composed of 746,198 inflected simple forms and 249,929 inflected compounds (including 245,436 compound nouns). This dictionary is a set of lexical entries, each of them being composed of an inflected form, a lemma, a part-of-speech, morphological information (*e.g.* gender, number), syntactic information (*e.g.* internal structure of MWUs) and semantic information (*e.g.* human feature for nouns). It is of a great interest because of its fine-grained linguistic precision and the large amount of MWUs. These MWUs are compound words of the following types:

– nouns: *pomme de terre* (potato), *faux témoignage* (perjury)

syntactic internal structure of the MWUs (*i.e. noun-preposition-noun* and *noun-preposition-noun-adjective*).

Level	Example
Text	Le groupe de télécommunications néerlandais KPN a annoncé avoir acquis une participation de 77,5% dans le troisième opérateur allemand de téléphonie mobile E-Plus.
Lexical	Le [$_N$ groupe de télécommunications] néerlandais KPN a annoncé avoir acquis une participation de 77,5% dans le troisième [$_N$ opérateur allemand de téléphonie mobile] E-Plus.
Chunk	Le [$_N$ groupe de télécommunications] [$_{XA}$ néerlandais] KPN a annoncé [$_{XVI}$ avoir acquis] une participation de 77,5% dans le [$_{XA}$ troisième] [$_N$ opérateur allemand de téléphonie mobile] E-Plus.
	[$_{XN}$ Le groupe de télécommunications] [$_{XA}$ néerlandais] [$_{XN}$ KPN] a annoncé [$_{XVI}$ avoir acquis] [$_{XN}$ une participation] de [$_{XN}$ 77,5%] dans [$_{XN}$ le troisième opérateur allemand de téléphonie mobile E-Plus].
	[$_{XN}$ Le groupe de télécommunications] [$_{XA}$ néerlandais] [$_{XN}$ KPN] [$_{XV}$ a annoncé avoir acquis] [$_{XN}$ une participation] [$_{XP}$ de 77,5%] [$_{XP}$ dans le troisième opérateur allemand de téléphonie mobile E-Plus].
Sentence	[$_{N0}$ Le groupe de télécommunications néerlandais KPN] [$_V$ a annoncé avoir acquis] [$_{N1}$ une participation de 77,5% dans le troisième opérateur allemand de téléphonie mobile E-Plus].

Table 1. Global process

– prepositions: *au milieu de* (in the middle of), *à cause de* (because of)
– adverbs: *par ailleurs* (moreover), *en pratique* (in practice)
– conjunctions: *bien que* (although), *pendant que* (while)

This large-coverage dictionary is compressed in the form of an FST in order to be efficiently applied to the text.

Our lexical resources also contain a library of lexicalized local grammars. Local grammars [10] are Recursive Transition Networks (RTNs) [11] and theoretically recognizes algebraic languages. They are of great interest for representing local lexical and syntactic constraints in a simple and compact way. We use them mostly to describe MWUs. They can define syntactic classes such as noun determiners and even syntactico-semantic classes such as time adverbials. Linguistic descriptions are in the form of Finite-State Graphs on an alphabet made of terminal and non-terminal symbols. A terminal symbol is a lexical mask, *i.e.* an underspecified lexical entry (some features are missing) equivalent to a feature structure representing a set of lexical entries: *e.g.* the lexical mask <*noun+p*> matches all nouns in the plural. Finally, a non-terminal symbol is a reference to another graph. A graph is a transducer and its output is the annotation assigned to structures described in the graph. An example of a local grammar is given in figure 1[5]. This grammar describes time adverbials and recognizes structures like

[5] The local grammars are drawn using the graph editor of the Unitex platform [12].

en mars 2007 (in March 2007) and *cinq minutes plus tard* (five minutes later). The sequences recognized by this graph are tagged as time adverbs (`ADV+time`[6]). Strings between < and > are lexical masks: for instance, <*minute*> stands for the inflected forms whose lemma is *minute*. Greyed vertices are call to other graphs. For example, `Dnum` and `month` are graphs that respectively recognizes numerical determiners and the names of months.

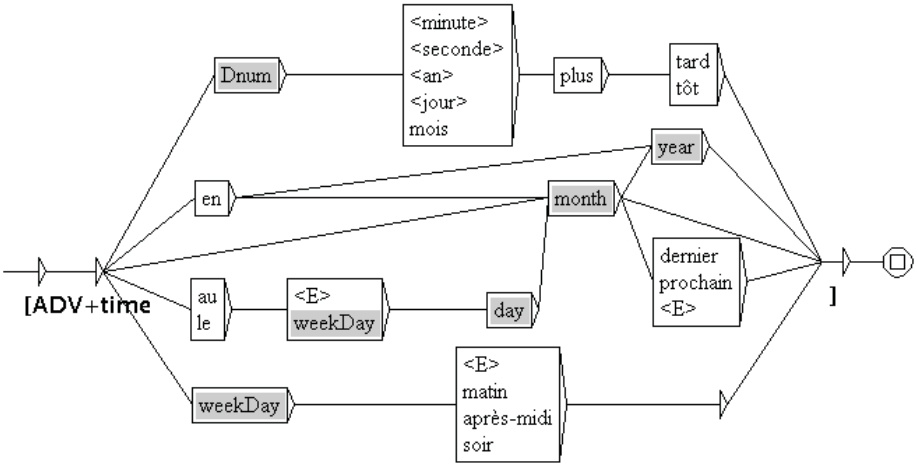


Fig. 1. Local grammar of time adverbials

Practically, the lexical module includes a network of 190 graphs. Local grammars recognize sequences of the following types:

- nouns: function names [*ministre anglais de l'Agriculture* (English minister of Agriculture)]
- prepositions: locative prepositions [*à dix kilomètres au nord de* (ten kilometers north of)]
- determiners: numerical determiners [*vingt-sept* (twenty seven), *des milliers de* (thousands of)], noun determiners [*dix grammes de* (ten grams of)]
- adverbs: time adverbials [*en octobre 2006* (in october 2006)]

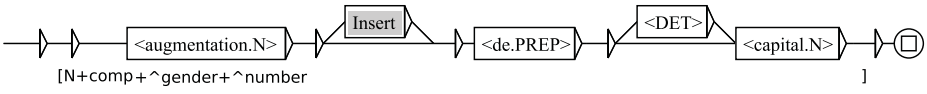


Fig. 2. Collocation: *rise of capital*

[6] The output is in bold and is displayed under the vertices of the graph.

2.2 Nominal collocations

The lexical analyzer also uses a set of nominal collocations. Collocations are sequences of words that co-occur more often than usually expected. Their automatic extraction from raw corpora has been the focus of many papers during the last decade. Nominal collocations can contain a preposition and therefore could be useful for preposition attachment in parsing. For instance, the collocation *offre d'emploi* (job offer) forms a basic semantic unit with the internal structure *noun-preposition-noun*. Considering it as a lexical unit would then resolve the prepositional attachment. To extract collocations, we used the approach defined in [13] and [14]. It consists in applying a set of nominal syntactic patterns on a tagged text and then in evaluating statistically identified candidates. Our learning text comprises 1 million words of French broadcast news and is tagged with TreeTagger [15]. Morpho-syntactic ambiguity, the principal source of noise in extraction, is therefore removed. Each word of the text is associated with a unique lemma and a unique part-of-speech. Next, local grammars representing basic nominal structures and their variants are applied to the tagged text. Extracted candidates that have a frequency greater than 5 are then statistically evaluated by computing the *log-likelihood* measure defined by [13] for bigrams and by [16] for trigrams. Finally, the best nominal collocations are kept and each one is assigned an internal syntactic structure. Given this structure, a local grammar is automatically constructed for each collocation. Each local grammar represents potential variations of the corresponding collocation (*e.g.* taking the insertion of a modifier into account). For instance, the local grammar associated with the collocation *augmentation de capital* (rise of capital, *cf.* Figure 2) recognizes the sequence *augmentations exceptionnelles de capital* (exceptional rises of capital). We therefore extracted 1,330 basic canonical bigrams and 163 basic canonical trigrams. Note that the number of extracted collocations could seem rather low. Nevertheless, as we want to obtain a very low error rate in order to have a totally automated process, we put very strong statistic constraints on the extraction computation. Note that, among the extracted collocations, 69.1% of bigrams and 86.5% of trigrams contain a prepositional attachment, which shows the great interest of locating the extracted nominal collocations during the lexical segmentation process. More details on this extraction process can be found in [17].

2.3 LR application

The lexical segmentation module is divided in two stages: (1) dictionary lookup then (2) application of lexicalized local grammars. The dictionary lookup stage enables to associate each token with all its possible linguistic tags and to recognize MWUs. The output of the process is a Text Finite State Automaton (TFSA). Then, local grammars are directly applied to the TFSA, which is then augmented with the analyses of the matching MWUs.

This process also allows for reducing artificial ambiguities by removing very infrequent analyses from the dictionary[7]. For instance, the analysis of *a* as a noun (letter *a*) is removed. In order to avoid the silence caused by the removal of analyses from the dictionary, it is possible to get these analyses given a specific context. To do so, we use local grammars. For instance, the form *par* is only tagged as a preposition, except if it belongs to local golf-specific lexicalized contexts such as *16 au-dessous du par* (16-under) in which context it is also tagged as a noun. Some MWUs also require a specific context to be analyzed as such. For instance, the sequence *en train de* can be interpreted as a preposition if it is followed by an infinitive verb:

Jean est [en train de PREP] dormir (John is sleeping)

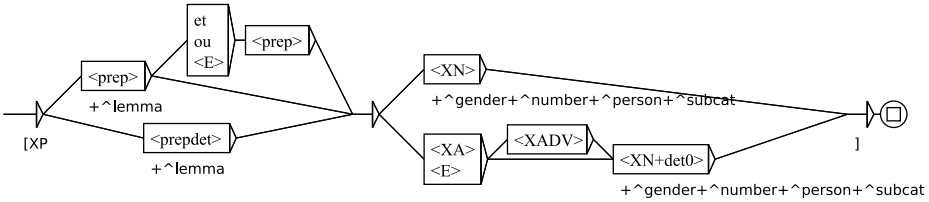


Fig. 3. Prepositional chunk

3 Chunk segmentation

Chunking is also an incremental process: it is in the form of a cascade of FSTs applied on the TFSA, which is then augmented each time a new chunk is found. It is composed of height stages and uses a network of 18 graphs. It consists in identifying:

1. adverbials (`XADV`): simple adverbs or multiword adverbials that have been recognized during the lexical segmentation
2. adjectival chunks (`XA`): adjectives that can be preceded by an adverb
3. nominal chunks (`XN`): simple noun phrases, named entities, some types of pronouns
4. prepositional chunks (`XP`): `XN` preceded by a preposition
5. verbal chunks (cascade of 4 FSTs): passive and active forms of infinitive, past participle, gerund and simple verbal chunks (`XVI`, `XVI-passive`, `XVK`, `XVK-passive`, `XVG`, `XVG-passive`, `XV`, `XV-passive`)

In general, the identified chunks inherit morpho-syntactic properties from their head as it is shown in figure 3 that represents an `XP`. `XP` inherits the gender and the number of its head (`^gender` and `^number`).

[7] Actually, we use a system of priorities.

Once the cascade of FSTs was applied on the TFSA, the latter is cleaned. The cleaning process consists in removing the transitions whose labels do not belong to the chunking level (*e.g.* nouns, verbs, adjectives, ...). It keeps only paths of the TFSA that go from the initial state (beginning of sentence) to the final state (end of sentence).

The chunking process applied to the sequence *au sujet d'un attentat terroriste* produces the TFSA given in the figure 4.

4 Incremental disambiguation

The chunk segmentation produces a set of possible analyses in chunks in the form of a TFSA for each sentence of the input text. In order to reduce or remove ambiguity, the chunker includes an incremental disambiguation module composed of three optional stages.

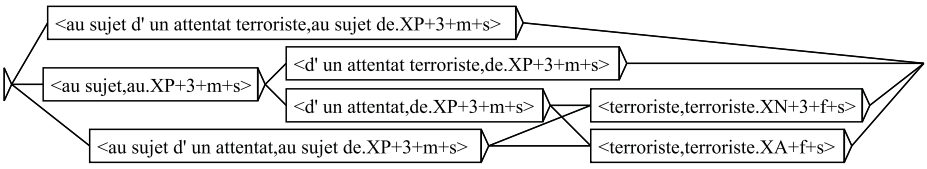


Fig. 4. TFSA after chunking

4.1 Applying the Shortest Path Heuristic (SPH)

The SPH consists in keeping only the shortest paths of the TFSA. This language-independent heuristic could seem simple and naïve at first sight. But, in practice, it is very efficient because it is based on the idea of preferring multiword expression analyses to sequences of simple analyses. The SPH algorithm is an adaptation of Dijkstra's algorithm to keep all shortest paths of a graph instead of one only.

The application of the heuristic to the TFSA in figure 4 produces a TFSA reduced to one path: *<au sujet d'un attentat terroriste.XP>*.

4.2 Applying hand-crafted rules

Given an instance of ambiguity and specific left and right contexts, the chunker user might want to prefer an analysis to the others. We therefore developed a simple formalism of disambiguation rules. A rule consists of three parts: two contextual parts (left and right) that are represented by local grammars (these two parts can be empty: `EMPTY`); a central ambiguous part that is a list of possible analyses. If the ambiguity is found in the TFSA with the defined left and right

contexts, then the first analysis in the list of ambiguous items is selected. The other analyses are then removed from the TFSA.

For instance,

```
XN_elag.wrtn
  <XP> <XN>
    EMPTY
```

When applied to the TFSA in figure 5, the rule above would keep only the `XP` analysis for the sequence *de lutte contre le terrorisme* (of war against terrorism) in the right context of an `XN` (recognized by the `XN_elag.wrtn` grammar).

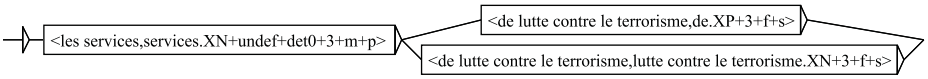


Fig. 5. Chunk ambiguity

4.3 Applying stochastic rules

There are some types of ambiguity that cannot be resolved by manually constructed general rules. A typical example is the `XV-XN` ambiguity (*e.g.* the word *massacre* can be an `XV` or `XN`). We therefore decided to use probabilistic rules automatically learnt from an automatically tagged corpus[8]. Given a word form that can be tagged either as a noun or a verb, the most frequent analysis is prefered. For instance, the form *massacre* would be tagged noun because the analysis noun has a probability of 0.7 while the analysis verb has a probability of 0.3 in our corpus. If no occurrences of a given form have been found in our corpus then the most frequent chunk category is selected (for instance, `XN` would be prefered to `XV`).

Note that all stages are optional because linearization is optional. There are some cases where it is better to keep ambiguities when resolving them is too risky: for instance, if chunking is applied just before an attachment resolution module. The `XN-XV` ambiguity is a typical example of ambiguity that is better resolved at the attachment level.

5 Evaluation and Discussion

Our evaluation process has been carried out on a corpus composed of broadcast news from `http://www.yahoo.fr` web site. This 13,492-word corpus includes 6,901 super-chunks.

[8] The corpus is one year of the newspaper `Le Monde` and has been tagged with TreeTagger.

Our definition of a chunk is different from the standard definition because it integrates the notion of MWU. As a consequence, there exists no reference annoted corpora that is compatible with our definition. The evaluation was therefore carried out manually. The 3-stage process described above was applied to the corpus: (1) lexical segmentation using dictionaries and local grammars presented in section 3; (2) segmentation in super-chunks by applying successively 18 finite-state rules; (3) desambiguation by applying SPH, then 26 hand-built rules and then a statistical module. Precision and recall were evaluated manually by two persons. Table 2 shows the results obtained.

Precision	92.9%
Recall	98.7%

Table 2. Results

From a general point of view, we observed that most of errors are due to incomplete lexical and syntactic resources. That means that improvements can easily be made and will soon be. In practice, we distinguish recall errors and precision ones.

Recall errors are only caused by incomplete LRs: dictionaries and local grammars. First, some compound grammatical words are missing from the dictionary, *e.g. tandis que* (while), *au-dessous de* (below). Moreover, elements are missing from the local grammars. For instance, in the named entity grammar, the sequence *Nouri al Maliki* is not recognized as a unit because the form *al* is unknown and has not been integrated in the grammar as family name prefix. There are some semi-fixed expressions that have not been integrated such as *vers 8h45* (at around 8.45 am). Some complex pronominal structures like *au cours de laquelle* are also missing in the local grammars.

Precision errors can be divided into four classes:

1. SPH-related errors

Lexical ambiguity can lead to wrong chunks after computing the SPH heuristic because it tends to keep the longest chunks. For instance, in the sentence *après l'affirmation du quotidien espagnol El Pais* (after the writing of the Spanish newspaper El Pais), there are two possible analyses:

[après l'affirmation XP] [du quotidien espagnol XP] [El Pais XN]

[après l'affirmation XP] [du quotidien XP] [espagnol XA] [El Pais XN]

As *quotidien* and *espagnol* are both adjectives or nouns, SPH prefers the `[Prep XA N]` analysis to the `[Prep N] [XA]` one.

2. Wrong decision based on probabilistic rules

For example, in the sentence *La côte Est et les villes de New York ...*, there are two possible analyses of the chunk *Est*: either an `XV` (to be) or an `XA` (East). Although *Est* is `XA` in this context, the statistical module prefers the analysis `XV` because *est* is more often tagged as a verb in our training corpus.

3. Errors caused by the application of disambiguation rules

That kind of errors is fortunately infrequent. They mainly concern the `XP`–`XN` ambiguity due to the lexical ambiguity of *de* which can be either a determiner or a preposition. For instance, a disambiguation rule that is applied at a late stage of the incremental disambiguation process, prefers `XP` to `XN`. This analysis is used to make an arbritrary decision. In the example *qui n'a pas fourni de plus amples détails* (who didn't provide more details), the chunk *de plus amples détails* is an *XN*.

4. Dictionary coverage

Some missing compound structures in the dictionary cause errors. For instance, *en outre* is a compound adverb but is missing from the dictionary. Therefore, the compositional analysis is chosen in the sentence *ils ont en outre pris plusieurs centaines de personnes en otage* (they took several hundreds of hostages). It is then chunked as follows

[ils XN] [ont XV] [en outre pris plusieurs centaines XP] [de personnes XP] [en otage XP]
instead of
[ils XN] [ont en outre pris XV] [plusieurs centaines XN] [de personnes XP] [en otage XP]

where *en outre* is an adverb inserted in a verbal chunk.

In addition to recall and precision evaluation, we also estimated the impact of MWUs for lexical attachment. We observed the actual realization of 36.6% of the lexical attachment, with no error, within noun phrases and prepositional ones.

We also applied our chunker on the same corpus without integrating MWU resources. Our chunker then becomes a standard chunker. The corpus passes from 6,901 super-chunks to 7,503 chunks (around 8% augmentation). We then observe a slight rise of precision by using MWUs: the number of errors falls from 600 to 485. Recall is also slightly better: 116 analyses are missing without using MWUs vs. 89 with MWUs. All these figures show the great interest of using super-chunks instead of only standard chunks. First, as the number of chunks decreases, the combinatorial ambiguity is reduced and further processes should be eased. Moreover, as MWUs form "islands of non-ambiguity", they are useful to reduce internal ambiguity within chunks and therefore to reduce precision errors.

6 Conclusion and perspectives

In this paper, we presented a chunking technique based on a significant augmentation of the lexical level by taking into account larger sequences (*i.e.* MWUs). By using this approach, we search for optimizing the disambiguation process on

the one hand and computing a part of the lexical attachment within noun and prepositional phrases on the other hand.

To evaluate the relevancy and the efficiency of our assumption, we carried out an experiment on a broadcast news corpus from the web. Results led us to a double conclusion:

- Our disambiguation procedure reaches excellent recall and precision rates without the use of any tagger;
- a significant amount of attachments within noun and prepositional phrases are actually resolved by the use of a large-coverage set of MWUs, and therefore do not have to be computed at the syntactic level.

Future work will focus on the improvement of the super-chunker by improving the lexical and syntactic resources and by integrating a more sophisticated statistical disambiguation module (*e.g.* use of Hidden Markov Models). We wish to extend it in order to process less stable textual data such as spoken texts or emails. Moreover, we would like to compare the super-chunker with standard chunkers and to evaluate its impact when it is integrated in a parser.

References

1. Joshi, A., Hopely, P.: A parser from antiquity. Natural Language Engineering **2**(4) (1997)
2. Nivre, J., Nilsson, J.: Multiword units in syntactic parsing. In Dias, G., Lopes, J.G.P., Vintar, S., eds.: Proceedings of the Workshop on Methodologies and Evaluation of Multiword Units in Real-World Applications, Language and Resource Evaluation Conference. (2004) 39–46
3. Blanc, O., Constant, M.: Outilex, a linguistic platform for text processing. In: Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions. (2006) 73–76
4. Abney, S.P.: Partial parsing via finite-state cascades. Natural Language Engineering **2**(4) (1996) 337–344
5. Karlsson, F., Voutilainen, A., Heikkila, J., Anttila, A.: Constraint Grammar: A language-independent system for parsing unrestricted text. Volume 4 of Natural Language Processing. Mouton de Gruyter (1995)
6. Federici, S., Montemagni, S., Pirelli, V.: Shallow parsing and text chunking: A view on underspecification in syntax. In: Proceedings of the ESSLLI'96 Workshop on Robust Parsing. (1996)
7. Ait-Mokhtar, S., Chanod, J.P.: Incremental finite-state parsing. In: Proceedings of the fifth Conference on Applied Natural Language Processing ANLP'97. (1997)
8. Courtois, B.: Un système de dictionnaires électroniques pour les mots simples du français. Langue Française **87** (1990) 11–22
9. Courtois, B., Garrigues, M., Gross, G., Gross, M., Jung, R., Mathieu-Colas, M., Monceaux, A., Poncet-Montange, A., Silberztein, M., Vivés, R.: Dictionnaire électronique DELAC : les mots composés binaires. Technical Report 56, LADL, University Paris 7 (1997)
10. Gross, M.: The construction of local grammars. In Roche, E., Schabes, Y., eds.: Finite-State Language Processing. The MIT Press, Cambridge, Mass. (1997) 329–352

11. Woods, W.: Transition network grammars for natural language analysis. Communications of the ACM **13**(10) (1970)
12. Paumier, S.: De la reconnaissance de formes linguistiques à l'analyse syntaxique. PhD thesis, Université de Marne-la-Vallée (2003)
13. Dunning, T.: Accurate methods for the statistics of surprise and coincidence. Computational Linguistics **19**(1) (1993) 61–74
14. Daille, B.: Combined approach for terminology extraction: lexical statistics and linguistic filtering. Technical report, Lancaster University (1995)
15. Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: International Conference on New Methods in Language Processing, Manchester, UK (1994)
16. Seretan, V., Nerima, L., Wehrli, E.: Extraction of multi-word collocations using syntactic bigram composition. In: Proceedings of th 4^{th} International Conference on Recent Advances in NLP (RANLP-2003). (2003) 424–431
17. Watrin, P.: Une approche hybride de l'extraction d'information : sous-langages et lexique-grammaire. PhD thesis, Université catholique de Louvain (2006)

Intersection Optimization is NP-Complete

Guillaume Bonfante[1] and Joseph Le Roux[2]

[1] INRIA – LORIA
[2] Nancy Universités – LORIA

Abstract. Finite state methods for natural language processing often require the construction and the intersection of several automata. In this paper, we investigate the question of determining the best order in which these intersections should be performed. We take as an example lexical disambiguation in polarity grammars. We show that there is no efficient way to minimize the state complexity of these intersections.

1 Introduction

The main concern of this paper is to answer the following question: given a set $\{A_1, \ldots, A_k\}$ of finite state automata, can we guess an order on them to efficiently perform their intersection? More precisely, can we find a permutation π for which the following algorithm will run as quickly as possible?

```
A = A[pi[1]];
for i = 2 to k do
   A = A intersect A[pi[i]]
done
```

Observe that computing the intersection as above takes in the worst case exponential time. Indeed, the size of the result, that is to say the number of states, is exponential $|\bigcap_{i\leq k} A_i| = \prod_{i\leq k} |A_i|$. We refer to Saaloma and Yu to learn more about state complexity [1, 2]. But this is not the issue addressed here. The question is to find the order in which we have to perform the intersections. And we show that this part of the problem is also inherently difficult. To get rid of the size problem, we consider the ordering problem with regards to some a priori upper bound on the size of automata. The decision problem will be proved NP-complete.

A standard NP-complete problem about automata intersections is the emptiness of the result. See for instance [3] or [4] which give explicit upper bounds. But, here, we are more concerned with the intersection process than the result itself. An analogous question to our present issue is matrix multiplication. Given a sequence of matrices $M_1, \ldots, M_k$ of different sizes, the way one parenthesizes the expression $M_1 \times \cdots \times M_k$ has a huge impact on the cost of evaluating the product (see [5]). For this problem, computing the best order can be done in polynomial time by a dynamic programming procedure.

Let us now present the practical application which originally motivated the present study: disambiguation for lexicalized polarized grammars (PGs) like Categorial Grammars [6], Interaction Grammars [7] or Polarized Unification Grammars [8]. A lexicalized grammar is defined by its lexicon, which associates a set of *syntactic items* to every word of the language. Each of these items specifies a grammatical construction in which the corresponding word participates.

One of the main features of PGs is that each syntactic item is equipped with polarized features. Polarities are used to guide the process of syntactic composition: features with the same type but with opposite polarities try to neutralize each other. The process ends successfully in a parse structure for a sentence where all polarized features are neutralized.

Syntactic items, if we forget their structure, become bags of polarized features. A necessary condition for a tagging to be successful is that summing polarized features in the bag must end with a zero. Automata are a well-suited way of factorizing this counting. The crux is that one may count different features, each of which provides an automaton. Hence, the resulting necessary condition is given by the intersection of these automata [9]. Unfortunately, it is known [10] that when performing multiple intersections, intermediate automata can possibly be huge, even if the final automaton is small.

We prove that looking for the order in which intersections have to be performed to create the minimal number of intermediate states is actually NP-hard. For that reason, we have used heuristics[3] in our implementation.

2 Polarized Grammars and Lexical Disambiguation

In this section, we present a general lexical disambiguation method for PGs relying on automata intersection.

2.1 Polarized Grammars and Parsing

We give here a very brief description of such grammars. Any reader who wants a wider presentation of these grammars should refer to [6–8]. A polarized grammar is equipped with:

- a set W of words (for instance English vocabulary);
- a set S of items (for example "noun phrase coordination");
- a function $\ell : W \to \mathcal{P}_{\text{fin}}(S)$ which associates words with finite sets of items;
- a set of feature names $\mathcal{F}$ (e.g. "category" and "gender") and a set of feature values $\mathcal{V}$ (e.g. "noun" and "masculine");
- a function $\rho : S \times \mathcal{F} \times \mathcal{V} \to I[\mathbb{Z}]$ which associates to any item and feature name/value a finite interval over the integers. This function counts the polarized features of items. For instance, $\rho(\text{give\_Verb}, \text{"cat"}, \text{"noun phrase"}) = [-3, -1]$ because a verb like *give* can be intransitive (expecting 1 noun phrase), transitive (2 noun phrases) or ditransitive (3 noun phrases).

[3] We cannot present these heuristics here for lack of space.

Given a sentence $w_1, \ldots, w_n$ of words in W, the parsing process consists of a) selecting one item for each word of the sentence, say $s_1, \ldots, s_n$ and b) checking that this selection verifies some properties depending on the grammatical framework. Still, there is one common property to all PGs which is that 0 must be an element of the sum of the intervals in the selection, where intervals are summed according to $[a, b] + [c, d] = [a + c, b + d]$. This property can be stated: $\forall f, v \in \mathcal{F} \times \mathcal{V} : 0 \in \sum_{i \leq n} \rho(s_i, f, v)$. We call this property the *global neutrality criterion* and it reflects the neutrality constraint on final structures.

2.2 Counting with Automata

We assume a sentence $w_1 w_2 \ldots w_n$ to parse with a PG G. Given a feature name and a feature value (f, v), consider the automaton $A(f, v)$ as follows:

- A state of the automaton is a pair (i, p), where i corresponds to the position of the word in the sentence and p is an interval of $\mathbb{Z}$, which represents the counting of polarities up to position i.
- Transitions have the form $(i, p) \xrightarrow{s_\alpha} (i + 1, q)$, where $s_\alpha \in \ell(w_i)$, $q = p + \rho(s_\alpha, f, v)$.
- The initial state is $(0, [0, 0])$.
- The accepting states are states (n, p) such that 0 is an element of p.

Every path in $A(f, v)$ from the initial state $(0, [0, 0])$ to an accepting state represents a lexical selection that verifies the global neutrality criterion. Other paths can be deleted. So, any path to an accepting state is a candidate for selection.

Actually, it is a necessary condition for a correct lexical selection to be recognized by polarity automata, for every *choice* of name f and value v. As a consequence, the intersection of polarity automata gives an automaton which also contains the valid solutions. The principle of our selection method is to build the automaton[4] $\bigcap_{(f,v) \in \mathcal{F} \times \mathcal{V}} A(f, v)$.

For example, in our implementation for Interaction Grammars, for a ten word long sentence we usually make twelve intersections. With this method we go from 5000 raw selections to 10 selections respecting the neutrality criterion. We have noticed some performance issues depending on the order in which we performed the intersection. On some sentences, we experienced tenfold variations in the number of states of the intermediate automata..

3 NP-Completeness of the Problem

In this section, we review three problems, which we prove to be NP-complete, related to our disambiguation technique based on automata intersections.

In these three problems we ask whether it is possible to determine the right order in which the intersection of several automata must be performed to minimize the number of intermediate states.

[4] Actually we can restrict our attention to some more particular values for f and v. See [9] for details.

We prove NP-hardness of these problems by reduction from the Traveling Salesman Problem (TSP) [3]. To fix the notations, we first recall this illustrious problem.

An instance of the TSP is a triple (V, d, K) where $V = \{1, \ldots, n\}$ is a set of cities, d is a distance function between any pair of different cities, $d(i, j) \in \mathbb{N}$, and a bound $K \in \mathbb{N}^+$. The problem is to decide whether there exists a tour of all cities with a length less than K or in other words if there exists a permutation π of the cities such that $(\sum_{i=1}^{i=n-1} d(\pi(i), \pi(i+1))) + d(\pi(n), \pi(1)) \leq K$. For clarity, when π is a function $[1..n] \to [1..n]$ and the context is clear, we write $\pi(n+1)$ for $\pi(1)$ and $\pi(0)$ for $\pi(n)$. So the previous sum can be written $\sum_{i=1}^{i=n} d(\pi(i), \pi(i+1)) \leq K$. From now on, we restrict our attention to those cases where $d(i, j) \leq 2$. The problem remains NP-complete (it corresponds to the reduction from Hamiltonian Circuit).

We will distinguish between the traditional TSP as it has been described above and a variant that we call the exact TSP in which the tour must be of length exactly K (see [3]).

3.1 Intersection Optimization Problems

We present a first intersection optimisation problem, that we will enrich to get the second and third problems, which are more difficult. In the proofs, we do not use automata with loops. So these problems can be stated with or without star languages. For an automaton A, *the size of* A that we write $|A|$ is the number of states of A. Every automaton is considered minimal, unless stated otherwise.

First Problem (IO1): Let $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ be a set of n finite state automata, $B \in \mathbb{N}^+$ a bound and K a target size. Is there an injective function $\pi : [1..j] \to [1..n]$ such that

- $|(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| = K$
- for all $k < j$, $|(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(k)}| \leq B$.

In other words, is there a subset $\mathcal{A} \subseteq \mathcal{A}_n$ such that $|\bigcap_{A \in \mathcal{A}} A| = K$ with all intermediate steps smaller than B? For disambiguation, this means that we are able to know the size of the final intersection.

Second Problem (IO2): Let $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ be a set of n finite state automata and $B \in \mathbb{N}$ a bound. Is there a bijection $\pi : [1..n] \to [1..n]$ such that for any $j \leq n$ we have $|(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$? For disambiguation, this means that given a set of polarity automata we are able to know how to perform their intersection in order to bound the size of each intermediate intersection.

Third Problem (IO3): Let $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ be a set of n finite state automata and $B \in \mathbb{N}$ a bound. Is there a permutation π of $[1..n]$ such that $\sum_{1 \leq j \leq n} |(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$? This is the problem that we deal with in disambiguation: is there an order to perform intersection for which the total number of states that we create is bounded?

3.2 NP Algorithms

These three problems are in NP. Each time we have to choose a permutation π and then:

- for (IO1), if an intermediate intersection is empty we stop and the answer to the problem is "no" (of course, if $K = 0$ it is "yes") if it has a size greater than B, the answer is no. Otherwise, we proceed to the next intersection. When j intersections have been performed we compare the size of the resulting automaton to K. Observe that those intersections can be performed in time bounded by B^2 since all intermediate steps must have a size lesser than B. And so, we are polynomial with regards to B.
- for (IO2), if an intermediate intersection is empty then the answer is "yes" else if it is greater than B (again, we may need to consider B^2 states before minimization) the answer to the problem is "no" else we proceed to the next intersection.
- for (IO3), we need to sum the sizes of the intermediate intersections and check that this sum is never greater than B. If an intersection is empty or if a partial sum exceeds B then we can stop immediately.

3.3 NP-Completeness

Theorem 1. ***(IO1)*** *is NP-complete.*

Proof. We consider some cells, that we will associate to build automata. They are given by Figure 1.

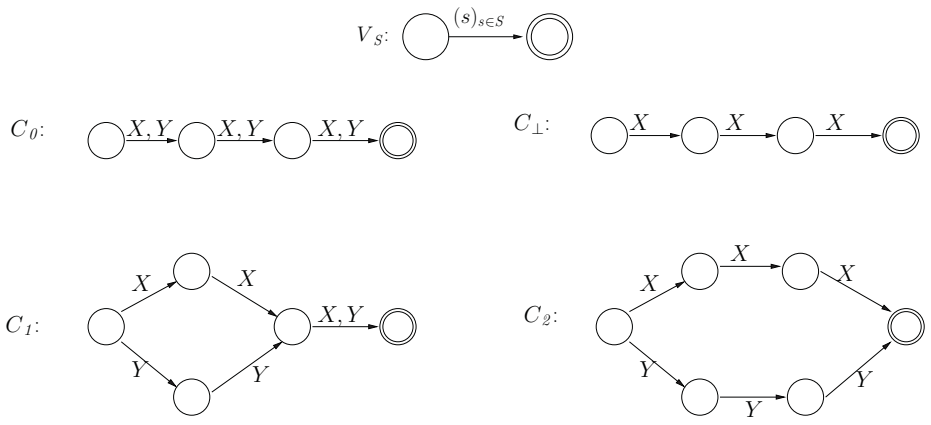


Fig. 1. Some brick automata

We can note that for $i \in \{0, 1, 2\}$ we have $|C_i| = |C_0|+i$. In other words, these automata encode the distance between two cities. Observe also that $C_i \cap C_0 =$

C_i. So that C_0 is the "neutral" element for the intersection. Finally, if A is some automaton, A' denotes the same automaton, but with primed letters. We suppose we are given an arbitrary (but minimal) automaton D of size $6 \times n + 3$.

Now, given an instance of the exact TSP (V, d, k), we consider a set of automata $\mathcal{A}_{i,j,m}$ with $i, j \in V$ and $m \leq n$ where n is the number of cities in V. To fix the intuition, the automaton $\mathcal{A}_{i,j,m}$ corresponds to the choice of going from city i to city j at step m of a tour. In other words, it corresponds to the choice $\pi(m) = i$ and $\pi(m+1) = j$. The m^{th} distance is set to $l = d(i,j)$ by cell C_l, between letters V_i and V_j. Moreover, if i is the initial city, it is also the last one. We define:

$$\begin{aligned}
\mathcal{A}_{i,j,1} &= V_i C_{d(i,j)} V_j (C_0 {V_{V\setminus\{i,j\}}})^{n-2} C_0 V_i + {V'}_{V\setminus\{1\}} D \\
\mathcal{A}_{i,j,m} &= ({V_{V\setminus\{i,j\}}} C_0)^{m-1} V_i C_{d(i,j)} V_j (C_0 {V_{V\setminus\{i,j\}}})^{n-m} + {V'}_{V\setminus\{m\}} D \\
&\quad \text{for } n > m > 1 \\
\mathcal{A}_{i,j,n} &= V_j C_0 ({V_{V\setminus\{i,j\}}} C_0)^{n-2} V_i C_{d(i,j)} V_j + {V'}_{V\setminus\{n\}} D
\end{aligned}$$

Let us consider the "witness" automaton $\mathcal{A} = ({V_V} C_0)^n {V_V}$ where no distance is set. Remark that $|\mathcal{A}_{i,j,m}| = |\mathcal{A}| + d(i,j) + |D|$. The (polynomial) reduction is then $(V, d, K) \mapsto ((\mathcal{A}_{i,j,m})_{i,j,m}, 2|D|, |\mathcal{A}| + K)$.

Correctness If there is a tour defined by π of length exactly K, observe that:

$$\bigcap_{1 \leq m \leq n} \mathcal{A}_{\pi(m),\pi(m+1),m} = V_{\pi(1)} C_{d(\pi(1),\pi(2))} V_{\pi(2)} C_{d(\pi(2),\pi(3))} \cdots C_{d(\pi(n),\pi(1))} V_{\pi(1)}$$

which has a size $|\bigcap_{1 \leq m \leq n} \mathcal{A}_{\pi(m),\pi(m+1),m}| = |\mathcal{A}| + \sum_{1 \leq m \leq n} d(\pi(m), \pi(m+1))$. The bound on intermediate automata is discussed widely in the next proof. So, if the TSP has a solution, then its encoding has a solution for (IO1).

Completeness We consider the set $\mathfrak{A}$ of automata $(\mathcal{A}_{i,j,m})_{i,j,m}$ closed by intersection for the converse part. Any non empty automata $A \in \mathfrak{A}$ has the following properties (proved by successive inductions):

(i) $A = A_1 + A_2$ with
  - $A_1 = \emptyset$ or
  - $A_1 = V_{\alpha_1} C_{\beta_1} V_{\alpha_2} C_{\beta_2} \cdots V_{\alpha_n} C_{\beta_n} V_{\alpha_{n+1}}$, $\alpha_i \subseteq V$, $\beta_i \in \{0,1,2\}$,

  and $A_2 = {V_S}' D$ with $S \subseteq \{1..n\}$;

(ii) In (i), if $\alpha_j = \{k\}$ for some j, then no other α_ℓ contains k for $\ell \leq n$,

(iii) In (i), $\beta_i = 0$ iff $i \in S$,

(iv) In (i), if $\beta_i \neq 0$, then $\alpha_i = \{k\}, \alpha_{i+1} = \{\ell\}$ are singleton sets and $\beta_i = d(k, \ell)$.

(v) In (i), $\alpha_1 = \alpha_{n+1}$

From (i), we can say that $|A| \leq |A_1| + |A_2|$. So, in the worst case, $|A| \leq 2 + \sum_{i=1}^{n} |C_{\beta_i}| + |D| < 2 \times |D|$, and the bound on intermediate steps is always respected. From (iii), we learn that ${V_S}' D$ is empty iff $\forall j : \beta_j \neq 0$. So that (iv)

with (ii,v) gives us the fact (F) that for all i, the set $\alpha_i = \{k_i\}$ is a singleton set and $\pi : [1..n] \to [1..n]$ which sends $i \mapsto k_i$ is a bijection and $k_1 = k_{n+1}$. Since, $|D| > |\mathcal{A}| + k$, $|A| = |\mathcal{A}| + k$ iff S is empty. The fact (F) above shows that it corresponds to an acceptable tour.

Theorem 2. ***(IO2)*** *is NP-complete.*

Proof. We reduce the TSP to IO2. Let (V, d, k) be an instance of the TSP, let 2 be the maximal distance between two cities and $n = |V|$. Again, for each pair of cities (i, j) with distance $d(i, j)$ we build n automata according to the possible positions of these cities in a tour. That is to say we build n^3 automata $\mathcal{A}_{i,j,p}$. Technically speaking, with regards to (IO1), we must have a stronger control on the order in which the intersection is performed. This is due to the fact that we have a weaker condition that applies to every intermediate automaton. That is also why the proof is much more complex.

We can decompose automata in three components:

1. The first one detailed in Fig. 2 (that we call $\mathcal{C}_{1,i,j,p}$) is responsible for computing the total distance of the tour, like in (IO1) but without indexing V by a set of cities. The end states of the C_0 sub-components are connected to the initial state of $\mathcal{C}_{2,i,j,p}$ by a dummy letter X. Hence, if all the distances are instantiated (as in IO1) then only the last V will connect this first component to the second component.

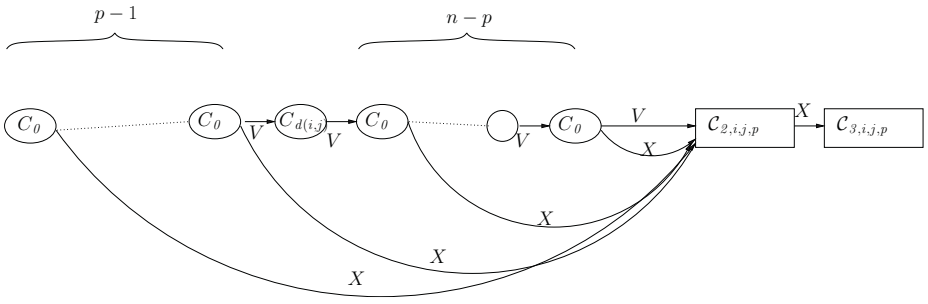


Fig. 2. Automaton $\mathcal{A}_{i,j,p}$ with detailed first component

2. The second one ($\mathcal{C}_{2,i,j,p}$) is responsible for chaining the edges correctly to make a valid tour. This component is shown on Figure 3. It should be observed that if it is intersected with $\mathcal{C}_{2,j,k,p+1}$ then the resulting automaton is of the same size. Otherwise (if city indices do not match) then it grows by $2n$ states.
3. The third one ($\mathcal{C}_{3,i,j,p}$), presented in Fig. 4, *forbids* (by making any unwanted intersection too big) the use of a position more than once and the use of position p without first considering positions $1, \ldots, p-1$. Otherwise it grows by $4n$ states.

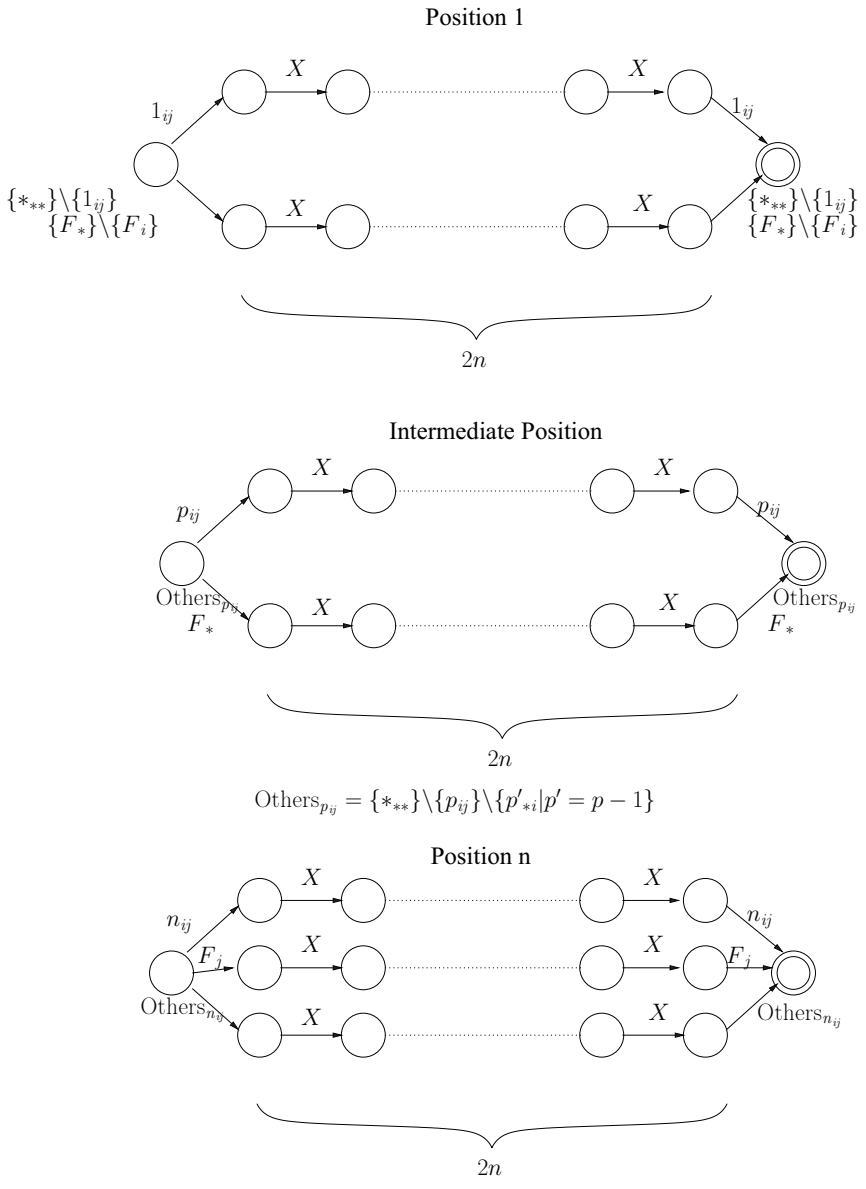


Fig. 3. The second component for the automaton $\mathcal{A}_{i,j,p}$

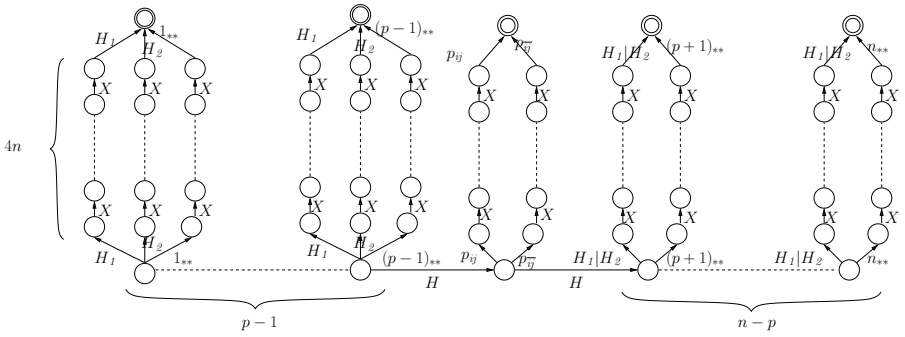


Fig. 4. The third component for the automaton $\mathcal{A}_{i,j,p}$

Finally, we need an additional automaton T, shown on Fig. 5. Its role is to end the intersection process.

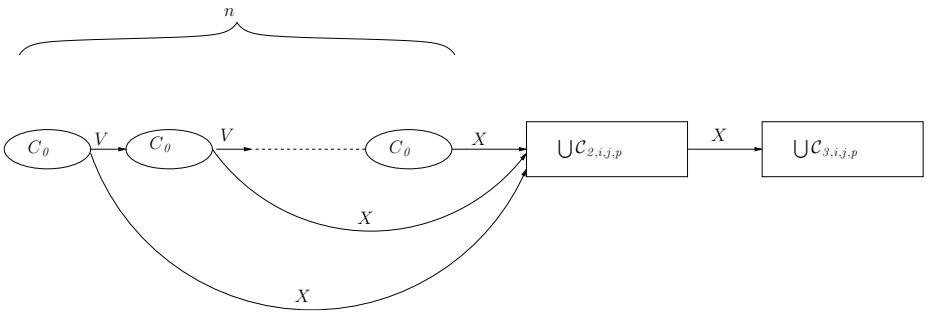


Fig. 5. The automaton T

The size of $\mathcal{A}_{i,j,p}$ is $|\mathcal{A}_{i,j,p}| = |\mathcal{C}_{1,i,j,p}| + |\mathcal{C}_{2,i,j,p}| + |\mathcal{C}_{3,i,j,p}|$ where

$$|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$$
$$|\mathcal{C}_{2,i,j,p}| = \begin{cases} 6n+2 \text{ if } p = n \\ 4n+2 \text{ otherwise} \end{cases}$$
$$|\mathcal{C}_{3,i,j,p}| = 3(p-1)(4n) + 2(n-p+1)(4n) + 2n = 2n(4n+2p-1)$$

$$|\mathcal{A}_{i,j,p}| = \begin{cases} 2 + d(i,j) + 4n(2+p+2n) & \text{if } 1 \leq p < n \\ 2 + d(i,j) + 2n + 4n(2+3n) \text{ otherwise} \end{cases}$$

We want to prove that $i_1, i_2, \ldots, i_n, i_1$ is a tour for the TSP with length lesser than k if and only if every intermediate automaton of the intersection

$$\bigcap_{1 \leq i \leq m, \alpha(i) \in I \subset [1..n]^3} \mathcal{A}_{\alpha(i)} \cap T \bigcap_{m+1 \leq j \leq n^3, \beta(j) \in [1..n]^3 \setminus I} \mathcal{A}_{\beta(j)}$$

is an automaton whose size is lesser than $B = 2 + K + 4n(1 + 2n)$. So the reduction is $(V = \{1 \dots n\}, d, K) \mapsto ((\mathcal{A}_{i,j,p})_{1 \leq i,j,p \leq n} \cup T, B)$

Preliminary observations. Without loss of generality, we can suppose that $K \leq 2n$. Otherwise the TSP is trivial. This entails that among all the automata $(\mathcal{A}_{i,j,p})_{i,j,p}$ only the automata $(\mathcal{A}_{i,j,1})_{i,j}$ are smaller than B:

i) $|\mathcal{A}_{i,j,1}| = 2 + d(i,j) + 4n(1 + 2n) < B$
ii) otherwise,

$$\begin{aligned} |\mathcal{A}_{i,j,p}| &\geq 2 + d(i,j) + 4n(1 + p + 2n) \\ &\geq 2 + d(i,j) + 4n(2 + 2n) + 4n(p-1) \\ &> 2 + d(i,j) + 4n(1 + 2n) + K > B \end{aligned}$$

Then, notice that:

iii) $|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$
iv)

$$|\mathcal{C}_{1,i,j,p} \cap \mathcal{C}_{1,k,l,q}| = \begin{cases} 2n + d(i,j) + d(k,l) & \text{if } p \neq q \\ 2n + \max(d(i,j), d(k,l)) & \text{otherwise} \end{cases}$$

v)

$$|\mathcal{C}_{2,i,j,p} \cap \mathcal{C}_{2,k,l,q}| = \begin{cases} |\mathcal{C}_{2,i,j,p}| & \text{if } q = p+1 \text{ and } j = k \\ |\mathcal{C}_{2,i,j,p}| + 2n & \text{otherwise} \end{cases}$$

vi)

$$|\mathcal{C}_{3,i,j,p} \cap \mathcal{C}_{3,k,l,q}| = \begin{cases} |\mathcal{C}_{3,i,j,p}| & \text{if } q = p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n|p-q| & \text{if } q > p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n & \text{if } q \leq p \end{cases}$$

(v) and (vi) mean that there is a way to preserve the size of the second and third components: it is to perform the intersection with respect to the order of a tour (v) and by considering each position once in ascending order (vi). Following these remarks, for any sequence prefix of a tour $i_1, i_2, \dots, i_k$ with $k \leq n+1$ (if $k = n+1$ we force $i_k = i_1$) in our instance of the TSP, we have

$$\begin{aligned} |\bigcap_{1 \leq p \leq k} \mathcal{A}_{i_p,i_{p+1},p}| &= |\bigcap_{1 \leq p \leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\bigcap_{1 \leq p \leq k} \mathcal{C}_{2,i_p,i_{p+1},p}| \\ &\quad + |\bigcap_{1 \leq p \leq k} \mathcal{C}_{3,i_p,i_{p+1},p}| \\ &= |\bigcap_{1 \leq p \leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| \\ &= 2n + (\textstyle\sum_{1 \leq p \leq k} d(i_p, i_{p+1})) + 4n + 2 + 2n(4n-1) \\ &= 2 + (\textstyle\sum_{1 \leq p \leq k} d(i_p, i_{p+1})) + 4n(1 + 2n) \end{aligned}$$

Correctness of the reduction. We first show that all intermediate intersections of $\mathcal{A}_{i_1,i_2,1} \cap \ldots \cap \mathcal{A}_{i_j,i_{j+1},j}$ for j ranging from 1 to n have a size lesser than B if there exists a tour $i_1, i_2, \dots, i_n, i_1$ with length lesser than k. We do this by induction on the steps of the intersection process.

As stated earlier, the initial automaton must be $\mathcal{A}_{i_1,i_2,1}$ because every other $\mathcal{A}_{i_1,i_2,p}$ would be too large. Then, by application of the equality defined above:

$$\begin{aligned}|A = \bigcap_{1\leq p\leq n} \mathcal{A}_{i_p,j_{p+1},p}| &= 2 + (\sum_{1\leq p\leq n} d(i_p, i_{p+1})) + 4n(1+2n)\\ &\leq 2 + K + 4n(1+2n) = B\end{aligned}$$

So these first n intersections straightforwardly encode the tour in the TSP instance. Now, observe that $A \cap T = \emptyset$ because every C_i from its first component is different from C_0. Consequently if the instance of the TSP has a solution, the sequence $\mathcal{A}_{i_1,i_2,1}, \ldots, \mathcal{A}_{i_n,i_1,n}, T, S$ where S is a sequence over $\{(\mathcal{A}_{i,j,p})_{i,j,p}\}\backslash\{\mathcal{A}_{i_k,i_{k+1},k} : k \leq n\}$ is a solution to IO2.

Completeness. Consider an intersection of the form

$$A = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i}) \cap T \cap (\bigcap_{(\alpha_j)_j \in [1..n]^3 \backslash I} \mathcal{A}_{\alpha_j})$$

where no intermediate automaton has a size greater than B. In particular, this is true for $A' = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i})$. We note $m = |I|$ and we can deduce that:

- α_1 is of the form $(x_1, y_1, 1)$; if $\alpha_i = (x_i, y_i, p)$ then $\alpha_{i+1} = (x_{i+1}, y_{i+1}, p{+}1)$ and $m \leq n$, otherwise component 3 would make $|A'| > B$. This implies that α_i is of the form (x_i, y_i, i)
- if $\alpha_i = (x_i, y_i, i)$ and $\alpha_{i+1} = (x_{i+1}, y_{i+1}, i+1)$ then $y_i = x_{i+1}$ and $m = n$ implies that α_m is of the form (x_m, x_1, m). Otherwise component 2 would make $|A'| > B$
- Finally, $m \geq n$ otherwise $|A' \cap T| > B$. This implies that $m = n$. Remark that this also implies that $|A' \cap T| = 0$.

In other words, A' encodes a tour $i_1, i_2, \ldots, i_n, i_1$ in our instance of the TSP. Furthermore, the size of A' if we follow its construction as stated above is

$$\begin{aligned}|A'| = |\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| &\leq B\\ |\mathcal{C}_1| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| &\leq B\\ |\mathcal{C}_1| + 4n + 2 + 2n(4n-1) &\leq B\\ |\mathcal{C}_1| + 2 + 2n(1+4n) &\leq 2 + K + 4n(1+2n)\\ |\mathcal{C}_1| &\leq K + 2n\\ 2n + d(i_1, i_2) + \cdots + d(i_n, i_1) &\leq K + 2n\\ d(i_1, i_2) + \cdots + d(i_n, i_1) &\leq K\end{aligned}$$

And so the tour is actually a solution for our instance of the TSP.

Theorem 3. ***(IO3)*** *is NP-complete.*

Proof. The encoding remains the same that the one for (IO2) except for the first component. The non-instantiated distances before position p are erased by intersection with $C_\perp$. (Notice that $C\perp \cap C_{i\in\{0,1,2\}} = C_\perp$)

$$\begin{aligned}\mathcal{A}_{i,j,p} = &(V(C_\perp + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{p-1}\\ &VC_{d(i,j)}(V(C_0 + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{n-p}\\ &V(\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p})\end{aligned}$$

The bound for this problem is $B = K + n(2 + 2n + |\mathcal{C}_2| + |\mathcal{C}_3|) = K + n(2 + 8n + 8n^2)$ which corresponds to K and the part of the first automaton of the tour that does not disappear by intersection before the intersection with T.

4 Conclusion

We showed that determining the best way to intersect a set of automata is an intractable problem. This compels finite state applications to look for clever heuristics. In our own implementation we choose to perform intersections according to the ascending size of the automata. [10] gives several other heuristics. Another possibility is to approximate the intersection, see [11].

References

1. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. **125**(2) (1994) 315–328
2. Yu, S.: On the state complexity of combined operations. In Ibarra, O.H., Yen, H.C., eds.: CIAA. Volume 4094 of Lecture Notes in Computer Science., Springer (2006) 11–22
3. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)
4. Karakostas, G., Lipton, R.J., Viglas, A.: On the complexity of intersecting finite state automata. In: IEEE Conference on Computational Complexity. (2000) 229–234
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press (1990)
6. Moortgart, M.: Categorial Type Logics. In van Benthem, J., ter Meulen, A., eds.: Handbook of Logic and Language. Elsevier (1996)
7. Perrier, G.: La sémantique dans les grammaires d'interaction. Traitement Automatique des Langues **45**(3) (2004) 123–144
8. Kahane, S.: Polarized unification grammars. In: ACL, The Association for Computer Linguistics (2006)
9. Bonfante, G., Guillaume, B., Perrier, G.: Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. (2004)
10. Tapanainen, P.: Applying a Finite-State Intersection Grammar. In: Finite-State Language Processing. MIT (1997)
11. Yli-Jyrä, A.M.: Simplifications of intermediate results during intersection of multiple weighted automata. In Droste, M., Vogler, H., eds.: Weighted Automata: Theory and Applications. (2004)

Developing a Finite-State Morphological Analyzer for Urdu and Hindi

Tina Bögel, Miriam Butt, Annette Hautli, and Sebastian Sulger

Universität Konstanz

Abstract. We introduce and discuss a number of issues that arise in the process of building a finite-state morphological analyzer for Urdu, in particular issues with potential ambiguity and non-concatenative morphology. Our approach allows for an underlyingly similar treatment of both Urdu and Hindi via a cascade of finite-state transducers that transliterates the very different scripts into a common ASCII transcription system. As this transliteration system is based on the XFST tools that the Urdu/Hindi common morphological analyzer is also implemented in, no compatibility problems arise.

1 Introduction

As part of the ParGram (Parallel Grammar) project [1], [2], we are developing a grammar for the South Asian language Urdu.[1] Very few resources exist for this language, in particular, no broad-coverage finite-state morphological analyzer exists to date. Part of the Urdu Grammar project is therefore to build a finite-state morphological analyzer for Urdu and to connect it up with the syntax via the morphology-syntax interface [3] defined for Lexical-Functional Grammar (LFG; [4]).

Current features of the Urdu ParGram project in the context of parallel grammar development have already been discussed elsewhere [5]. In this paper, we focus on some issues that have arisen with respect to the morphological analyzer in particular. All the (larger) ParGram grammars to date include a finite-state morphological analyzer that interfaces with the syntax. These morphological analyzers are generally built with the Xerox finite-state technology tools and follow the methodology established by [6]. The finite-state tools and the solutions already proposed by [6] prove to be more than adequate to meet the challenges posed by Urdu. However, some interesting issues do arise with respect to 1) the script and tokenization (section 2); 2) reduplication (section 3) ; 3) potentially ambiguous information at the morphology-syntax interface (section 4).

2 Two Different Scripts, One Representation

Urdu is structurally almost identical to Hindi. The major difference is that the vocabulary of Urdu bears more Persian/Arabic influences, while the vocabulary

[1] Thanks go to Tafseer Ahmed for helping us understand some issues with respect to the script and the morphology.

of Hindi is more Sanskrit based. Both are ultimately descended from a version of Sanskrit (i.e., are Indo-European). Urdu as a separate version of the language came into being when the Moghuls invaded the Indian subcontinent. The language of their court was Persian, which came into contact with a local language generally referred to as Hindustani (or Hindi). The very Persianized version of this language came to be known as Urdu.[2]

This brief historical sketch is of relevance because lexical items borrowed in from Persian tend to behave differently (i.e., have different inflectional possibilities). However, questions of lexical and morphological origin tend to be minor issues. A more major issue is that Urdu and Hindi are written in very different scripts. Urdu is written with a version of the Arabic script.[3] Hindi, in contrast, is written in *Devanagari*, a phonetic-based script passed down over the millenia from Sanskrit.

2.1 A Common Transliteration System

(1) and (2) show a couplet (162,9) from the poet Mirza Ghalib (1797–1869): (1) is written in Urdu, (2) is the same couplet, but written in Devanagari (Hindi). Note that Urdu is written right-to-left, whereas Hindi is written left-to-right.

ہاں بھلا کر ترا بھلا ہوگا
اور درویش کی صدا کیا ہے

(1)

हां भला कर तिरा भला होगा
और दर्वेश की सदा क्या है

(2)

Although the two writing systems differ markedly, the languages they encode are structurally almost identical. Given this fact, our general strategy in building a morphological analyzer is to produce a resource that can be used for text written in both Urdu and Hindi. This involves building a transliteration system that goes from whichever script is being processed to a common ASCII base and then being able to generate back out from the common ASCII base to either one of the scripts. That is, both the texts in (1) and (2) are rendered as in (3).

[2] Modern Hindi naturally also bears traces of language contact with Persian, but not as markedly as Urdu.

[3] Unicode fonts for this script have only recently been developed (e.g., see `http://www.crulp.org`; [7]).

(3) hAN bHalA kar tirA bHalA hOgA
yes good.M.Sg do then good be.Fut.M.Sg
Or darvES kI sadA kyA he
and dervish Gen.F.Sg call.F.Sg what be.Pres.3.Sg
'Yes, do good then good will happen, what else is the call of the dervish.'

Our transliteration is based on proposals by [8]. Capitalized vowels indicate length, H marks aspiration, N nasalization, S stands for ∫ and other capitalized consonants indicate retroflexes.

A transliterator in accordance with our overall strategy has been implemented by [9]. Malik's HUMTS (Hindi-Urdu Machine Transliteration System) is written as a cascade of finite-state transducers that transliterate from the Urdu and Hindi scripts to SAMPA [10], a common underlying phonetic ASCII alphabet, and back out from SAMPA to the two differing scripts. SAMPA has been developed to enable coverage of all the world's languages; however, for the purposes of Urdu, it is unwieldy and very difficult to read. In integrating Malik's work into the Urdu grammar, we will therefore use Glassman's transliteration system. Beyond the simple conversion of letters that is necessary to do this, we anticipate no further (major) problems as HUMTS was written with the same XFST tools used in our Urdu grammar project.[4]

2.2 Future Morphology: Illustrating Tokenization Problems

Writing a transliterator that takes one script as an input and is able to output another script is not an easy task. Many of the problems that arise are discussed in Malik's work. In terms of the Urdu Grammar, most relevant to us are problems of tokenization. In particular, problems associated with the future morphology in Urdu/Hindi was one of the first to arise.

We already had an example of future usage in (1) and (2). An inspection of each example will quickly reveal one of the very general problems in dealing with the Urdu script: while in Hindi, each word is clearly demarcated and easy to identify, in Arabic-based scripts in general, word boundaries are very difficult to identify. One must basically know the language (i.e., be able to access the lexical items) in order to be able to read the script.[5]

Beyond this very general problem, the scripts also encode differences of opinion as to what exactly a word is. This is illustrated in (1) and (2) with respect to the future form of 'be' *hOgA*. In (1) it is expressed by the last two letter groups on line one (reading from right to left). In (2), the form is expressed by just one letter group: the last one (reading from left to right) on line 1. This difference in encoding reflects an on-going historical change.

The future in Urdu/Hindi is formed as shown in the paradigm (4) for the stem *mAr* 'hit/kill'. The stem is followed by information about person and number

[4] Related work has been done by [11], who provide a transliterator into ASCII as well, but do morphological analysis using the Functional Morphology Toolkit [12].

[5] The same is not true for Devanagari, which, being phonetically based, allows a sounding out of the words.

(*UN/E/EN/O*), to which the future marker *g* is attached. This, finally, is followed about information about number and gender.

(4) **Urdu Future Paradigm**

	Singular **M/F**	**Plural** **M/F**	**Respect** (Ap) **M/F**	**Familiar** (tum) **M/F**
1st	mAr-UN-g-A/I	mAr-EN-g-E/I		
2nd	mAr-E-g-A/I		mAr-EN-g-E/I	mAr-O-g-E/I
3rd	mAr-E-g-A/I	mAr-EN-g-E/I		

mAr- 'hit'

The future paradigm is thus a relatively complex assemblage of morphological pieces. The person/number morphology is identical to that used in the subjunctive paradigm, shown in (5). To these essentially subjunctive forms, a *-g-* is attached to mark the future. The consensus in the available literature is that the future *-g-* is derived from a Sanskrit participle of the verb *gā* 'go' [13], [14]. This analysis immediately explains the gender and number agreement morphology (*A/I/E*) exhibited by the future: Participles functioned like adjectives and so generally had number and gender agreement morphology. This morphology has simply been retained in all the verb forms in Urdu/Hindi that derive from old participles (i.e., the perfect, imperfect and progressive forms), including the future.

(5) **Urdu Subjunctive Paradigm**

	Singular	**Plural**	**Respect** (Ap)	**Familiar** (tum)
1st	mAr-UN	mAr-EN		
2nd	mAr-E		mAr-EN	mAr-O
3rd	mAr-E	mAr-EN		

mAr- 'hit'

The old participle of the verb *gā* 'go' used to form its own word. Indeed, as recently as a century ago, clitics like the emphatic *hI* 'even/only' could intrude between the *-g-* and the stem+subjunctive morphology. This is illustrated in (6).

(6) kɑh-ũ=hi=ga
say-1.Sg=Emph=Fut.M.Sg
'I will say (it), of course.' (Hindi, from Kellogg 1893:§399)

These examples suggest that while the old participle was no longer functioning as an independent word a century ago, it retained some prosodic independence and was probably functioning as a clitic (indicated by the glossing with '='). This is entirely consonant with well known processes of historical change whereby words are reanalyzed as clitics and then reanalyzed further as inflectional morphology as they move from expressing content words to functional elements (e.g., [15], [16]).

The examples in (6) are only marginally possible in modern Urdu, whereas speakers of Hindi tend to reject them outright. This difference in native speaker

judgements may or may not be correlated with the differences encoded in the writing system. Recall that in written Hindi, the future is expressed in one word together with the subjunctive stem. In Urdu however, the stem+subjunctive and the future+number+gender are generally written as two separate words.

In both languages all the pieces of morphology involved nevertheless perform exactly the same function, so our morphological analyzer should treat them in parallel. In the morphological analyzer, the future *-g-* is treated as an inflectional morpheme and a form like *mArEgI* would be analyzed as in (7).

(7) mArEgI ⇔
mAr+Verb+Subjunct+2P+Sg+Fut+Fem
mAr+Verb+Subjunct+3P+Sg+Fut+Fem

The tokenizer thus has to turn the Urdu input of *mArE gI* into *mArEgI*. This in and of itself does not present a problem, since the deletion of white space is not a problem. In principle, since forms like *marE* are also words in their own right, a serious ambiguity problem could arise. However, as *gI/gA/gE* are not words in their own right,[6] we do not anticipate serious problems with our basic approach.

In sum, the future morphology discussed here provides a good example of the potentially problematic factors that must be dealt with. Another, perhaps more interesting problem posed by Urdu is that of reduplication.

3 Reduplication

Urdu/Hindi, like most of the South Asian languages, tends to use reduplication quite frequently [17]. All content words can generally be reduplicated and the effect of the reduplication is to either strengthen/emphasize the original word or to express something like "and those kinds of things".

(8) a. kHAnA vAnA
food.M.Sg. Redup
'food and those kinds of things'
b. tHanDA tHanDA
cold.M.Sg. Redup
'ice cold (cold cold)'
c. kHAtA vAtA
eat.Impf.M.Sg Redup
'he is eating and such'

There are two different kinds of reduplication strategies. In the one illustrated by (8a), the onset of the content word is replaced with another consonant. This consonant could be either *v*, *ṯ* (T) or ∫ (S). In another strategy ((8b)), the word is simply repeated. We will refer to this latter strategy as *full word reduplication*, the former strategy is generally described as *echo formation* or *echo reduplication*.

[6] *gA* is a word, namely the bare form of the verb 'sing'. However, this would never (or rarely) occur in conjunction with a subjunctive verb.

3.1 General Strategy

Generally, reduplications are written as seperate words in both Urdu and Hindi. The fundamental problem facing the tokenizer is thus the fact that a reduplicated item must be recognized. The transliteration system will yield two words, as shown in (9), for example, which are separated by white space.

(9) calnA valnA
walk.Inf.M.Sg Redup
'walking and such things'

Our morphological analyzer basically follows the solution for full stem reduplication presented by [6] for Malay. The basic lexicon built independently of reduplication for nouns, verbs, adjectives and other content words interacts with reduplicating regular expressions.

The morphological analysis of reduplications as in (9) is shown in (10). That is, within the morphological analyzer, the reduplicated form is simply registered via the tag +Redup and is passed on as such to the Urdu grammar, which can decide how to use this information (or whether to use the very subtle semantic information implied by reduplication at all).

(10) cal+Verb+Inf+Masc+Sg+Redup

In the Malay example presented by Beesley and Karttunen (B&K), the original word and the reduplicated part are merged into a single word. Our implementation differs from theirs in that we need to deal with the white space. Currently, we do this by introducing the multiword `%^Hyphen` into the *lexc* source file (which encodes the basic lexicon plus the morphological continuation classes). When dealing with reduplication, we thus internally represent the two words involved as being connected with a hyphen.

Reduplication itself is managed, as in B&K, via the introduction of the multicharacter brackets `"^["` and `"^]"` in order to mark the domain of reduplication. The right bracket is additionally marked with the characters ^2. The lower side of the finite-state network thus ends up being marked up via the brackets `"^["` and `"^2^]"`. As discussed in B&K, the *compile-replace* algorithm can be applied to the resulting network — compile-replace essentially treats the marked up lower side as a regular expression which is to be interpreted. The overall effect is that something like *calnA* ends up being doubled to *calnA-calnA* due to the ^2 specification (and the addition of the hyphen).

We illustrate our approach more concretely with respect to just the adjective 'strange' in terms of full word reduplication. The code illustrates a simple *lexc* file which allows for two possibilities for all adjectives. In one, a bracketing is begun which is intended for the reduplicated version. This is notated by the regular expression ^2, which results in the doubling of the material delimited by the brackets. The bracket filter from B&K removes any unmatched brackets that

may have resulted from paths which contain only one bracket.[7] The bracket filter and the lexc file are composed, and the compile-replace algorithm is applied to the resulting network. Compile-replace translates the reduplication [...]^2 into well-formed strings of this type: [...]%^Hyphen[...]%^Hyphen. In a last step a regular expression (illustrated below as hyph.regex) then replaces the hyphens (%^Hyphen) used for internal management of the reduplicated forms with a white space.

```
* !AdjRedup.txt, lexc file just for ajIb 'strange'
* Multichar_Symbols
* +Adj +Unmarked +Redup +Intensifier
*
* Lexicon Root
* 0:^[[{ Unmarked ;
*        Unmarked;
*
* Lexicon Unmarked
* ajIb Ending ;    !the adjective 'strange'
*
* Lexicon Ending
* +Adj+Unmarked+Redup+Intensifier:}%^Hyphen]^2^]  # ;
* +Adj+Unmarked:0 # ;
*********************************************************
* ! bracketfilter.regex --- bracket filter from B&K
* [ ~ [ ?* "^[" ~$["^]"] ] & ~[ ~$["^["] "^]" ?* ] ];
*********************************************************
* !hyph.regex, removes '%^Hyphen' and inserts a white space
* [ %^Hyphen -> 0 || %^Hyphen ?* _ ]
* .o.
* [ %^Hyphen -> " " ] ;
```

3.2 Echo Reduplication

Recall that echo reduplication further requires the use of a different consonant/onset in the reduplicated form ((11)). In order to deal with this further complication, we introduce replace rules to effect the phonological change and further make use of flag diacritics (`@P.ECHO.v@` in the rules below, cf. B&K) in order to flag that the echo type of reduplication has taken place.

(11) AlU vAlU
potato.M Redup
'potatoes and such'

[7] This can be done differently, by controlling the continuation paths of the lexc file more tightly, however, in the long run, this results in a conceptually more complex structure of the lexc file and it is thus preferable (and more efficient) to simply apply the bracket filter on unwanted paths.

The phonological replace rules shown below exemplify just two cases. In reduplicating contexts (i.e., contexts which have been marked up by a Hyphen), either the first consonant[8] is replaced by a *v*, or if there is no onset as in (11), a *v* is inserted. We have formulated similar rules for reduplications with ṯ (T) or ʃ (S).

`Cons` stands for the set of consonants (this is predefined). The phonological replacement rule below thus operates on Consonants or Vowels (listed here individually, though this could also be done differently). Consonants are replaced by a *v* (or T or S in the rules not shown here). If there is no consonant, then a *v* (or T or S) is inserted before the vowel.

```
Cons -> v || ?* %^Hyphen _ ?* "@P.ECHO.v@"
.o.
a -> v a , e -> v e , i -> v i, o -> v o,
u -> v u || ?* %^Hyphen _ ?* "@P.ECHO.v@";
```

We thus implement the two differing reduplication strategies by using a range of FST methodologies. Full word reduplication is treated via a markup that feeds into the compile-replace algorithm. Echo reduplication additionally requires the use of phonological replace rules and flag diacritics.

Overall, allowing for reduplication results in a threefold increase of the basic lexicon. However, this increase is dealt with in a conceptually elegant manner and can be achieved by writing just a few extra lines of code (regular expressions) that are composed with the source lexc file. In our approach, we have based ourselves on the B&K solution — the successful application of their basic idea to Urdu provides a confirmation of the basic principles of finite-state based non-concatenative morphology formulated by B&K.

4 Issues in Potential Ambiguity

In this final section of the paper, we address some issues that arise with respect to the morphology-syntax interface. Recall from the discussion of the Urdu/Hindi future in section 2.2 that the future is formed in combination with subjunctive forms. Our present analysis of future forms is thus as in (12).

(12) mArUNgI ⇔
mAr+Verb+Subjunct+1P+Sg+Fut+Fem

From the perspective of the syntax (and semantics), marking these forms as subjunctive as well as future is unnecessary as every future form also carries some subjunctive meaning with it (this has been dubbed *contingent future* in the literature). Experience gathered with respect to the German ParGram grammar [1] has shown that it is ultimately better to eliminate tags of this kind from the

[8] So far, all the words in our lexicon have just simple consonants as onsets — this seems to be a strong tendency, if not a hard phonotactic constraint of Urdu.

morphology, since dealing with them complicates the morphology-syntax interface. Given that there are simple subjunctive uses as in (13), the interpretation of the +Subjunct tag within the morphological component will need to differ depending on whether it is found in conjunction with future morphology or not.

(13) mArUN ⇔
mAr+Verb+Subjunct+1P+Sg

We have therefore decided to eliminate the +Subjunct tag from the morphological analysis of future forms altogether even though the morphology involved is in actual fact the subjunctive morphology.

A somewhat different version of this same problem is found with respect to Urdu/Hindi infinitives as in *dEkHnA* 'to look/looking', which can also be used as verbal nouns. To date, the morphology provides analyses as in (14).

(14) dEkHnA ⇔
dEkH+Verb+Inf+Masc+Sg

It will be imperative to know that infinitives can also function as nouns in the grammar. It might therefore be necessary to anticipate this in the morphology and provide both the analyses in (15) for the syntax.

(15) dEkHnA ⇔
dEkH+Verb+Inf+Masc+Sg
dEkH+Noun+Deverb+Masc+Sg

However, this would result in quite a bit of ambiguity within the morphological analyzer. Our current solution, shown in terms of LFG functional annotations in (16) is therefore to add the information that this form could optionally (denoted by the round brackets) be used as a noun whose type is deverbal as part of the definition of the morphology-syntax interface.

(16) +Inf ((↑NTYPE) = deverbal).

The abstract morphological tag +Inf is thus annotated with the functional information that it could also be used as a noun, in which case it is deverbal. This solution pushes the ambiguity from the morphology into the syntax, but since the syntax can eliminate the ambiguity by means of unifying in other information, it may be better to deal with the ambiguity in the syntax, rather than in the morphology, where no contextual information is available. We are currently experimenting with both possible solutions to determine the better one.

5 Conclusion

In this paper, we have introduced and addressed a number of issues that arise in the process of building a finite-state morphological analyzer for Urdu. Our

approach allows for an underlyingly similar treatment of both Urdu and Hindi via a cascade of finite-state transducers that transliterates the very different scripts into a common ASCII transcription system. As this transliteration system is based on the XFST tools that the Urdu/Hindi common morphological analyzer is also implemented in, no compatibility problems arise.

We further explored reduplication in Urdu, again basing ourselves on solutions proposed with respect to XFST and show how differing reduplication patterns in Urdu/Hindi can be dealt with elegantly with the finite-state methods proposed by B&K.

Finally, we addressed some potential ambiguity problems and discussed different ways of solving them. The discussion here mainly revolves around where and how information should be encoded with respect to the morphology-syntax interface that has been defined between finite-state morphological analyzers and LFG grammars as part of the ParGram project.

References

1. Butt, M., King, T.H., Niño, M.E., Segond, F.: A Grammar Writer's Cookbook. CSLI Publications (1999)
2. Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The Parallel Grammar project. In: Proceedings of COLING, Workshop on Grammar Engineering and Evaluation, Taipei (2002) 1–7
3. Kaplan, R.M., Maxwell III, J.T., King, T.H., Crouch, R.: Integrating finite-state technology with deep LFG grammars. In: Proceedings ESSLLI, Workshop on Combining Shallow and Deep Processing for NLP. (2004)
4. Dalrymple, M.: Lexical Functional Grammar. Academic Press (2001)
5. Butt, M., King, T.H.: Urdu in a parallel grammar development environment. Language Resources and Evaluation (2007) Special Issue on Asian Language Processing: State of the Art Resources and Processing. To Appear.
6. Beesley, K., Karttunen, L.: Finite State Morphology. CSLI Publications, Stanford, CA (2003)
7. Rahman, S., Hussain, S.: Development of character based Urdu Nastaleeq font. Asian Media and Communication Bulletin **33**(2) (2003)
8. Glassman, E.H.: Spoken Urdu. Nirali Kitaben, Lahore (1977)
9. Abbas Malik, M.: Hindi Urdu machine transliteration system. MSc Thesis, Paris 7 (2006)
10. Wells, J.: SAMPA computer readable phonetic alphabet. In Gibbon, D., Moore, R., Winski, R., eds.: Handbook of Standards and Resources for Spoken Language Systems. Mouton de Gruyter, Berlin and New York (1997)
11. Humayoun, M., Hammarström, H., Ranta, A.: Urdu morphology, orthography and lexicon extraction. In Farghaly, A., Megerdoomian, K., eds.: Proceedings of the 2nd Workshop on Computational Approaches to Arabic Script-based Languages. (2007) 59–66 Held at the Stanford LSA 2007 Institute.
12. Forsberg, M., Ranta, A.: Functional morphology. In: Proceedings of Ninth ACM SIGPLAN International Conference of Functional Programming. (2004) 213–223
13. Kellogg, S.H.: Grammar of the Hindi Language. Munshiram Manoharlal Publishers Pvt. Ltd., Delhi (1893) Second Edition, reprinted 1990.

14. McGregor, R.: The Language of Indrajit of Orchā. Cambridge University Press, Cambridge (1968)
15. Harris, A.C., Campbell, L.: Historical Syntax in Cross-Linguistic Perspective. Cambridge University Press, Cambridge (1995)
16. Hopper, P.J., Traugott, E.C.: Grammaticalization. Cambridge University Press, Cambridge (1993)
17. Abbi, A.: Reduplication in South Asian Languages. An Areal, Topological and Historical Study. Allied, New Delhi (1991)

Perfect Hashing Tree Automata

Jan Daciuk

Department of Knowledge Engineering, Gdańsk University of Technology, Poland

Abstract. We present an algorithm that computes a function that assigns consecutive integers to trees recognized by a deterministic, acyclic, finite-state, bottom-up tree automaton. Such function is called minimal perfect hashing. It can be used to identify trees recognized by the automaton. Its value may be seen as an index in some other data structures. We also present an algorithm for inverted hashing.

1 Introduction

Hashing [1] is a technique where a key is transformed into an integer in a limited range with a *hash function*. Usually, there are far more possible keys than integers in the range, so conflicts where different keys are mapped into the same integers are unavoidable. However, in certain contexts, it is possible to map n keys without any conflicts. A function that implements it is called a *perfect hash function*. If it maps n keys into a consecutive range of n integers, it is called a *minimal perfect hash function*.

The nature of a hash function and its application depends closely on the hash key. Minimal, deterministic, acyclic, finite state automata (FSAs) provide a minimal perfect hash function on strings [2], [3]. This allows for fast and compact representation of dictionaries storing arbitrary information associated with words. An insight from perfect hashing with FSAs is useful in developing perfect hashing with deterministic, acyclic, bottom-up tree automata (DTAs), although the latter case is more complex. DTAs store a finite set of finite trees. Trees are ubiquitous in both computer science and natural language processing. They are used e.g. for storing the result of parsing a program or a sentence. A language that is best suited for parsing with DTAs is XML. It is widely used both in computer science and in natural language processing. For example, in natural language processing, it is used for annotating corpora. Perfect hashing with tree automata implements a mapping from trees to integers. It can be used for identification of trees, which allows for e.g. retrieval of arbitrary information associated with the given tree, like all locations in a corpus where the given parse tree occurs. The inverse mapping has an even greater potential, as a tree automaton can be used as a compact representation for a forest of trees. A mapping from an integer to a tree makes it possible to retrieve fast a given tree. A dictionary associating words with trees can be implemented as a perfect hash FSA associating words with numbers, a vector associating word numbers with tree numbers, and a perfect hash DTA.

The rest of the paper is structured as follows: Section 2 provides basic definitions, while Section 3 discusses issues related to counting trees in a tree automaton. An implementation of a minimal perfect hash function and its inverse is given in Section 4, while their complexity is investigated in Section 5. The paper ends with conclusions given in Section 7.

2 Basic Definitions

A *finite-state, bottom-up tree automaton* [4] is defined as $A = (Q, \Sigma, \Delta, F)$, where Q is a finite set of states, Σ is a finite set of symbols called the *alphabet*, $\Delta \subset \bigcup_{i=0}{}^{m} \Sigma \times Q^{m+1}$ is a final set of *transitions*, and $F \subseteq Q$ is a set of *final states*. Another name for bottom-up is *frontier-to-root*. Another name for final states is *accepting states*. This definition is similar to the definition of finite-state automata, except for two differences: there is no initial (start) state, and a transition is a relation between an alphabet symbol and an arbitrary number of states (and not necessarily two states, as in case of finite-state automata). In a *deterministic*, finite-state, bottom-up tree automaton (or a DTA for short), for each $(\sigma, q_1, \ldots, q_m) \in \Sigma \times Q^m$, there is at most one $q \in Q$ such that $(\sigma, q_1, \ldots, q_m, q) \in \Delta$. In that case, we can define a function δ:

$$\delta_m(\sigma, q_1, ..., q_m) = \begin{cases} q \text{ if } q \in Q \text{ is such that } (\sigma, q_1, ..., q_m, q) \in \Delta \\ \perp \text{ if no such } q \in Q \text{ exists} \end{cases} \quad (1)$$

States $q_1, \ldots, q_m$ are *source states*, while q is a *target state*. Finite-state automata accept strings. Tree automata accept trees. Trees are defined as follows:

1. Each symbol $\sigma \in \Sigma$ is a tree.
2. For each $t = \sigma(t_1, \ldots, t_m)$, where $\sigma \in \Sigma$, and $t_1, \ldots, t_m$, $m \geq 0$ are trees, t is a tree.

Any subset of all trees defined over an alphabet Σ is called a *tree language* T_Σ. We can define an extended transition function on trees:

$$\delta_A(t) = \begin{cases} \delta_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ \delta_m(\sigma, \delta_A(t_1), \ldots, \delta_A(t_m)) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases} \quad (2)$$

A language of a state q in an automaton A is a set of trees such that the extended transition function returns q for each of them:

$$L_A(q) = \{t \in T_\Sigma : \delta_A(t) = q\} \quad (3)$$

A language of the whole automaton A is the union of the languages of all its final states:

$$L(A) = \bigcup_{q \in F} L_A(q). \quad (4)$$

A language of a transition $\tau = (\sigma, q_1, \ldots, q_m, q) \in \Delta$ is the subset of $L_A(q)$ recognized by following τ:

$$L_A(\tau) = \begin{cases} \sigma & \tau = (\sigma, q) \\ \bigcup_{(t_1,\ldots,t_m)\in L(q_1)\times\ldots\times L(q_m)} \sigma(t_1, \ldots, t_m) & \tau = (\sigma, q_1, \ldots, q_m, q) \end{cases} \quad (5)$$

A DTA A is *acyclic*, when $L(A)$ is a finite set of finite trees. From this moment on in the paper, when we refer to automata, we mean deterministic, acyclic finite-state, bottom-up tree automata without useless states, unless otherwise specifically stated.

3 Numbering Trees in a DTA

A tree t has number i (counting from 0 to $n-1$, where $n = |L(A)|$ is the number of trees recognized by the automaton) if there are i trees that precede it in an order imposed by the automaton. To compute i, we have to count the trees that precede t. The first step is to compute the number of trees that precede the current tree t in the language of the state $\delta_A(t)$:

$$\iota_A(t) = |\{t' : \delta_A(t') = \delta_A(t) \wedge t' \prec_A t\}| \quad (6)$$

This can be done recursively. Let $t = \sigma(t_1, \ldots, t_m)$, $\tau = (\sigma, \delta_A(t_1), \ldots, \delta_A(t_m), q)$, and $q = \delta_A(t)$. Then $\iota_A(t)$ is the sum of the number of trees that precede t but use the same transition τ, and the number of trees recognized while following transitions preceding τ:

$$\iota_A(t) = \rho_A(t) + \sum_{\tau'=(\sigma', q'_1, \ldots, q'_{m'}, q) \prec_A \tau} |L_A(\tau')| \quad (7)$$

The language of a transition (see Equation (5)) can also be defined recursively:

$$L_A(\tau) = \begin{cases} \sigma & \text{if } \tau = (\sigma, q) \\ \bigcup_{(t'_1,\ldots,t'_m)\in L_A(q_1)\times\ldots\times L_A(q_m)} \sigma(t'_1, \ldots, t'_m) & \text{if } \tau = (\sigma, q_1, \ldots, q_m, q) \end{cases} \quad (8)$$

Its cardinality is:

$$|L_A(\tau)| = \begin{cases} 1 & \text{if } \tau = (\sigma, q) \\ \prod_{i=1}^{m} |L_A(q_i)| & \text{if } \tau = (\sigma, q_1, \ldots, q_m, q), m > 0 \end{cases} \quad (9)$$

The key component in (9) is $|L_A(q)|$. We rewrite definition (3) recursively:

$$L_A(q) = \bigcup_{\tau=(\sigma,q_1,\ldots,q_m,q)\in\Delta, m\geq 0} |L_A(\tau)| \quad (10)$$

so that its cardinality can easily be computed as:

$$|L_A(q)| = \sum_{\tau=(\sigma,q_1,\ldots,q_m,q)\in\Delta, m\geq 0} |L_A(\tau)| \quad (11)$$

To compute $\rho_A(t)$, i.e. the number of trees that reach $q = \delta_A(t)$ by the same transition τ and precede t, we need to introduce some order of trees in the language $L_A(\tau)$. Let $\tau = (\sigma, q_1, \ldots, q_m, q)$, and let $\mathrm{next}(t_i)$ be the next subtree in $L_A(q_i)$. Then $\forall_{1 \le j < k \le m} \sigma(t'_1, \ldots, \mathrm{next}(t'_j), \ldots, t'_k, \ldots, t'_m) \prec_A \sigma(t'_1, \ldots, t'_j, \ldots, \mathrm{next}(t'_k), \ldots, t'_m)$, where $t'_i \in L_A(q_i)$. So

$$\rho_A(t) = \begin{cases} 1 & \text{if } t \in \Sigma \\ \sum_{i=1}^{m} \iota_A(t_i) \cdot \prod_{j=i+1}^{m} |L_A(\delta_A(t_j))| & \text{if } t = \sigma(t_1, \ldots, t_m) \in T_\Sigma - \Sigma \end{cases} \tag{12}$$

In practice, we would use ${\rho_A}^i$ defined as:

$${\rho_A}^i(t) = \begin{cases} 1 & \text{if } i = 0 \\ {\iota_A}^i(t) = {\rho_A}^{i-1}(t) \cdot |L(\delta_A(t_i))| + \iota_A(t_i) & \text{if } 1 \le i \le m \end{cases} \tag{13}$$

Thus, $\rho_A(t) = {\rho_A}^m(t)$.

A tree t is recognized if $\delta_A(t) \in F$. However, there may be more than one final state, and languages of final states are disjoint. We assume that final states $f_i \in F$ are ordered: $f_1 \prec_A \ldots \prec_A f_{|F|}$. A tree number for a tree t is thus $\iota_A(t)$ plus $\sum_{F \ni f' \prec_A \delta_A(t)} |L_A(f')|$.

4 Perfect Hash Function

The perfect hash function is given on Figure 1. The argument is a tree t, for which we want to obtain the hash value. A call to function rh in line 2 returns a pair $(\delta_A(t), \iota_A(t))$. The loop in lines 4–6 adds the number of trees belonging to languages of those final states that precede $\delta_A(t)$. If $t \notin L(A)$, i.e. either $\delta_A(t) \notin Q$ or $\delta_A(t) \notin F$, $h_A(t)$ returns -1. In function rh, the loop in line 13 finds numbers associated with subtrees t_i of t, and the loop in lines 14–20 computes ${\rho_A}^i(t)$. In lines 22–25, $\sum_{\tau'(\sigma', q'_1, \ldots, q'_{m'}, q) \prec_A \tau} |L_A(\tau')|$ is added to that value.

Inverse perfect hash function is given on Figure 2. The parameter n is the tree number. First, we process the final states f_i one by one, keeping the number of trees recognized at all preceding final states in the variable h. If $h \le n < h + |L_A(f_i)|$, then the tree number n belongs to the language of f_i, and it is $(n-h)$-th tree in that language. Function rh^{-1} takes two parameters: a state q being the root of a subtree, and a tree number among all trees in $L_A(q)$. All transitions reaching q are tried in order, and the number of trees recognized while following them is added to variable h. The process continues until for the current transition τ_i, $h \le n < h + |L_A(\tau_i)|$. Then the subtree we are looking for belongs to $L_A(\tau_i)$. To calculate the subtree number among $L_A(\tau_i)$, the subtree t is decomposed into individual subtrees $t_1, \ldots, t_{m_i}$ with roots being the states $q_1, \ldots, q_{m_i}$. The loop in lines 17–20 builds the tree t from its subtrees. Basically, it calculates the inverse of ${\rho_A}^i(t)$.

Note that the values $|L_A(q)|$ for all states, and $|L_A(\tau)|$ for all transitions can be calculated in advance and stored in appropriate states and transitions.

```
1:  function h_A(t)
2:      (q, v) ← rh(t);
3:      if q ∈ F ∧ v ≥ 0 then
4:          foreach p ∈ F : p ≺_A q
5:              v ← v + |L_A(p)|;
6:          end foreach;
7:          return v;
8:      else
9:          return -1;
10:     end if;
11: end function;

12: function rh(t = σ(t_1, ..., t_m))
13:     h ← 0; for i ← 1 to m do (q_i, v_i) ← rh(t_i) end for;
14:     for i ← 1 to m
15:         if q_i = ⊥ ∨ v_i = −1 then
16:             return -1;
17:         else
18:             h ← h · |L_A(q_i)| + v_i;
19:         end if;
20:     end for;
21:     q ← δ(σ, q_1, ..., q_m);
        Let N = |{(σ, q_1, ..., q_m, q) ∈ Δ}|;
        Let τ_1 ≺_A ... ≺_A τ_N, τ_i = (σ_i, q_{1_i}, ..., q_{m_i}, q);
22:     i ← 1;
23:     while τ_i < (σ, q_1, ..., q_m, q) do
24:         h ← h + |L_A(τ_i)|; i ← i + 1;
25:     end while;
26:     return (q, h);
27: end function;
```

Fig. 1. Perfect hash function

5 Computational Complexity

The time complexity of the perfect hash function given on Figure 1 is $\mathcal{O}(|t| + |F| + |\Delta|)$, where $|t|$ is the number of tree nodes – defined below in (14), $|F|$ is the number of final states, and $|\Delta| = |\{\tau : \tau \in \Delta\}|$ is the number of transitions in the automaton (this could be replaced with $|\Delta| = \sum_{\tau \in \Delta} |\tau|$, where $|\tau| = m + 1$, in case we were not to store $|L_A(\tau)|$ in transitions).

$$|t| = \begin{cases} 1 & \text{if } t = \sigma \in \Sigma \\ 1 + |t_1| + \ldots + |t_m| & \text{if } t = \sigma(t_1, \ldots, t_m) \in T_\Sigma - \Sigma \end{cases} \tag{14}$$

In function $h_A(t)$, we have one loop that executes at most $|F|$ times, and consists of adding a constant to a variable (a constant-time operation), as well as a call to function *rh*. In function *rh*, there is a loop in lines 23–25 that adds a constant to

```
1:   function h_A^{-1}(n)
2:       h ← 0;
3:       for i ∈ 1,...,|F| : f_1 ≺_A ... ≺_A f_{|F|} do
4:           if h + |L_A(f_i)| > n then
5:               return rh^{-1}(f_i, n − h);
6:           else
7:               h ← h + |L_A(f_i)|;
8:           end if;
9:       end for;
10:  end function;

11:  function rh^{-1}(q, n)
         Let N = |{(σ, q_1, ..., q_m, q) ∈ Δ}|;
         Let τ_1 ≺_A ... ≺_A τ_N, τ_i = (σ_i, q_{1_i}, ..., q_{m_i}, q);
12:      i ← 1; h ← 0;
13:      while h + |L_A(τ_i)| ≤ n do
14:          h ← h + |L_A(τ_i)|; i ← i + 1;
15:      end while;
16:      h ← n − h; th ← |L_A(τ_i)|;
17:      for j = 1,...,m_i do
18:          th ← th/|L(q_{j,i})|;
19:          t_j ← rh^{-1}(q_{j,i}, h/th); h ← h − (h/th);
20:      end for;
21:      return σ_i(t_1, ..., t_{m_i});
22:  end function;
```

Fig. 2. Inverse perfect hash function

a variable and increments a variable – also constant-time operations. As only $|\Delta|$ transitions can precede the current one across all calls to *rh*, this loop contributes an $\mathcal{O}(|\Delta|)$ component to the time complexity. Another loop in the same function in lines 14–21 contains constant-time operations and one recursive call to *rh*. Since there is one call to *rh* per every node of the tree, this contributes $|t|$ to the time complexity.

There is one important trick that eliminates the $\mathcal{O}(|\Delta|)$ component, and also reduces the size of the automaton as we are no longer forced to keep back-transitions. Instead of storing $|L_A(\tau)|$ in transitions, we store $\sum_{\tau' \in \Delta : \tau' \prec_A \tau} |L_A(\tau')|$ there. The computation in lines 24–26 is then no longer needed. The same can be done with final states, i.e. they can hold the number of trees recognized in those final states that precede the current one. This eliminates the $\mathcal{O}(|F|)$ component, giving us $\mathcal{O}(|t|)$ time complexity.

For the inverse perfect hashing, the time complexity is $\mathcal{O}(|t| + |F| + |\Delta|)$, regardless of the use of the trick described above. We also have to keep back transitions, as we need to find a transition (or a final state) with the appropriate value. The loop in function ${h_A}^{-1}(t)$ executes at most $|F|$ times, with all but one

run containing constant-time operations. The loop is finished with a single call to $rh^{-1}(q, n)$. Inside that function, the loop in lines 13–15 counts transitions – at most $|\Delta|$ across all calls, and it calls itself – once per tree node, giving the $|t|$ component.

6 Example

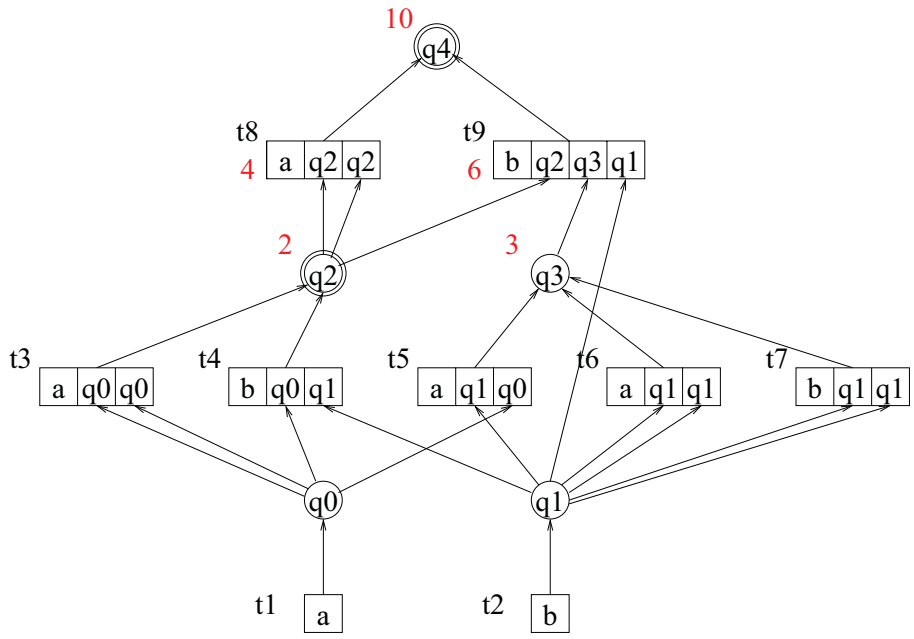


Fig. 3. A DTA $A = (\{q_0, \ldots, q_4\}, \{a, b\}, \Delta = \{t_1, \ldots, t_9\}, F = \{q_2, q_4\})$. Numbers by circles representing states, or by boxes representing transitions, give the cardinalities of the languages of states and transitions, respectively. If the number is not given, it is 1.

A DTA recognizing trees:

0. $a(a, a)$,
1. $b(a, b)$,
2. $a(a(a, a), a(a, a))$,
3. $a(b(a, b), a(a, a))$,
4. $a(a(a, a), b(a, b))$,
5. $a(b(a, b), b(a, b))$,
6. $b(a(a, a), a(b, a), b)$,
7. $b(b(a, b), a(b, a), b)$,
8. $b(a, (a, a), a(b, b), b)$,

9. $b(b(a,b),a(b,b),b)$,
10. $b(a(a,a),b(b,b),b)$,
11. $b(b(a,b),b(b,b),b)$

is given on Figure 3. Let us find the number $h_A(t)$ associated with $t = b(b(a,b), a(b,b),b)$. First, $rh(t)$ is called. It is called on t and recursively on each of its subtrees. We call $rh(b(b(a,b),a(b,b),b))$, take the first subtree and call $rh(b(a,b))$, then take the first subtree and call $rh(a)$. The last call returns $(q_0, 0)$, since q is set in line 22, and h, which is set in line 13, cannot be modified in loops in lines 14–21 and 24–26 as they are not executed. Similarly, $rh(b)$ returns $(q_1, 0)$. In $rh(b(a,b))$, the loop in lines 14–21 runs twice, but leaving $h = 0$. The loop in lines 24–26 increases h by $|L_A(t_3)|$ (the number of trees in languages of preceding transitions), making $rh(b(a,b))$ return $(q_2, 1)$. Next, $rh(a(b,b))$ is called, which in turn calls $rh(b)$ (twice), which returns $(q_1, 0)$ (twice) as described above. In $rh(a(b,b))$, the loop in lines 14–21 leaves $h = 0$, as no trees precede $a(b,b)$ in the language of t_6, but the loop in lines 24–26 increases that value by $|L_A(t_5)|$, i.e. the sun of cardinalities of languages of transitions preceding t_6, making the function return $(q_3, 1)$. Back in $rh(b(b(a,b),a(b,b),b))$, the value returned by $rh(b(a,b))$ (i.e. 1) is multiplied by $|L_A(q_3)| = 3$ before adding 1 returned by $rh(a(b,b))$. The result is 4. Since $rh(b)$ returned $(q_1, 0)$, and $|L_A(q_1)| = 1$, 4 is multiplied by 1, and then 0 is added. Afer having added $|L_A(t_8)|$, the value returned by $rh(b(b(a,b),a(b,b),b))$ is then $(q_4, 8)$. Since $q_2 \in F$ precedes q_4, and $|L_A(q_2)| = 2$, $h_A(t)$ returns $8 + 2 = 10$.

Now, let us find which tree has number 10. The process is illustrated in Table 1.

7 Conclusions

We have presented an efficient implementation for minimal perfect hashing with finite-state, deterministic, acyclic, bottom-up tree automata. It can be used for computing an index for trees that can further be used to access additional data structures associated with the trees. We have also shown how to compute the inverse perfect hash function, which can help to retrieve trees stored in a compact way in a tree automaton. Our implementation for minimal automata does not preserve the order of trees at input. However, when the automata are to be used in a static way, the order imposed by the automaton can easily be found.

The author wishes to thank Rafael Carrasco and Mikel Forcada for discussions about tree automata during the author's short visits to Alicante supported by the Spanish CICyT through grants TIN2006-15071-C03-01 and TIC2003-08681-C02-01, and for jointly developing a program that helped in testing ideas presented in this paper. Comments from reviewers helped in improving the paper.

References

1. Czech, Z.J., Havas, G., Majewski, B.S.: Fundamental study perfect hashing. Theoretical Computer Science **182** (1997) 1–143

$\boxed{h_A{}^{-1}(10)}$			
$i \leftarrow 1$; $p \leftarrow q_2$;			
$h \leftarrow 2$;			
$i \leftarrow 2$; $p \leftarrow q_4$;			
$rh^{-1}(q_4, 10-2)$;	$\boxed{rh^{-1}(q_4, 8)}$		
	$i \leftarrow 1$; $\tau_i = t_8$;		
	$h \leftarrow 4$; $i \leftarrow 2$;		
	$\tau_i \leftarrow t_9$;		
	$h \leftarrow 4$; $th \leftarrow 6$;		
	$j \leftarrow 1$; $th \leftarrow 3$;		
	$rh^{-1}(q_2, 1)$;	$\boxed{rh^{-1}(q_2, 1)}$	
		$i \leftarrow 1$; $\tau_i = t_3$;	
		$h \leftarrow 1$; $i \leftarrow 2$;	
		$\tau_i = t_4$;	
		$h \leftarrow 0$; $th \leftarrow 1$;	
		$j \leftarrow 1$; $th \leftarrow 1$;	
		$rh^{-1}(q_0, 0)$;	$\boxed{rh^{-1}(q_0, 0)}$
		$t_1 \leftarrow a$; $h \leftarrow 0$;	$\hookleftarrow a$;
		$j \leftarrow 2$; $th \leftarrow 1$;	
		$rh^{-1}(q_1, 0)$;	$\boxed{rh^{-1}(q_1, 0)}$
		$t_2 \leftarrow b$; $h \leftarrow 0$;	$\hookleftarrow b$;
	$t_1 \leftarrow b(a, b)$;	$\hookleftarrow b(a, b)$;	
	$h \leftarrow 1$;		
	$j \leftarrow 2$; $th \leftarrow 1$;		
	$rh^{-1}(q_3, 1)$;	$\boxed{rh^{-1}(q_3, 1)}$	
		$i \leftarrow i$; $\tau_i \leftarrow t_5$;	
		$h \leftarrow 1$; $i \leftarrow 2$;	
		$\tau_i \leftarrow t_6$;	
		$h \leftarrow 0$; $th \leftarrow 1$;	
		$j \leftarrow 1$; $th \leftarrow 1$;	
		$rh^{-1}(q_1, 0)$;	$\boxed{rh^{-1}(q_1, 0)}$
		$t_1 \leftarrow b$; $h \leftarrow 0$;	$\hookleftarrow b$;
		$j \leftarrow 2$; $th \leftarrow 1$;	
		$rh^{-1}(q_1, 0)$	$\boxed{rh^{-1}(q_1, 0)}$
		$t_2 \leftarrow b$; $h \leftarrow 0$;	$\hookleftarrow b$;
	$t_2 \leftarrow a(b, b)$;	$\hookleftarrow a(b, b)$;	
	$h \leftarrow 0$;		
	$j \leftarrow 3$; $th \leftarrow 1$;		
	$rh^{-1}(q_1, 0)$;	$\boxed{rh^{-1}(q_1, 0)}$	
	$t_3 \leftarrow b$; $h \leftarrow 0$;	$\hookleftarrow b$;	
$\hookleftarrow$	$\hookleftarrow$		
$b(b(a, b), a(b, b), b)$	$b(b(a, b), a(b, b), b)$		

Table 1. Call sequence in $h_A{}^{-1}(10)$.

2. Lucchiesi, C., Kowaltowski, T.: Applications of finite automata representing large vocabularies. Software Practice and Experience **23**(1) (Jan. 1993) 15–30
3. Revuz, D.: Dictionnaires et lexiques: méthodes et algorithmes. PhD thesis, Institut Blaise Pascal, Paris, France (1991) LITP 91.44.
4. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugier, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Draft book, available at http://www.grappa.univ-lille3.fr/tata (September 2002)

SynCoP – Combining Syntactic Tagging with Chunking Using Weighted Finite State Transducers

Jörg Didakowski

Berlin-Brandenburgische Akademie der Wissenschaften
Jägerstr. 22/23, 10117 Berlin

Abstract. This paper describes the key aspects of the system SynCoP (Syntactic Constraint Parser) developed at the Berlin-Brandenburgische Akademie der Wissenschaften. The parser allows to combine syntactic tagging and chunking by means of constraint grammar using weighted finite state transducers (WFST). Chunks are interpreted as local dependency structures within syntactic tagging. The linguistic theories are formulated by criteria which are formalized by a semiring; these criteria allow structural preferences and gradual grammaticality. The parser is essentially a cascade of WFSTs. To find the most likely syntactic readings a best-path search is used.

1 Introduction

In several natural language processing tasks such as information extraction and machine translation and especially in corpus linguistics information about syntactic structures is needed. The main interest lies in detecting syntactic relations between words. This is generally done by building *dependency structures* of sentences.

This paper presents an approach to *dependency parsing* which combines *chunking* ([1]) and *syntactic tagging* by means of *constraint grammar* ([2]). Chunks are interpreted here as local dependency structures within syntactic tagging; this approach is related to [3]. The advantages of these two linguistic theories are linked: robustness is achieved by local structures and underspecified dependencies and disambiguation is done by a greedy strategy and by a pattern preference strategy.

Previous work which implements chunking (e.g. [4]) and syntactic tagging (e.g. [5]) with finite state machines leads to some restrictions and problems: chunking is implemented with the *left-to-right, longest-match replacement operator* [5] with which chunks are marked by brackets and are disambiguated by a left-to-right, longest-match strategy if there are various chunking possibilities. But the operator restricts the analysis to unambiguous input; in case of ambiguous input the longest-match is calculated for each input string independently. Hence, to achieve unambiguous chunking an unambiguous input has to be used.[1]

[1] Additionally, sometimes the left-to-right constraint causes unexpected analyses.

Furthermore, the operator leads to large finite state machines which require the limitation of patterns (see [6]). Syntactic tagging is implemented as *finite-state intersection grammar* by means of the *restriction operator* [7], which implements constraints as elimination rules. With these rules readings can be eliminated in an ambiguous input. But this implementation lacks the possibility of violating or preferring (weighting) constraints. Thus the main problem of implementing chunking and syntactic tagging is the way of doing disambiguation.

In the new approach proposed in this paper disambiguation is done by *linguistic criteria*. These criteria are formalized by a *semiring* over weights of a *weighted finite state transducer* (WFST). Scored dependency structures are generated over an input by a cascade of WFSTs (i.e. WFSTs are applied sequentially). Then the structures can be ordered on the basis of their weights by means of linguistic criteria to extract the most likely syntactic readings. This approach solves the problems mentioned above.

The paper is organized as follows: Section 2 gives basic definitions and notations. Section 3 is a brief reminder of the used linguistic theories. In sections 4, 5 and 6 the implementation of chunking, syntactic tagging and their combination with linguistic criteria is presented. Finally, an overview of the system *SynCoP* which implements the approach is given in section 7.

2 Definitions and Notations

In our approach syntactic analyses are generated and scored over an input by a WFST representing a constraint grammar such that syntactic readings can be judged by linguistic criteria. A weighted finite state transducer $T = (\Sigma, \Delta, Q, q_0, F, E, \lambda, \rho)$ over a semiring S is an 8-tuple such that Σ is the finite input alphabet, Δ is the finite output alphabet, Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final states, $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times S \times Q$ is the set of transitions, λ is the initial weight and $\rho : F \mapsto S$ is the final weight function mapping final states to elements in S.

In this paper individual linguistic criteria are formalized by the notion of a semiring. Let $S \neq \emptyset$ be a set and $\oplus$ (called addition) and $\otimes$ (called multiplication) binary operations on S, then $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is called a semiring if $(S, \oplus, \bar{0})$ is a commutative monoid, $(S, \otimes, \bar{1})$ is a monoid and $\otimes$ distributes over $\oplus$. Linguistic criteria are represented by this structure. To judge syntactic readings via addition an additive idempotent semiring has to be used to create a partial order over S. Thus a partial order is defined by $(a \leq_S b) \Leftrightarrow (a \oplus b = a)$. Here $a \leq_S b$ means that a is "better" than b in respect to linguistic criteria.

It will be necessary to judge analyses by more than one linguistic criterion; the criteria are ranked by preference. To model this we define the *composition* of additive idempotent semirings as follows: if a linguistic preference $(S_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1) \succ (S_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2) \succ ... \succ (S_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n)$ is given and if for each semiring a partial order is defined by $\oplus$, then:[2]

[2] The definition is similarly to the *cross product* of semirings.

$$(S, \oplus, \otimes, \bar{0}, \bar{1}) =$$

$$(S_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1) \circ (S_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2) \circ ... \circ (S_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n) =$$

$$(S_1 \times S_2 \times ... \times S_n, \oplus, \otimes, (\bar{0}_1, \bar{0}_2, ..., \bar{0}_n), (\bar{1}_1, \bar{1}_2, ..., \bar{1}_n)) \quad (1)$$

The composition $\circ$ is the vectorization of the individual domains. The operation $\otimes$ of the semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is defined as a vectorization too, if $(a_1, a_2, ..., a_n) \in S$ and $(b_1, b_2, ..., b_n) \in S$ are given:

$$(a_1, a_2, ..., a_n) \otimes (b_1, b_2, ..., b_n) = (a_1 \otimes_1 b_1, a_2 \otimes_2 b_2, ..., a_n \otimes_n b_n) \quad (2)$$

And finally, the operation $\oplus$ which compares syntactic readings is defined as follows:

$$(a_1, a_2, ..., a_n) \oplus (b_1, b_2, ..., b_n) =$$

$$\begin{cases} (a_1, a_2, ..., a_n) & \text{if } (a_1, a_2, ..., a_n) = (b_1, b_2, ..., b_n) \\ (a_1, a_2, ..., a_n) & \text{if } a_1 = b_1 \text{ and } a_2 = b_2 \text{ and ... and } a_{k-1} = b_{k-1} \\ & \text{and } a_k \oplus_k b_k = a_k \\ & \text{with } k \leq n \text{ and } a_k \neq b_k \\ (b_1, b_2, ..., b_n) & \text{if } a_1 = b_1 \text{ and } a_2 = b_2 \text{ and ... and } a_{k-1} = b_{k-1} \\ & \text{and } a_k \oplus_k b_k = b_k \\ & \text{with } k \leq n \text{ and } a_k \neq b_k \end{cases} \quad (3)$$

With this composition it is possible to combine ranked linguistic criteria represented by several semirings to one semiring. The resulting semiring is now additive idempotent as well and a partial order can be defined by $\oplus$.

Syntactic analyses are ordered by linguistic criteria according to their degree of grammatical acceptance. This is done by a simple comparison: a reading is better than another or not. That allows structural preferences and gradual grammaticality. Extracting the most likely readings in a WFST T is a classical best-path problem. Weights along a path of T are combined by multiplication and create costs. If several paths are in T their weight equals the addition of weights of the different paths, that means the "best" cost (see [8]). The most likely syntactic analyses are simply represented by paths which cause these "best" costs.

A constraint grammar R can be applied by composition in linear time according to the size of an input acceptor S. Here, the composition can be computed in time $O(|R||S|)$ where $|R|$ and $|S|$ denote the number of states of R and S respectively (cf. [9]). The application of the constraint grammar results in a WFST which is acyclic. The best-path search can be calculated in $O(|Q| + |E|)$ in the acyclic case if Q is the set of states and E is the set of transitions (see [8]). Thus, the worst case time and space complexity of the application of a constraint grammar is $O(|S|)$ (cf. [6]).

In the following the ENGCG tagset and the regular expression notation of [5] (slightly extended) are used.[3]

[3] See appendix for regular expression notation details. Here the precedence is defined top down. The distinction between the language A and the identity relation which maps every string of A into itself is ignored.

3 Linguistic Background

In this section a short summary of syntactic tagging by means of constraint grammar, chunking and the combination of these two theories is given.

A constraint grammar [2] consists of a set of *constraints*, which can be seen as rules which are applied on linear patterns. The analysis starts from a large number of alternative analyses (syntactic and morphological) that are reduced by the application of constraints. *Syntactic tags* are used here to mark dependency relations where every tag represents a special dependency relation within a clause. In the following example an analysis of the sentence *Bill saw the little dog* is given:[4]

Bill	N NOM SG	@SUBJ
saw	V PAST	@+FMAINV
the	DET	@DN>
little	A	@AN>
dog	N NOM SG	@OBJ

(4)

The left column of the example above corresponds to the input sentence, the middle column to the morphological analyses and the right column shows the syntactic functions.

Chunks [1] are local constituent structures which are built by two principles: *chunk connectedness* and *chunk inclusiveness*. Chunk connectedness means that *functional elements* (e.g. a function word or an empty word) have to be grouped with their selected *thematic elements* (e.g. a content word or other chunks) forming a chunk. In addition to that, chunk inclusiveness claims that every word has to belong to a chunk with the exception of a distinguished subset of function words, which can't be grouped to their selected thematic elements because of intervening chunks. This function words are called *orphaned words*. The following example shows an analysis by chunks of the sentence used above:[5]

$$[\emptyset_{Det}\ \text{Bill}][\emptyset_{Comp}\ \text{saw}][\text{the little dog}][\text{in}[\text{the park}]] \tag{5}$$

In our approach, chunks are integrated in the syntactic tagging formalism. To infer a dependency structure from a constituent structure, a transformation rule can be applied: a head α governs his dependent β, iff α is the syntactic head of the constituent and β is its complement. To apply this transformation rule the syntactic head within a chunk has to be marked by a tag. Furthermore chunks by themselves have to be marked by a syntactic tag to be integrated into the syntax of the syntactic tagging. The following example shows this integration:[6]

$[\emptyset_{Det}$ Bill@HEAD]	N NOM SG	@SUBJ
$[\emptyset_{Comp}$ saw@HEAD]	V PAST	@+FMAINV
[the little dog@HEAD]	N NOM SG	@OBJ

(6)

[4] syntactic functions: @+FMAINV = finite main verb, @SUBJ = subject, @OBJ = object, @DN> = determiner, @AN> = premodifying adjective

[5] $\emptyset_{Det}$ and $\emptyset_{Comp}$ are empty functional elements.

[6] The heads in chunks are marked by @HEAD; the morphological analyses and syntactic functions refer to these heads.

Additionally, orphaned words can also be linked to their chunk by a special tag. In some languages, for example German, many orphaned words occur. One can question if chunk inclusiveness holds in all languages. With linking this problem can be overcome (see [10]).

4 Syntactic Tagging with Linguistic Criteria

Our interpretation of constraint grammar is related to [11]. Instead of implementing constraints as elimination rules constraints are regarded as score rules. But in addition we use scores to punish and to support patterns.

The *constraint optimization criterion* which represents the constraint grammar principle is formulated as follows: the patterns which receive the best scores by the constraints of a constraint grammar are in this sense the most grammatical. The criterion is formalized by the max semiring $(\mathbb{R} \cup \{-\infty\}, max, +, -\infty, 0)$; here the score is coded by numbers: positive numbers support patterns gradually and negative numbers punish patterns gradually.

To implement a constraint which supports patterns with respect to their context the *optional score operator* is defined as follows (it is sufficient to assign the weights optionally):

$$\begin{array}{ll} \texttt{A}(\Rightarrow_{\omega})\ \texttt{B}\ \texttt{\_}\ \texttt{C} & =_{def}\ \texttt{[?*}\ \texttt{B}\ \texttt{A}\ \texttt{C}\ \langle\omega\rangle\texttt{]*?*} \\ \texttt{A}(\Rightarrow_{\omega})\ \texttt{B}\ \texttt{\_} & =_{def}\ \texttt{[?*}\ \texttt{B}\ \texttt{A}\ \langle\omega\rangle\texttt{]*?*} \\ \texttt{A}(\Rightarrow_{\omega})\ \texttt{\_}\ \texttt{C} & =_{def}\ \texttt{[?*}\ \texttt{A}\ \texttt{C}\ \langle\omega\rangle\texttt{]*?*} \end{array} \tag{7}$$

In contrast to the restriction operator [7], the optional score operator does not need the complementation of its operands. Thus, the resulting WFSTs stay small and transductions can be performed.[7]

The weights have to be assigned obligatorily (concerning the domain) to implement a constraint which punishes patterns with respect to their context. Otherwise the constraints have no effect. Thus, the *mandatory score operator* is defined as follows:[8]

$$\begin{array}{ll} \texttt{A}\Rightarrow_{\omega}\ \texttt{B}\ \texttt{\_}\ \texttt{C} & =_{def}\ \texttt{[}\sim\$Dom(\texttt{B}\ \texttt{A}\ \texttt{C})\ \texttt{B}\ \texttt{A}\ \texttt{C}\ \langle\omega\rangle\texttt{]*}\ \sim\$Dom(\texttt{B}\ \texttt{A}\ \texttt{C}) \\ \texttt{A}\Rightarrow_{\omega}\ \texttt{B}\ \texttt{\_} & =_{def}\ \texttt{[}\sim\$Dom(\texttt{B}\ \texttt{A})\ \texttt{B}\ \texttt{A}\ \langle\omega\rangle\texttt{]*}\ \sim\$Dom(\texttt{B}\ \texttt{A}) \\ \texttt{A}\Rightarrow_{\omega}\ \texttt{\_}\ \texttt{C} & =_{def}\ \texttt{[}\sim\$Dom(\texttt{A}\ \texttt{C})\ \texttt{A}\ \texttt{C}\ \langle\omega\rangle\texttt{]*}\ \sim\$Dom(\texttt{A}\ \texttt{C}) \end{array} \tag{8}$$

With this operator transductions can be performed, too. But the operator is defined by complementation, hence the operator causes larger WFSTs.

[7] Complementation presupposes deterministic finite state acceptors and the determinization of the complement acceptors shows an exponential behaviour concerning the number of states.

[8] The function *Dom* returns the domain of the constraint patterns. So the regular language which does not contain the constraint patterns in respect of the domain can be built by complementation.

To assign positive or negative potentials to patterns we complete the optional score operator and the mandatory score operator:

$$\begin{aligned} \texttt{A}(\Rightarrow_{\omega})\ \texttt{\_} &=_{def} \texttt{[?}^{*}\ \texttt{A}\ \langle\omega\rangle\texttt{]}^{*}\texttt{?}^{*} \\ \texttt{A}\Rightarrow_{\omega}\ \texttt{\_} &=_{def} \texttt{[}\sim\$Dom(\texttt{A})\ \texttt{A}\ \langle\omega\rangle\texttt{]}^{*}\ \sim\$Dom(\texttt{A}) \end{aligned} \tag{9}$$

In the following an example of a constraint and its application are given. Let *... a move ...* be an input fragment. We define the constraint `[@.N.]`$(\Rightarrow_{10})$`[@.DET.] _` to analyse the input fragment as follows:[9][10]

```
@ a     DET @DN>                                    (10)
@ move [[N [@SUBJ|@OBJ|@I-OBJ](⟨10⟩)]|
       [V  @+FMAINV]]
```

With the introduced weight $\langle 10\rangle$ the reading in which *move* is a verb can be suppressed. A constraint grammar is constructed by combining the initial and the context constraints by composition. The big advantage of this approach is the possibility to violate constraints; so the application of a constraint grammar to an input is never empty.

5 Chunking with Linguistic Criteria

The grouping of chunks is formalized on the POS level by patterns which include the functional element optionally and its selected thematic element and the elements which can occur between them obligatorily. These patterns are marked in the input by brackets representing the syntactic projections. This is done by the *optional insertion operator* which is defined as follows if `A` denotes the chunk pattern and `P` and `S` the brackets:

$$\texttt{A}(\rightarrow)\texttt{P...S} =_{def} \texttt{[?}^{*}\texttt{[0.x.P]A[0.x.S]]}^{*}\texttt{?}^{*} \tag{11}$$

Building chunks with this implementation is not definite because of optional chunking, optional functional elements, ambiguous possible selections and especially ambiguous input. The following example *this cell structure* which includes the POS sequence `[DET|PRON]N[N|V]` shows the ambiguous chunking, if the chunker `[[(DET)N*]|[PRON]]`$(\rightarrow)$`%[ ... %]` is applied:[11]

$$\begin{array}{c} \text{this cell structure} \\ \text{[this] [cell] [structure]} \\ \text{[this] [cell structure]} \\ \text{[this cell structure]} \\ \ldots \end{array} \tag{12}$$

[9] The macro "`.`" is defined as `~$@` and the symbol "`@`" is used to mark word borders.
[10] For the sake of readability the morphological features are left out in the example.
[11] The lexicalized multi-word unit *cell structure* is treated as a non-lexicalized multi-word unit like *president Kennedy* or *city Berlin* which can also be seen as one thematic element.

The problem of ambiguity arises, because the greedy definition of chunks by chunk connectedness and chunk inclusiveness is not implemented.

Our approach concerning longest match is based on the work of [6] and [12]. There, a longest match constraint is implemented using the weights of a WFST. In our approach, the longest match is formalized by the tropical semiring $(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$ and is implemented as follows: intra-chunk symbols receive a high negative number to prefer *exhaustive grouping* and brackets receive a low positive number to prefer *large groupings*. Following the example above, words within chunks get the weight ⟨-1⟩ and bracket pairs the weight ⟨0.1⟩; the syntactic readings are ordered by their longest match satisfaction:

$$\begin{array}{rl} \text{-2.9} & \text{[this cell structure]} \\ \text{-2.8} & \text{[this] [cell structure]} \\ \text{-2.7} & \text{[this] [cell] [structure]} \\ & \quad ... \\ 0 & \text{this cell structure} \end{array} \tag{13}$$

The analysis on the top represents the longest match (with the weight -2.9). It is possible to disambiguate chunking with the help of this greedy strategy.

Note that this approach is inconsistent in one case: Let $\mathtt{a}^*$ be the chunk pattern and let $\mathtt{a}_1\ \mathtt{a}_2\ ...\ \mathtt{a}_n$ be the input. The result of the chunking contains the weighted strings $\mathtt{[}\ ...\ \mathtt{|}\ \langle\omega_1\rangle\mathtt{\%[a}_1\mathtt{\%]\%[a}_2\mathtt{\%]}\ ...\ \mathtt{\%[a}_n\mathtt{\%]}\ \mathtt{|}\ \langle\omega_2\rangle\mathtt{\%[a}_1\mathtt{a}_2\ ...\ \mathtt{a}_{n-1}\mathtt{\%]a}_n\ \mathtt{|}\ \ldots\ \mathtt{]}$; there exists an n with $\omega_1 > \omega_2$ but there also exists another n with $\omega_1 \leq \omega_2$. It is possible that the result of the multiplication of the bracket costs contains the inverse of the cost assigned to a symbol within chunks. Hence in case of ambiguous input problems can arise.

To avoid this problem we formulate the disambiguation by chunk connectedness and chunk inclusiveness as criteria. The two criteria are ranked by preference: chunk inclusiveness $\succ$ chunk connectedness. The criteria say that words should belong to chunks primarily and words should form large chunks secondarily; that means functional elements should be grouped to their selected thematic elements. The chunk connectedness criterion and the chunk inclusiveness criterion are formalized separately by the tropical semiring $(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$. Following the approach above the criteria are implemented by assigning negative numbers to symbols within chunks in respect of chunk inclusiveness on the one hand and assigning positive numbers to brackets in respect of chunk connectedness on the other hand. The optional insertion operator is now adjusted as follows:

$$\mathtt{A}(\rightarrow)\mathtt{P...S} =_{def} \mathtt{[?^*[0.x.P][A.o.[?}\langle -1, 0\rangle\mathtt{]^*][0.x.S]}\langle 0, 1\rangle\mathtt{]^*?^*} \tag{14}$$

This implementation works on ambiguous input and disambiguates the input locally according to the linguistic criteria.[12] No complementation operation is used, hence transductions can be performed and the resulting WFSTs stay small

[12] With the given linguistic criteria the acceptance of the syntactic readings [*this*] [*cell structure*] and [*this cell*] [*structure*] equals. Such ambiguities should not be solved within this approach; the remaining ambiguities have no consequences. But in [10]

in size. Furthermore, the chunking can be affected by a later step because all chunking possibilities are enhanced.

To build a chunker with several planes of projection, several chunk types are built by the optional insertion operator and combined by composition. If several chunk types should compete within one level they are combined by union. Then the Kleene star is applied.

6 Combining Syntactic Tagging with Chunking

To integrate chunking within syntactic tagging, the syntactic heads in chunks are marked and the chunk itself is labeled with potential syntactic functions. This is achieved by the optional insertion operator. The following example shows this:

$$\texttt{[DET N[0.x.@HEAD]]}(\rightarrow)\texttt{\%[... \%][@SUBJ|@OBJ|@IOBJ]} \tag{15}$$

Constraints can refer to chunks by their brackets and functions.

Via constraints it should be possible to restrict chunking to resolve *garden-path effects* which result from the greedy implementation. The problem is comparable to the *late closure* parsing principle [13] and is shown by the following example (the example is taken from [14]):

1. *[the emergency crews] really hate is [family violence]
   → garden-path effect
2. [the emergency] [crews] really hate is [family violence]
   → resolved

In order to do this, the constraint optimization criterion has to be ranked over the criteria concerning chunking. But the chunking should not be restricted by constraints in every case. Hence we distinguish between two kinds of constraint optimization criteria: strong and weak. The following ranking is used: strong constraint optimization $\succ$ chunk inclusiveness $\succ$ chunk connectedness $\succ$ weak constraint optimization. Now an analogical semiring can be built by composition.

Consequently, a constraint grammar contains both chunking and constraints. Here it is possible to assign costs concerning the weak constraint optimization criterion to chunk internal patterns. The advantage of combining chunking and constraints is obvious: robustness is reached by underspecified and local structures and the input is disambiguated by chunking and by constraints. Thus very robust and rich parsing is possible while the WFSTs of a constraint grammar stay small in size.

7 The System

SynCoP (Syntactic Constraint Parser) depends on the *Potsdam Finite State Library (FSM<2.0>)* [15], which makes it possible to change semirings via a

a criterion is presented which implements a *left-to-right preference strategy* which eliminates these ambiguities.

template. The system consists of a *grammar compiler* and a *grammar applier*. The grammar compiler takes an XML specification which contains definitions concerning constraints and chunking and builds a constraint grammar which consists of a cascade of WFSTs. This constraint grammar is used by the grammar applier to parse ambiguous input which is the result of the *TAGH morphology* [16].[13]

So far our goal is not to eliminate all morphological ambiguities but to extract dependency structures. Material which is not integrated into the dependency structures is not disambiguated. Our current hand-written grammar for German newspaper texts is compiled to a cascade of five WFSTs:

(1) morphology interface (48 states, 12415 transitions)
(2) chunking (70501 states, 246535 transitions)
(3) local dependency structures (6573 states, 356750 transitions)
(4) embedded clauses (1609 states, 188570 transitions)
(5) main clauses (2515 states, 346788 transitions)

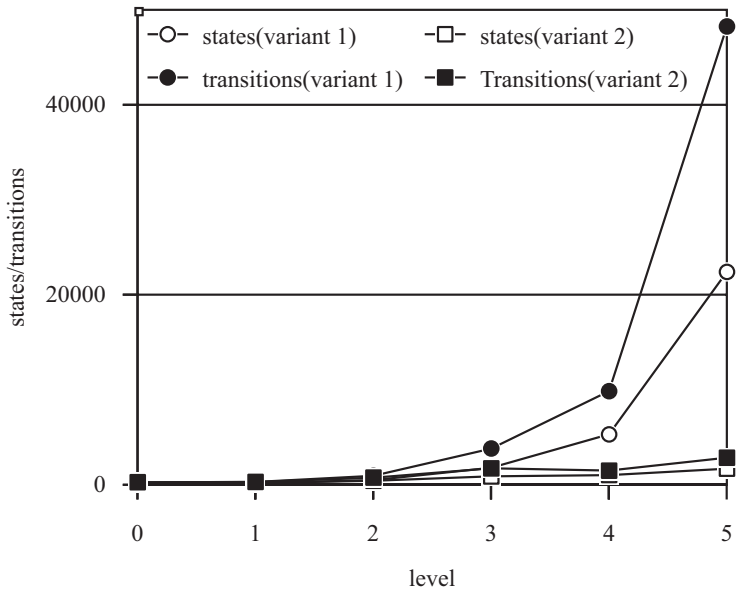


Fig. 1. Number of states and transitions in respect of the level of analysis

Level (1) does simple transductions mapping different POS tagsets and features and introduces possible main-clause and sub-clause borders and potential syntactic functions. Level (2) performs the chunking (NP-chunks, PP-chunks, AP-chunks) and chunk labeling with potential syntactic functions. Level (3) links

[13] The TAGH morphology contains information about lexicalized multi-word units. Hence, lexicalized and non-lexicalized multi-word units are discriminated.

orphaned words and disambiguates modifier functions or conjunctions which are related to chunks. The main reason for the large WFSTs concerning level (2) and (3) is the modeling of agreement in respect of case, number and gender. Finally level (4) disambiguates the head functions for embedded clauses and level (5) does the same for main clauses, while main-clauses and sub-clauses are detected.

During the analysis by our constraint grammar the resulting WFST grows level by level before the most likely syntactic readings are extracted by a best-path search. This is shown by the analysis of the German sentence *der Mann, der das Auto auf dem Parkplatz lieben wird, weint. (the man, which is going to love the car at the parking lot, cries.)*.[14] [15] The steps of this analysis concerning the number of states and transitions are shown in figure 1. Here, the number of states and transitions concerning the "normal" application of the constraint grammar is shown by variant 1. The input WFST has 158 states and 249 transitions. However the final result has 21406 states and 48135 transitions. The reason for that rapid growth is redundancy: ambiguous structures are looped through the levels instead of being resolved as early as possible.

To diminish this rapid growth we use a local disambiguation strategy. Material within chunks is never affected by constraints of higher levels; hence disambiguation can be performed within chunks without eliminating readings which are necessary for later steps. The same holds for sub-clauses. To do so, we implement a best-path search which refers to chunk and sub-clause brackets: after level (2) we disambiguate within chunks and after level (4) we disambiguate within sub-clauses. The amount of states and transitions concerning this strategy is shown in the figure above by variant 2. The resulting WFST has finally 1683 states and 2841 transitions; in level 4 the amount of transitions actually decreases.

8 Conclusion and Future Work

A new approach to robust dependency parsing which brings together syntactic tagging and chunking has been presented. Their combination is implemented by WFSTs over a semiring which represents several linguistic criteria. With these linguistic criteria disambiguation is done by a simple comparison concerning the degree of grammatical acceptance of syntactic analyses; this allows structural preferences and gradual grammaticality. It is possible to extend these linguistic criteria.

The System SynCoP is currently used for the construction of an engine analogical to the *Word Sketch Engine* [17].[16] Our word sketches show a promising

[14] PP-attachment is not covered by our constraint grammar yet.

[15] That sentence is analysed as follows: [$_{main\_cl}$[$_{np}$der Mann@HEAD]@SUBJ, [$_{sub_cl}$der@SUBJ [$_{np}$das Auto@HEAD]@OBJ [$_{pp}$auf@HEAD [$_{np}$dem Parkplatz@HEAD]] lieben@-FMAINV wird@FAUXV], weint@+FMAINV .]

[16] A *word sketch* is a summary which is deduced from corpora showing grammatical and collocational behaviour of a word.

quality of annotation. An exhausting evaluation of our constraint grammar has not been done yet; this will be of interest in future work.

9 Appendix: Notations

(A)	option (union of A with epsilon)
∼A	complement
$A	contains (all strings containing at least one A)
A^*	Kleene star
A^+	Kleene plus
A B	concatenation
A | B	union
A .x. B	crossproduct
A .o. B	composition
Dom(A)	the domain of a rational transduction
[and]	square brackets which group expressions
?	sigma
$?^*$	sigma star
0	epsilon
%	escape character
$\langle\omega\rangle$	weights

References

1. Abney, S.: Syntactic affixation and performance structures. In Bouchard, D., Leffel, K., eds.: Views on Phrase Structure. Kluwer (1990)
2. Karlsson, F.: Constraint grammar as a framework for parsing running text. In: Proceedings of the 13th International Conference on Computational Linguistics (COLING-90). Volume 3. (1990) 168–173
3. Aït-Mokhtar, S., Chanod, J.P.: Incremental finite state parsing. In: Proceedings of the 5th. International Conference on Applied Natural Language Processing (ANLP'97). (1997) 72–79
4. Grefenstette, G.: Light parsing as finite state filtering. In Kornai, A., ed.: Extended Finite-State Models of Language. Cambridge University Press (1999) 86–94
5. Karttunen, L.: Directed replacement. In Joshi, A., Palmer, M., eds.: Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics. (1996) 108–115
6. Hanneforth, T.: Longest-match pattern matching with weighted finite state automata. In: Proceedings of Finite-State Methods and Natural Language Processing (FSMNLP 05). (2005) 79–85
7. Koskenniemi, K.: Finite-state parsing and disambiguation. In: Proceedings of the the 13th International Conference on Computational Linguistics (COLING 90). Volume 2. (1990) 229–232
8. Mohri, M.: Semiring frameworks and algorithms for shortest-distance problems. Languages and Combinatorics **7**(3) (2002) 321–350
9. Tapanainen, P.: Applying a finite-state intersection grammar. In Roche, E., Schabes, Y., eds.: Finite-State language processing. MIT Press (1997) 311–327

10. Didakowski, J.: Robustes Parsing und Disambiguierung mit gewichteten Transduktoren. Volume 23. Linguistics in Potsdam (2005)
11. Tzoukerman, E., Radev, D.R.: Use of weighted finite state transducers in part of speech tagging. In Kornai, A., ed.: Extended Finite-State Models of Language. Cambridge University Press (1999) 193–207
12. Nasr, A., Volanschi, A.: Integrating a pos tagger and a chunker implemented as weighted finite state machines. In: Proceedings of Finite-State Methods and Natural Language Processing (FSMNLP 05). (2005) 167–178
13. Frazier, L., Clifton, Jr., C.: Construal. MIT Press (1996)
14. Abney, S.: Chunks and dependencies: Bringing processing evidence to bear on syntax. In Cole, J., Green, G., Morgan, J., eds.: Computational Linguistics and the Foundations of Linguistic Theory. CSLI (1995) 145–164
15. Hanneforth, T.: FSM<2.0> – C++ library for manipulating (weighted) finite automata. http://www.fsmlib.org (2005)
16. Geyken, A., Hanneforth, T.: Tagh: a complete morphology for german based on weighted finite state automaton. In: Proceedings of Finite-State Methods and Natural Language Processing (FSMNLP 05). (2005) 55–66
17. Kilgarriff, A., Rychly, P., Smrz, P., Tugwell, D.: The sketch engine. In: Proceedings of the Eleventh EURALEX International Congress. (2004) 105–116

Syntactic Error Detection and Correction in Date Expressions using Finite-State Transducers

Arantza Díaz de Ilarraza, Koldo Gojenola, Maite Oronoz, Maialen Otaegi, and Iñaki Alegria

Department of Computer Languages and Systems
University of the Basque Country
P.O. box 649, E-20080 Donostia

Abstract. This paper presents a system for the detection and correction of syntactic errors. It combines a robust morphosyntactic analyser and two groups of finite-state transducers specified using the Xerox Finite State Tool (XFST). One of the groups is used for the description of syntactic error patterns while the second one is used for the correction of the detected errors. The system has been tested on a corpus of real texts, containing both correct and incorrect sentences, with good results.

1 Introduction

In this work we present a research carried out to detect and correct syntactic errors in date expressions using finite-state transducers (FSTs). Finite-state constraints, encoded in the form of automata and transducers, have been applied to the linguistic analysis. We have used XFST for the definition of complex linguistic patterns over morphosyntactic information.

We chose to deal with date expressions due to the fact that they contain morphologically and syntactically rich enough phenomena where several types of errors can be found. They can be considered representative of the errors that are detectable by examining local syntactic contexts. Besides, and based on copy-editors' and language teachers' opinion, date expressions in Basque are one of the most frequent source of errors in both, language learners and native speakers.

Basque is an agglutinative language, and as a consequence, most of the elements appearing in date expressions (year numbers, months and days) must inflect, i.e. the corresponding article and case morphemes must be attached to them. Moreover, each different date format requires that the elements involved appear in fixed combinations of, for example, cases (see table 1), so different types of agreement are needed. These require a previous linguistic analysis before applying the FSTs for detection and correction.

Finite-state techniques have been used to create most of the NLP tools for linguistic analysis for Basque [1]. Following a previous experience in the construction of a robust spelling checker based on FSTs, XUXEN[1] [2], we have faced the task of syntactic error detection and correction in the same way.

[1] `http://ixa.si.ehu.es`

The remainder of this paper is organised as follows. Section 2 reviews several related works. After commenting on the linguistic resources we have used in section 3, we give a general overview of the system in section 4. Section 5 describes the error detection process, while section 6 presents the correction procedure. Then, we evaluate the system in section 7, to conclude in section 8.

2 Related work

The problem of syntactic error detection and correction has been addressed since the early years of natural language processing. For the treatment of the significant amount of errors (typographic, phonetic, cognitive and grammatical) that result in valid words ([3]; [4]) different techniques have been proposed:

- Grammar-based techniques. These systems use the results of a parser as input. Techniques that use chart-based methods [5] or the relaxation of syntactic constraints [6] could be categorised into this group. In general, these methods share the problem of incomplete coverage of the underlying grammars. Manually written grammars are often unable to analyse the full range of sentences in running text. Moreover, when dealing with ill-formed sentences, the systems should accept not only correct sentences, but also the much wider spectrum of incorrect ones. On the other hand, statistical parsers induced from treebanks are able to analyse any sentence, but they can not easily distinguish correct sentences from incorrect ones.
- Error patterns ([7]; [8]; [9]), which are either hand-coded rules or are automatically learned using statistical techniques. Most of these approaches are implemented using finite-state techniques, for example the Constraint Grammar (CG) formalism [10] is used in ([11]; [12]; [13]) for error detection in Swedish and Catalan, or the Xerox Finite State Tool (XFST)[14] for finding grammar errors in Swedish texts written by children [15].

Kukich [7] surveys the state of the art in syntactic error detection. She estimates that between 25% and over 50% of the total errors are in fact are valid words. On the other hand, [16] made a manual study concluding that 55% of the errors are detectable by an examination of the local syntactic context, 18% are due to global syntactic errors (involving long-distance syntactic dependencies, which need a full parse of the sentence), and 27% are semantic errors.

Errors in date expressions can be deemed as a representative of local syntactic errors. A work similar to the one presented here is that of Karttunen [17], who describes a system that mapped numbers to numerals in Finnish. This language has in common to Basque that the created linguistic structures are inflected, and some of their components must agree in case. That makes the transduction process of these languages more complex than in languages like English, with a simpler morphology.

Regarding the treatment of Basque date expressions, [18] presented a system that detected some types of errors using an unification based partial parser. This work extends that system with a more comprehensive set of error types and also including the task of error correction.

Error type	Example
0. If the place name is inflected in inessive case (*Donostian*), the day number must be inflected in inessive case. If the place name is inflected in absolutive case (*Donostia*), the day number must be inflected in absolutive case.	Donostia**[n]**, 2007ko maiatzaren 27a**[]** *27th May, 2007* Donostia, 2007ko maiatzaren 27a**[n]**
1. The year number cannot be inflected using a hyphen	Donostian, 1995**[-]**eko maiatzaren 14an
2. The month (*maiatza*) must appear in lowercase	1999ko **[M]**aiatzaren 2an
3. The optional place name preceding dates (*Frantzia*) must be followed by a comma	Frantzia 1997ko maiatzaren 8an
4. The day number after a month in genitive case (*maiatzaren*) must have a case mark	Donostian, 1995eko maiatzaren 22**[]**
5. The day number after a month in ergative case (*maiatzak*) cannot have a case mark	1998.eko maiatzak 14**[ean]** argitaratua
6. The month (*maiatza*) must be inflected in genitive or absolutive case	Donostian, 1995eko Maiatza**[ren]** 14an
7. The dot that makes a number ordinal (*1995.eko*) cannot appear after the year number except when the word *urte* ('*year*') follows it	Donostian 1997**[.]**eko Maiatzan 28an
8. Numbers 11 and 31 can not take the absolutive singular.	1997-ko maiatzaren 31**[a]**

Table 1. Most frequent error types in dates (errors marked in boldface).

3 Linguistic Resources

For the analysis of the input text, we use part of the Basque shallow syntactic analyser [1], mainly based on finite-state technology [19]. Although information at chunk or syntactic levels could be used for the treatment of other error phenomena, morphosyntactic information is enough for the recognition of errors in date expressions.

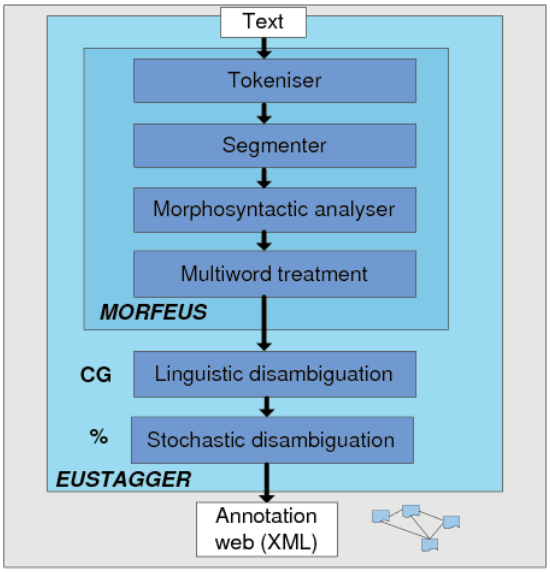


Fig. 1. Morphosyntactic analysis and disambiguation.

Figure 1 shows the morphosyntactic analyser and the modules for disambiguation. The process starts with the outcome of the morphosyntactic analyser (MORFEUS), which was created following the two-level morphology [20], and deals with all the lexical units of a text, both simple words and multi-word units. The tagger/lemmatiser EUSTAGGER not only obtains the lemma and category of each form but also performs disambiguation using for this task information about part of speech, fine grained part of speech or case. The disambiguation process is carried out by means of linguistic rules using CG and stochastic rules based on Hidden Markov Models [21], which reduces the high word-level ambiguity to a limited amount of remaining interpretations.

All the information in the analysis chain is exchanged by means of standardised XML files [22] and a class library for the management of all the linguistic information. The full system provides a robust basis, essential not only for any treatment based on corpora but also for error detection.

4 General Overview of the System

The process for error detection and correction starts after analysing the input text. The system (see figure 2) is composed of two groups of FSTs, one for error detection (see section 5) and the other one for the generation of correct dates (see section 6). Two filters prepare the input for each of these FST groups.

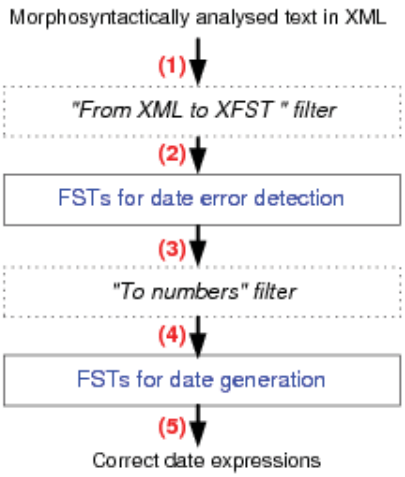


Fig. 2. General architecture of the system.

Take, for example, the date expression *"1995eko maiatzaren 15"* (15th of May, 1995). It is incorrectly written because in Basque the day number after a month in genitive case must take a case mark. Given this text as input, the date expression will go through the following modules:

1. *"From* XML *to* XFST*" filter.* In a first step, the preprocessing filter changes the morphosyntactic information in XML to a more suitable format for the FSTs. Figure 3 shows the feature structures that gather the lemma and morphosyntactic information about the incorrect date example, including POS, FPOS (fine grained part of speech), CASE, NUM and MUG (definite/indefinite article). Figure 4 represents the corresponding simplified format.
2. FST*s for date error detection.* For error detection in date expressions, we have sequentially applied nine finite-state transducers, one for each kind of error defined (see table 1), creating a cascade of FSTs. In the output of each of the FSTs, the incorrect linguistic structures are surrounded by tags describing

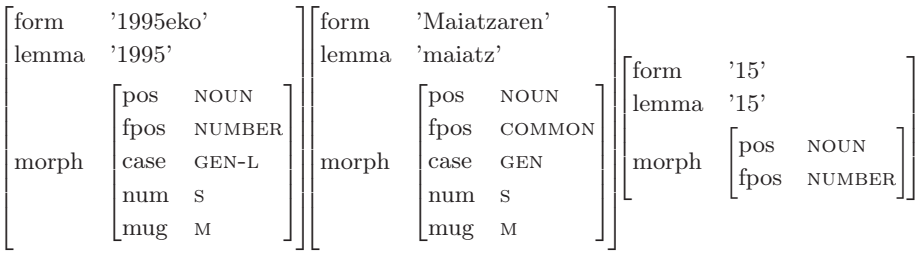


Fig. 3. Feature structures representing a date expression (see label 1 in figure 2)

each type of error. Figure 4 shows how the incorrect structure is surrounded by two error tags (BEGINERRORTYPE4 and ENDERRORTYPE4)[2] .

{ **WS** { **AS** +form +1995eko +lemma +1995 +morph +POS +NOUN +FPOS +NUMB +CASE +GEN-L +NUMBER +S +MUG +M **AE** } **WE** }
BEGINERRORTYPE4
{ **WS** { **AS** +form +Maiatzaren +lemma +maiatz +morph +POS +NOUN +FPOS +COMMON +CASE +GEN +NUM +S +MUG +M **AE** } **WE** }
{ **WS** { **AS** +form +15 +lemma +15 +morph +POS +NOUN +FPOS +NUMBER **AE** } **WE** }
ENDERRORTYPE4

Fig. 4. Output of the error detection grammar (labeled 3 in figure 2).

3. *"To numbers" filter*. Once the errors in date expressions are tagged (it is frequent to find more than one error in each date expression), the correction process starts. The FSTs used for error correction were not created specifically for this purpose but for helping Basque language learners to write date expressions[3]. The group of FST*s for date generation* obtain the corresponding text equivalences from numbers representing date expressions. The *"To numbers"* filter obtains a numbered expression with the format *"year/month/day"*[4] (see figure 5) for each date expression tagged with an error.
4. FST*s for date generation*. As we have previously mentioned, the correction module uses FSTs that change numbers representing date expressions to their

[2] The following tags are added to the morphosyntactic information to facilitate the regular expression definition in the XFST grammar: WS (word starts), WE (word ends), AS (analysis starts) and AE (analysis ends).
[3] http://kantauri.eleka.net/neh and http://sisx04.si.ehu.es/~ iniebla001/idazlagun/
[4] In Basque numerically written dates follow the format year/month/day.

Donostia *LocPn* 1995/05/15

Fig. 5. Result after the application of the "To numbers" filter (labeled 4 in figure 2).

corresponding full-text equivalences. Figure 6 shows two correction candidates created for correcting the error in the example.

Donostia, 1995eko maiatzaren 15a
Donostian, 1995eko maiatzaren 15ean

Fig. 6. Corrected date expressions (labeled 5 in figure 2).

5 Error Detection

Inflection in date expressions is a common source of errors, not detectable by a spelling checker, as each isolated word-form is correct. Figure 7 shows one of the formats of a valid date expression:

Durangon,	*1999ko*	*martxoaren*	*7an*
Durango, inessive, sing	1999, genitive	martxoa, genitive, sing	7, inessive, sing
In Durango,	1999,	March	the 7th

Fig. 7. Format of a valid date expression.

After examining different instances of errors, we chose the nine most frequent error types (see table 1). Some of these errors belong to idiosyncratic facts of date expressions (errors 0, 3, 4, 5 and 6), while four of them must be considered linguistically incorrect facts that can be reused in other general contexts (errors 1, 2, 7 and 8). A group of error detection patterns has been defined in XFST for each of the error types, and after compiling them, a cascade of FSTs is applied to the input text.

We adopted a kind of "gradual relaxation" approach, considering that several mistakes could co-occur, as quite often two or three errors might appear in the same expression. We had to design error patterns bearing in mind not only the correct expression, but also its erroneous versions. This relaxation on what could be considered a correct date had the risk of increasing the number of false alarms.

The error pattern for the fourth kind of error (the day number after a month in genitive case must have a case mark) is defined in two steps (see figure 8). First, the syntactic pattern of the error is defined (a correct month or a month

```
1. define Month_Gen ...
2. define Incorrect_Month_Gen_in_Upper ...
3. define Correct_Year ...
4. define Incorrect_Year_with_Hyphen ...
5. define Year  [ Correct_Year |  Incorrect_Year_with_Hyphen ]
6. define Error_Type_4
      [Month_Gen |  Incorrect_Month_Gen_in_Upper ] Not_Inflec_Numb;
7. define Mark_Error_Type_4 [ Error_Type_4 ]
      @ -> BEGINERRORTYPE4 ... ENDERRORTYPE4 ||  Year _   ;
```

Fig. 8. Regular expressions for an error pattern.

incorrectly written in uppercase followed by a non inflected number, see definitions 1 through 6), and named `Error_Type_4`. Second, a longest-match left-to-right replace operator (`@ - >`) is used (`Mark_Error_Type_4`) to surround the incorrect pattern (represented by ...) by two error tags (BEGINERRORTYPE4 and ENDERRORTYPE4). To further restrict the application of the rule, left and right contexts for the error can be defined, mostly to assure that the rule is only applied to dates, thus avoiding false alarms. In this case, a year must be found to the left of the month. The year could be a correctly written year or a mispelled one (with a hyphen). As we can see, the error-description pattern considers the possibility that previous error patterns occur.

6 Error Correction

For the correction task, we took a group of already defined XFST transducers that was created to map numbers representing date, time and number expressions to text [23], and adapted a subset of them in order to correct date expressions.

According to The Royal Academy of the Basque Language[5], the most appropriate ways to express a date are those in which the locative (place name) and the declension of the day agree in absolutive or inessive cases (in figure 6 the first date expression agrees in absolutive case and the second one in inessive case), so, we create date expressions in this format.

A FST has been used for each of the cases. These transducers, nevertheless, do not create the word indicating location and the comma after it. A FST for morphological generation created using *lexc* [24] is used to generate the locative in inessive case. The comma is generated after checking that a proper name indicating a locative (*LocPn* in figure 5) precedes the date.

The process of creating a full-text date is simple. Let us explain the rules for specifying the FST that generates dates in inessive case (see figure 9). The input is divided into three groups separated by slashes: year, month and day.

[5] http://www.euskaltzaindia.net, 37th rule

When a year is found, a genitive locative[6]) morpheme (*"-ko"*) is added to the year number (`Translate_Year`, rule number 1). Months are mapped from numbers to text by means of replacement operators that are restricted to the date context (`Translate_Month`, rule number 3). Finally, the inessive singular morpheme (*"-an"*) is added to the day (`Translate_Day`, rule number 4). There are several exceptions to these mappings: when the year or day number finishes in a consonant, an epenthetic *"-e"* is added to the genitive locative case in the year (*"-e"* + *"-ko"* = *"-eko"*), and to the inessive case (*"-e"* + *"an"* = *"-ean"*) in the day (`Add_E_Day`, rule number 6).

```
1. define Translate_Year [  "/" -> "ko" || _ Number Number "/" ];
2. define Translate_Month05 [ "0" 5 "/ " -> " maiatzaren " || "ko" _ ];
3. define Translate_Month [ Translate_Month01 .o. Translate_Month02
    .o. ... .o. Translate_Month12 ];
4. define Translate_Day [ [ .. ] -> "a" "n" || Number _ .#. ];
5. define Translate [ Translate_Year .o. Translate_Month .o.
     Translate_Day ];
6. define Add_E_Day [ "a" "n" -> "e" ... ||
     [ [ "0" | 2 | 4 | 6 | 8 ] 1 | [ 1 | 3 | 5 | 7 | 9 ] "0" | 5 ] _ ];
         ...
n-1. define Clean [ Add_E_Day .o. Add_E_Year ];
n. define Translate_Clean [ Translate .o. Clean ];
```

Fig. 9. Regular expressions for date generation.

This method, based on the generation of correct date expressions, guarantees the correction of all the errors in the expression even if not all of them were detected. For example, if only 2 errors out of 3 are detected, all of them are properly corrected.

7 Evaluation

The evaluation corpus (development + test) is composed of 267 essays written by students (with a high proportion of errors) and texts from newspapers and magazines, more than 500,000 words altogether. From them we chose 658 sentences, including correct dates, incorrect dates, and also structures similar to dates. It was relatively easy to obtain test data compared to other kinds of errors. Although the data must be obtained mostly manually, date expressions contain several cues (month names, year numbers) that help in the process of finding semiautomatically test sentences.

[6] The genitive locative case "-ko" ("of") is attached to phrases that denote location, or to phrases that denote a property.

All the corpus was inspected looking for false alarms (see table 2), that is, correct dates or sentences similar to dates that could be flagged as erroneous. The problem of false alarms is one of the biggest challenges we must face when dealing with unrestricted texts. As a result of the selection procedure, the proportion of errors was higher than in normal texts. Therefore, we divided our data into two groups. One of them was used for development and we left the second one for the final test. The proportion of correct dates was higher in the case of test data with respect to those in the development corpus, so that the effect of false alarms would be evaluated with more accuracy.

	Development corpus		**Test corpus**	
Number of test items	411		247	
Correct dates	51		35	
Structures "similar" to dates	263		173	
Incorrect dates	97		38	
Incorrect dates with 1 error	48	49.6 %	9	23.7 %
Incorrect dates with 2 errors	35	36.0 %	25	65.8 %
Incorrect dates with 3 errors	10	10.3 %	3	7.9 %
Incorrect dates with 4 errors	4	4.1 %	1	2.6 %

Table 2. Test data.

	Development corpus		**Test corpus**	
Number of test items	411 (97 errors)		247 (38 errors)	
Undetected date errors	4	4.1 %	3	7.9 %
Detected date errors	93	95.9 %	35	92.1 %
False alarms	2		4	

Table 3. Evaluation results.

Table 3 shows the results of the evaluation. As the development corpus could be inspected during the refinement of the patterns, the results in the second and third columns can be understood as an upper limit of the system in its current state, with 95.9% recall[7] and 97.8% precision[8] (93 detected errors/95 error proposals, that is, 2 false alarms).

The system obtains 92.1% recall over the corpus of previously unseen 247 test items. Regarding precision the system correctly detects 35 errors, giving 39 proposals (89.7%). If the false alarms are divided by the number of test items (4/247) of the test corpus, we can estimate the false alarm rate to be around

[7] recall = correctly detected errors/all errors

[8] precision = correctly detected errors/(correctly detected errors + false alarms)

1.6% over the number of dates in real texts. Table 4 examines some of the false alarms and their cause. Although the results are good, more corpus data will be needed in order to maximize precision.

The correction guarantees that all the errors in date expressions were corrected even when some of them could not be detected. That is, even when a sentence contains more than one error, once one is detected, it is transformed to the numerical format. As correct date expressions are generated from this format, all the errors are corrected.

Example	**Cause of the error**
1998ko abenduak 20. Bizkaiko → 1998ko abenduak 25. *20th December, 1998. From Bizcay 25th December, 1998*	The analyser does not detect the line end and analyses the *Bizkaiko* place name as it was immediately preceding the date expression. If it was the case, the comma is missing.
Primakovek 1998ko irailaren 11n hartu zuen ... *Primakov took it on the 11th of September 1998*	The unknown word *Primakov* is interpreted as a place name.

Table 4. False alarms.

8 Conclusions and Future Work

This work shows an application of XFST for syntactic error detection and correction in date expressions. The reported experiment is based on a corpus, and tested on real examples of both correct and incorrect sentences. This approach implies the existence of big corpora and manual annotation for most of the errors.

Two of the most successful methods for error detection, i.e., relaxation of syntactic constraints and error patterns, have been combined in our system with good results. Relaxation has not been dynamically applied at parsing time, but it has been manually coded. This implies a considerable amount of work, as we had to consider the formats for valid sentences as well as for all their incorrect variants. Regular expressions in the form of automata and transducers are suitable for the definition of complex error patterns based on linguistic units.

We are currently exploring extensions to the system to detect new kinds of errors by combining rule-based error detection and automatic acquisition of error patterns. We think that this could help to smooth the scaling-up problem associated to the increase in the number of rules, and the amount of work in the process of hand-coding them. Using either hand-coded rules or automatically learned ones, both methods have still the problem of obtaining and marking big test corpora. Some experiments with the automatic creation and tagging of errors ([25];[26]) seem to be a possible solution to this bottleneck.

We plan to extend the error detection/correction system to other qualitatively different types of errors, such as those involving agreement between the main components of the sentence, which is very rich in Basque, errors due to incorrect use of subcategorization and errors in post-positions. Errors in post-positions, determiner-noun agreement errors, ... could be treated using XFST, but a deeper study must be made if we want to deal with errors involving long-distance dependencies in the sentence (e.g. agreement between verb and subject, object or indirect object). Although the number of potential syntactic errors is huge, we think that the treatment of the most frequent kinds of error with high recall and precision can result in useful grammar-checking tools.

Acknowledgments. This research is supported by the University of the Basque Country (GIU05/52) and the Ministry of Industry of the Basque Government (ANHITZ project, IE06-185). We would like to thank Ruben Urizar for his collaboration in this work.

References

1. Aduriz, I., Díaz de Ilarraza, A.: Morphosyntactic Disambiguation and Shallow Parsing in Computational Processing of Basque. Inquiries into the lexicon-syntax relations in Basque (2003) 1–21
2. Agirre, E., Alegria, I., Arregi, X., Artola, X., Díaz de Ilarraza, A., Urkia, M., Maritxalar, M., Sarasola, K.: XUXEN: A Spelling Checker/Corrector for Basque Based on Two-Level Morphology. In: Proceedings of ANLP'92, Povo Trento (1992) 119–125
3. Weischedel, R., Sondheimer, N.: Meta-rules as a Basis for Procedding Ill-Formed Input. American Journal of Computational Linguistics **9**(3-4) (1983) 161–177
4. Heidorn, G., k. Jensen, Miller, L., Byrd, R., Chodorow, M.: The EPISTLE Text-Critiquing System. IBM Systems Journal **21**(3) (1982)
5. Min, K., Wilson, W.H.: Integrated Control of Chart Items for Error Repair. In: COLING-ACL. (1998) 862–868
6. Douglas, S., Dale, R.: Towards Robust PATR. In: COLING. (1992) 468–474
7. Kukich, K.: Techniques for Automatically Correcting Words in Text. ACM Computing Surveys **24**(4) (December 1992) 377–439
8. Golding, A.R., Schabes, Y.: Combining Trigram-Based and Feature-Based Methods for Context-Sensitive Spelling Correction. In Joshi, A., Palmer, M., eds.: Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics, San Francisco, Morgan Kaufmann Publishers (1996) 71–78
9. Mangu, L., Brill, E.: Automatic Rule Acquisition for Spelling Correction. In: Proceedings of the 14th International Conference on Machine Learning, Morgan Kaufmann (1997) 187–194
10. Karlsson, F., Voutilainen, A., Heikkila, J., Anttila, A.: Constraint Grammar: Language-independent System for Parsing Unrestricted Text. Prentice-Hall, Berlin (1995)
11. Arppe, A.: Developing a Grammar Checker for Swedish. In: Proceedings from the 12th Nordiske datalingvistikkdager, Department of Linguistics, Norwegian University of Science and Technology (NTNU), Nordgard (December 9-10 2000)
12. Birn, J.: Detecting Grammar Errors with Lingsofts Swedish Grammar-checker. In: Proceedings from the 12th Nordiske datalingvistikkdager, Department of Linguistics, Norwegian University of Science and Technology (NTNU), Nordgard (December 9-10 2000)

13. Badia, T., Gil, A., Quixal, M., Valentín, O.: NLP-enhanced Error Checking for Catalan Unrestricted Text. In: Proceedings of the fourth international conference on Language Resources and Evaluation, LREC 2004, Lisbon, Portugal (2004) 1919–1922
14. Karttunen, L., Gaál, T., Kempe, A.: Xerox Finite State Tool. Technical report, Xerox Research Centre Europe (1997)
15. Hashemi, S.S., Cooper, R., Andersson, R.: Positive Grammar Checking: A Finite State Approach. In: Computational Linguistics and Intelligent Text Processing, 4th International Conference, CICLing 2003, Mexico City, Mexico, February 16-22. Volume 2588 of Lecture Notes in Computer Science., Springer (2003) 635–646
16. Atwell, E., Elliot, S.: Dealing with Ill-Formed English Text. In: The Computational Analysis of English: a Corpus-Based Approach. De Longman (1987)
17. Karttunen, L.: Numbers and Finnish Numerals. In A Man of Measure Festschrift in Honour of Fred Karlsson on his 60th Birthday, a special supplement to SKY Journal of Linguistics **19** (2006) 407–421
18. Gojenola, K., Oronoz, M.: Corpus-based Syntactic Error Detection Using Syntactic Patterns. In: NAACL-ANLP00,Student Research Workshop, Seattle, USA (April 30 2000)
19. Aduriz, I., Aldezabal, I., Alegria, I., Arriola, J.M., Díaz de Ilarraza, A., Ezeiza, N., Gojenola, K.: Finite State Applications for Basque. In: EACL 2003 Workshop on Finite-State Methods in Natural Language Processing. (2003)
20. Koskenniemi, K.: Two-Level Morphology: a General Ccomputational Model for Word-form Recognition and Production. University of Helsinki, Helsinki (1983)
21. Ezeiza, N.: Corpusak ustiatzeko tresna linguistikoak. Euskararen etiketatzaile sintaktiko sendo eta malgua. PhD thesis. University of the Basque Country, Donostia (2003)
22. Artola, X., Díaz de Ilarraza, A., Ezeiza, N., Gojenola, K., Labaka, G., Sologaistoa, A., Soroa, A.: A Framework for Representing and Managing Linguistic Annotations Based on Typed Feature Structures. In: Proceedings of Recent Advances on NLP (RANLP05), Borovets, Bulgaria (2005)
23. Otaegi, M.: Datak, orduak eta zenbakiak euskaraz. Technical report, University of the Basque Country (2006)
24. Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI Publications (2003)
25. Sjöbergh, J., Knutsson, O.: Faking Errors to Avoid Making Errors: Very Weakly Supervised Learning for Error Detection in Writing. In: Proceedings of RANLP 2005, Borovets, Bulgaria (2005) 506–512
26. Wagner, J., Foster, J., van Genabith, J.: A Comparative Evaluation of Deep and Shallow Approaches to the Automatic Detection of Common Grammatical Errors. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). (2007) 112–121

Temporal propositions as regular languages

Tim Fernando

Trinity College Dublin

Abstract. Temporal propositions are mapped to sets of strings that witness (in a precise sense) the propositions over discrete linear Kripke frames. The strings are collected into regular languages to ensure the decidability of entailments given by inclusions between languages. (Various notions of bounded entailment are shown to be expressible as language inclusions.) The languages unwind computations implicit in the logical (and temporal) connectives via a system of finite-state constraints adapted from finite-state morphology. Applications to Hybrid Logic and non-monotonic inertial reasoning are briefly considered.

1 Introduction

Model-theoretic semantics cashes out the meaning of a formula φ by specifying when a model M satisfies φ. The present work converts a temporal formula φ to a regular language that indicates (in a precise sense) which models M satisfy φ. The strings in these languages are not only easier to grasp computationally than the models M but are natural candidates for events witnessing φ. The languages are formed by adapting finite-state methods widely used in morphology [1].

An instructive example of a temporal formula that can be converted to a regular language is p *until* q, which is equivalent to the disjunction

$$q \ \vee \ (p \wedge next(q)) \ \vee \ (p \wedge next(p \wedge next(q))) \ \vee \ \cdots \tag{1}$$

over discrete linear orders. If we draw the disjuncts as $\boxed{q}$, $\boxed{p}\boxed{q}$, $\boxed{p}\boxed{p}\boxed{q}$ and so on, and rewrite $\vee$ as non-deterministic choice $|$, then (1) becomes the language

$$\boxed{q} \mid \boxed{p}\boxed{q} \mid \boxed{p}\boxed{p}\boxed{q} \mid \ \cdots = \boxed{p}^*\boxed{q}$$

where $\cdot^*$ is Kleene star (for zero or more iterations). In general, given a finite set Φ of formulas and a string $a_1 a_2 \cdots a_n \in Pow(\Phi)^*$ of subsets a_i of Φ, let $fmla_0(a_1 a_2 \cdots a_n)$ be the conjunction

$$fmla_0(a_1 a_2 \cdots a_n) \stackrel{\text{def}}{=} (\bigwedge a_1) \wedge next(\bigwedge a_2) \wedge \cdots \wedge next^{n-1}(\bigwedge a_n)$$

with conjuncts $next^{i-1}(\bigwedge a_i)$ that are themselves built from conjunctions $\bigwedge a_i$. The conjunction of the empty set is, as usual, some fixed tautology $\top$. For instance, we have

$$fmla_0(\boxed{p}\Box\boxed{q,r}) = p \wedge next(next(q \wedge r))$$

where formulas constituting a symbol a_i in a string are enclosed by a box rather than by curly braces $\{\cdot\}$, and the tautology $\top$ for the conjunction $\bigwedge\Box$ of the empty set $\Box$ is suppressed. Also, $\mathit{fmla}_0(\boxed{p}^0\boxed{q}) = q$ and

$$\mathit{fmla}_0(\boxed{p}^{n+1}\boxed{q}) = p \wedge \cdots \wedge \mathit{next}^n(p) \wedge \mathit{next}^{n+1}(q) \ ,$$

making p *until* q equivalent to the disjunction

$$\bigvee \{\mathit{fmla}_0(s) : s \in \boxed{p}^*\boxed{q}\}$$

over the models of interest.

1.1 Representations over the integers

Before specifying what "the models of interest" are, let us not forget past operators such as the converse *prev* of *next*, and sharpen the map fmla_0 accordingly. To mark out the present, we introduce a fresh formula $\mathit{now} \notin \Phi$, refining our picture $\boxed{p}\boxed{q}$ for $p \wedge \mathit{next}(q)$ to $\boxed{\mathit{now}, p}\boxed{q}$, so as to represent $\mathit{prev}(p) \wedge q$ as $\boxed{p}\boxed{\mathit{now}, q}$. Given a subset a of Φ, let us write $a_\dagger$ for the union $a \cup \boxed{\mathit{now}}$, drawing a box instead of $\{\cdot\}$ as we shall form strings from such sets. Let us call a string now_Φ*-pointed* if it has the form $sa_\dagger s'$ for some strings s and s' over the alphabet $\mathit{Pow}(\Phi)$ and some subset a of Φ. We define a backward version $\mathit{almf}(s)$ of $\mathit{fmla}_0(s)$

$$\mathit{almf}(a_1 a_2 \cdots a_n) \stackrel{\text{def}}{=} \mathit{prev}^n(\bigwedge a_1) \wedge \mathit{prev}^{n-1}(\bigwedge a_2) \wedge \cdots \wedge \mathit{prev}(\bigwedge a_n)$$

and map a now_Φ-pointed string $sa_\dagger s'$ to the conjunction

$$\mathit{fmla}(sa_\dagger s') \stackrel{\text{def}}{=} \mathit{almf}(s) \ \wedge \ \mathit{fmla}_0(as') \ .$$

For example, $\mathit{almf}(\boxed{p}\Box) = \mathit{prev}(\mathit{prev}(p))$ and thus,

$$\mathit{fmla}(\boxed{p}\Box\boxed{\mathit{now}, q, r}) = \mathit{prev}(\mathit{prev}(p)) \wedge q \wedge r$$

(dropping $\top$ as before).

Turning now to models, we base our Kripke models for a set P of atomic formulas on not only the natural numbers (for future operators) but also the negative integers (for the past). Let $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$ be the set of integers, and *satisfaction* $\models$ be defined relative to an integer $x \in \mathbb{Z}$ and a function $V : P \to \mathit{Pow}(\mathbb{Z})$, called a valuation, as follows. For an atomic formula $p \in P$, we set

$$\langle V, x\rangle \models p \iff_{\text{def}} x \in V(p) \ .$$

The unary connective *next* moves us to the successor $x + 1$

$$\langle V, x\rangle \models \mathit{next}(\varphi) \iff_{\text{def}} \langle V, x+1\rangle \models \varphi$$

while *prev* steps back to the preceding integer $x-1$

$$\langle V,x\rangle \models \mathit{prev}(\varphi) \stackrel{\text{def}}{\iff} \langle V,x-1\rangle \models \varphi \ .$$

Conjunctions are formed from the binary connective $\wedge$

$$\langle V,x\rangle \models \varphi\wedge\psi \stackrel{\text{def}}{\iff} \langle V,x\rangle \models \varphi \text{ and } \langle V,x\rangle \models \psi$$

as well as the 0-ary connective $\top$

$$\langle V,x\rangle \models \top \ .$$

In what follows, we let Φ denote some fixed *finite* set of formulas on which $\models$ is well-defined, and form now_Φ-pointed strings s over the alphabet $\mathit{Pow}(\Phi\cup\{\mathit{now}\})$ with formulas $\mathit{fmla}(s)$ on which $\models$ is well-defined (under the clauses above for *next*, *prev*, $\wedge$ and $\top$). We leave the full specification of clauses for $\models$ open-ended, introducing clauses such as

$$\begin{aligned}\langle V,x\rangle \models \varphi \ \mathit{until}\ \psi \stackrel{\text{def}}{\iff} (\exists y\geq x)\ &\langle V,y\rangle \models \psi \text{ and}\\ &(\forall z<y)\ z\geq x \text{ implies } \langle V,z\rangle \models \varphi\end{aligned}$$

to pose the problem of representing a formula such as $\mathit{prev}(p \ \mathit{until}\ (q\wedge r))$.

Definition. A set L of now_Φ-pointed strings *stringwise* Φ*-represents* φ if φ is equivalent to the disjunction $\bigvee\{\mathit{fmla}(s) : s\in L\}$ in that

$$\langle V,x\rangle \models \varphi \iff (\exists s\in L)\ \langle V,x\rangle \models \mathit{fmla}(s)$$

for all $V : P\to \mathit{Pow}(\mathbb{Z})$ and $x\in\mathbb{Z}$.

It is easy to see that $\mathit{prev}(p \ \mathit{until}\ (q\wedge r))$ is stringwise Φ-represented by

$$\boxed{q,r}\boxed{\mathit{now}} \mid \boxed{p}\boxed{\mathit{now},q,r} \mid \boxed{p}\boxed{\mathit{now},p}\boxed{p}^*\boxed{q,r}$$

assuming $p,q,r\in\Phi$. Indeed, every $\varphi\in\Phi$ is stringwise Φ-represented by $\boxed{\mathit{now},\varphi}$. For $\varphi\notin\Phi$, the idea is to turn $\boxed{\mathit{now},\varphi}$ into an "equivalent" set of now_Φ-pointed strings. But some cases are hopeless.

Consider, for instance, the formula $\mathsf{G}\varphi$ asserting that φ is true now and at every point in the future

$$\langle V,x\rangle \models \mathsf{G}\varphi \stackrel{\text{def}}{\iff} (\forall y\geq x)\ \langle V,y\rangle \models \varphi \ .$$

If $\Phi\subseteq P$ is a set of atomic formulas and $p\in P$, then $\mathsf{G}p$ has no stringwise Φ-representation. But we will (in section 4 below) modify the notion of representation so that

$$\Box^*\boxed{\mathit{now},p}\boxed{p}^* \ \mathit{path}\text{wise } \{p\}\text{-represents } \mathsf{G}p$$

and in general, for every set L of now_{Φ}-pointed strings,

$$L \text{ stringwise } \Phi\text{-represents } \varphi \text{ implies } \Box^* L \Box^* \text{ pathwise } \Phi\text{-represents } \varphi.$$

Stringwise or pathwise, can we ensure that our Φ-representations are regular languages? In view of the prevalence of automata-theoretic methods in temporal logic [2, 3], it is perhaps not surprising how much we can. Nonetheless, some delicacy is required to keep these languages regular. For instance, a straightforward analog (within the present setting) of the replace operator in [1] takes us outside the realm of finite automata (see §3.2 below). Instead, we adapt Koskenniemi's restriction operator [1] over an alphabet of symbols that have structure reflecting concurrent computation.

But why should it matter that the languages are regular? One of many useful properties of regular languages is the decidability of inclusions $\subseteq$ between them (as opposed say, to context-free languages). In the present context, entailments are naturally expressed as inclusions between languages (§2.2 below). The regularity of these languages gives us a computational handle on entailments that arguably compensates for the loss of first-order logic due to infinitary disjunctions from Kleene star.[1]

1.2 Related work

Inclusions between languages figure prominently in *Model Checking*, where a system $\mathcal{A}$ satisfies specification $\mathcal{S}$ precisely if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$ for certain languages $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{S})$ associated with $\mathcal{A}$ and $\mathcal{S}$, respectively ([3], page 124). The languages in this case consist of infinite strings accepted according to Büchi's criterion. That criterion is closely related to the notion of pathwise Φ-representation spelled out in §4.2 below.

Staying with finite strings, the present approach uses inclusions to define entailments relative to constraints derived from those in [1] through an operation called superposition [4], reviewed in §2.1 below. Superposition supports a form of "true" concurrency different from the non-deterministic interleaving typically associated with temporal logic ([2], page 1017). The notions of entailment induced by superposition do *not* in general preserve length, and therefore cannot be regular under the conventions of *Regular Model Checking* [5]. They are, however, computable by finite-state transducers with componentwise ϵ-moves, and are therefore regular according to [1]. A notable difference between [5] and [1] is that regular relations are closed under intersection in the former but not in the latter. Henceforth, we adopt the latter definition of a regular relation (relaxing the requirement of length preservation). What is important for our purposes is that given a relation R and a language L, the *L-restriction R_L of R* defined by

$$R_L \stackrel{\text{def}}{=} \{(s, s') \in R \ : \ s' \in L\}$$

[1] A routine compactness argument shows that the disjunction (1) above cannot be expressed as a first-order theory.

and its domain, the *Peirce product* $\langle R\rangle L$ *of* R *with* L

$$\langle R\rangle L \stackrel{\text{def}}{=} \{s \ : \ (\exists s')\ sR_Ls'\}\ ,$$

are both regular if R and L are.[2] As explained in §3.1 below, Koskenniemi's restriction operation can be derived from the Peirce product. The transitive closure of a regular relation, length preserving or not, need not be regular. An example in the present context is provided by the obvious analog of the replace operation from [1]. Because its transitive closure need not be regular, we resort instead to Koskenniemi-esque constraints.

2 Inclusions comparing information content

This section shows how to compare the information content of languages through inclusion $\subseteq$. The first way, through subsumption $\trianglerighteq$, is described most directly via an operation of superposition. The second way, through weak subsumption $\underline{\blacktriangleright}$, allows for $\Box$-padding in $\trianglerighteq$. As relations on strings, both $\trianglerighteq$ and $\underline{\blacktriangleright}$ are regular, and can be lifted to languages through Peirce products, leading to natural notions of bounded entailment.

2.1 Superposition and subsumption

Given languages L and L' over the alphabet $Pow(\Phi \cup \{now\})$, the *superposition* $L\&L'$ *of* L *and* L' consists of the componentwise union of strings in L and L' of the same length

$$L\&L' \stackrel{\text{def}}{=} \bigcup_{n\geq 0}\{(a_1 \cup a'_1)\cdots(a_n \cup a'_n)\ :\ a_1\cdots a_n \in L \text{ and } a'_1\cdots a'_n \in L'\}$$

[4]. For instance, the superposition of $\boxed{p}^*\boxed{q}$ and $\boxed{now}\Box^*$ injects *now* into the first box of every string in $\boxed{p}^*\boxed{q}$

$$\boxed{p}^*\boxed{q}\ \&\ \boxed{now}\Box^* = \boxed{now, q}\ |\ \boxed{now, p}\boxed{p}^*\boxed{q}\ .$$

[2] The terminology "Peirce product" is from [6], but the notation $\langle R\rangle L$ is borrowed from dynamic logic [7]. A finite-state transducer for R with transitions $\rightarrow_R$ and a finite automata for L with transitions $\rightarrow_L$ combine to form a finite-state transducer for R_L with transitions

$$(q, q') \stackrel{a,b}{\rightarrow} (r, r') \iff^{\text{def}} q \stackrel{a,b}{\rightarrow}_R r \text{ and } (q' \stackrel{b}{\rightarrow}_L r' \text{ or } (b = \epsilon \text{ and } q' = r'))$$

(as in the usual construction for the intersection of regular languages, but with ϵ-moves). For the Peirce product $\langle R\rangle L$, we existentially quantify b out for

$$(q, q') \stackrel{a}{\rightarrow} (r, r') \iff^{\text{def}} (\exists b)\ q \stackrel{a,b}{\rightarrow}_R r \text{ and } (q' \stackrel{b}{\rightarrow}_L r' \text{ or } (b = \epsilon \text{ and } q' = r'))\ .$$

To accept $L\&L'$ given finite automata A and A' accepting L and L' respectively, we run A and A' in lockstep. That is, an automaton accepting $L\&L'$ can be built as in the usual product construction for the intersection $L \cap L'$ except that the transitions $\rightsquigarrow$ for $L\&L'$ are obtained by unioning the labels on the transitions $\rightarrow_A$ of A and $\rightarrow_{A'}$ of A'

$$(q, q') \overset{b}{\rightsquigarrow} (r, r') \overset{\text{def}}{\iff} (\exists a, a' \subseteq b)\ b = a \cup a' \text{ and } q \overset{a}{\rightarrow}_A r \text{ and } q' \overset{a'}{\rightarrow}_{A'} r' \ .$$

The definition of superposition depends on the assumption that our alphabet consists of sets closed under union. Instead of using subsets of $\Phi \cup \{now\}$ as symbols in our alphabet, we can add a fresh symbol $\wr \notin \Phi \cup \{now\}$, pronounced "tick" (as in a ticking clock), and rewrite, for instance, the string $\boxed{now, p}\boxed{q}$ of length 2 to the string $now\ p\ \wr\ q\ \wr$ of length 5 over the alphabet $\{p, q, now, \wr\}$. This is essentially the approach pursued in [8], taking us back to the usual interleaving model of concurrency except that the passage of time is marked by a tick $\wr$. As noted in [9], one can build finite-state transducers between $Pow(\Phi \cup \{now\})^*$ and $(\Phi \cup \{now, \wr\})^*$ that translate between these in the obvious way and preserve regularity. For convenience, we work with the alphabet $Pow(\Phi \cup \{now\})$, abbreviating it to Σ when the exact choice of Φ is immaterial.

Superposition allows us to compare the information content of languages L and L' over Σ as follows. We say L *subsumes* L' and write $L \trianglerighteq L'$ if L is included in the superposition $L\&L'$

$$L \trianglerighteq L' \overset{\text{def}}{\iff} L \subseteq L\&L' \ .$$

Conflating a string s with the language $\{s\}$, it follows that for strings $a_1 \cdots a_n$ and $b_1 \cdots b_m$ over Σ, $\trianglerighteq$ picks out pairs with the same length $n = m$ and are componentwise related by the converse of $\subseteq$

$$a_1 \cdots a_n \trianglerighteq b_1 \cdots b_m \iff n = m \text{ and } a_i \supseteq b_i \text{ for } 1 \leq i \leq n \ .$$

As a relation on strings, $\trianglerighteq$ is regular; it is computable by a finite-state transducer with one state q_0, both initial and final, and transitions

$$q_0 \overset{a,b}{\rightarrow} q_0 \quad \overset{\text{def}}{\iff} \quad b \subseteq a$$

for all $b, a \in \Sigma$. Taking the Peirce product of $\trianglerighteq$ with a language L, we get

$$s \in \langle \trianglerighteq \rangle L \iff (\exists s' \in L)\ s \trianglerighteq s' \ .$$

We can then restate $L \trianglerighteq L'$ as an inclusion involving the Peirce product $\langle \trianglerighteq \rangle L'$

$$L \trianglerighteq L' \iff L \subseteq \langle \trianglerighteq \rangle L' \ .$$

2.2 Padding and entailments

Although the relation $\trianglerighteq$ will prove useful for formulating constraints later on, it will also be convenient to weaken it slightly so that strings of different length can

be compared. Towards this end, we define for every string $s \in \Sigma^*$ its unpadded form, $unpad(s)$, obtained by deleting all initial and final $\Box$'s from s. That is,

$$unpad(s) \stackrel{\text{def}}{=} \begin{cases} s & \text{if } s \text{ neither begins nor ends with } \Box \\ unpad(s') & \text{if } s = \Box s' \text{ or else if } s = s'\Box \end{cases}$$

so that for example, $unpad(\Box\boxed{p}\Box\boxed{now}\Box\Box) = \boxed{p}\Box\boxed{now}$. As a relation between strings, *unpad* is obviously regular (so long as we don't require length preservation). Next, we say that a string s *weakly subsumes* s' and write $s \underline{\blacktriangleright} s'$ if s subsumes some string equivalent to s' up to unpadding

$$s \underline{\blacktriangleright} s' \stackrel{\text{def}}{\iff} (\exists s'')\ s \trianglerighteq s'' \text{ and } unpad(s'') = unpad(s')\ .$$

It is easy to see that the relation of *unpad*-equivalence

$$\{(s, s') : unpad(s) = unpad(s')\}$$

is regular, making weak subsumption $\underline{\blacktriangleright}$ regular (since regular relations are closed under relational composition).

If we think of strings in a language as possibilities in the same way that worlds in a proposition are under possible worlds semantics (or models of a sentence are in model-theoretic semantics), then it is natural to lift $\underline{\blacktriangleright}$ to sets L, L' of strings through the Peirce product

$$\begin{aligned} L \underline{\blacktriangleright} L' &\stackrel{\text{def}}{\iff} L \subseteq \langle \underline{\blacktriangleright} \rangle L' \\ &\iff (\forall s \in L)(\exists s' \in L')\ s \underline{\blacktriangleright} s' \end{aligned}$$

(paralleling the definition in possible worlds semantics that a proposition p entails p' if $p \subseteq p'$). Defining $L'_\Box$ to be the set of strings *unpad*-equivalent to strings in L'

$$\begin{aligned} L'_\Box &\stackrel{\text{def}}{=} \{s \in \Sigma^* \ :\ (\exists s' \in L')\ unpad(s) = unpad(s')\} \\ &= \Box^* unpad(L') \Box^* \end{aligned}$$

(where $unpad(L') \stackrel{\text{def}}{=} \{unpad(s) : s \in L'\}$), we can relate $\underline{\blacktriangleright}$ back to superposition & via subsumption $\trianglerighteq$ and (un)padding

$$\begin{aligned} L \underline{\blacktriangleright} L' &\iff L \trianglerighteq L'_\Box \\ &\iff L \subseteq L \& L'_\Box\ . \end{aligned}$$

As some strings may represent spurious possibilities, we can weed out strings from $\langle \underline{\blacktriangleright} \rangle L$ by intersecting it with a language C to form

$$\begin{aligned} C[L] &\stackrel{\text{def}}{=} C \cap \langle \underline{\blacktriangleright} \rangle L \\ &= \{s \in C \ :\ s \underline{\blacktriangleright} L\} \end{aligned}$$

which is a regular language whenever L and C are. Recall that regular languages are closed under Boolean operations, including complementation

$$\overline{L} \stackrel{\text{def}}{=} \Sigma^* - L \ .$$

We can express the set of now_{Φ}-pointed strings as $C'[L']$ if we choose C' and L' as follows. Let $L' \stackrel{\text{def}}{=} \boxed{now}$ and let C' be the set of strings that do *not* contain two occurrences of *now*

$$\begin{aligned} C' &\stackrel{\text{def}}{=} \overline{\langle \underline{\blacktriangleright} \rangle (\boxed{now} \Box^* \boxed{now})} \\ &= [\underline{\blacktriangleright}] \overline{\boxed{now} \Box^* \boxed{now}} \end{aligned}$$

where $[R]L$ is the dual of the Peirce product $\langle R \rangle L$

$$[R]L \stackrel{\text{def}}{=} \overline{\langle R \rangle \overline{L}}$$

just as $\forall$ is the dual of $\exists$.

In general, we can beef up $L \underline{\blacktriangleright} L'$ to an entailment $L \vdash_C L'$ by relativizing it to a language C that turns L to $C[L]$

$$\begin{aligned} L \vdash_C L' &\stackrel{\text{def}}{\iff} C[L] \ \underline{\blacktriangleright} \ L' \\ &\iff C \cap \langle \underline{\blacktriangleright} \rangle L \ \subseteq \ \langle \underline{\blacktriangleright} \rangle L' \ . \end{aligned}$$

Clearly, $L \vdash_C L'$ whenever $L \underline{\blacktriangleright} L'$. The introduction of C allows us not only to enlarge a string in L, but also to restrict attention to strings meeting the membership conditions for C

$$L \vdash_C L' \iff (\forall s \in C) \ s \underline{\blacktriangleright} L \text{ implies } s \underline{\blacktriangleright} L' \ .$$

These membership conditions can be viewed as constraints (to satisfy), as we see next.

3 Constraints and their application

In this section, we formulate constraints corresponding to the semantic clauses for $\vee, \wedge, \mathit{next}, \mathit{prev}, \mathit{until}$ and *since*, and apply them to build stringwise representations. For this, a useful regular relation between strings is that of a factor: s' is a *factor of* s if $s = us'v$ for some (possibly empty) strings u and v.

3.1 Constraints conditioned by subsumption

Given languages L and L' over Σ, let $L \Rightarrow L'$ be the set of strings s such that every factor of s that subsumes L also subsumes L'

$$\begin{aligned} L \Rightarrow L' \stackrel{\text{def}}{=} \{ s \in \Sigma^* \ : \ &\text{for every factor } s' \text{ of } s, \\ &s' \trianglerighteq L \text{ implies } s' \trianglerighteq L' \} \ . \end{aligned}$$

For example, to pick out strings that contain $\varphi \wedge \psi$ only if they contain φ and ψ in the same box , we let

$$\boxed{\varphi \wedge \psi} \Rightarrow \boxed{\varphi, \psi} \tag{2}$$

and for disjunction $\varphi \vee \psi$,

$$\boxed{\varphi \vee \psi} \Rightarrow \boxed{\varphi} \mid \boxed{\psi} \; . \tag{3}$$

Writing $\sqsupseteq^L$ for the $\langle \trianglerighteq \rangle L$-restriction of the factor relation

$$s \sqsupseteq^L s' \stackrel{\text{def}}{\iff} s' \text{ is a factor of } s \text{ and } s' \in \langle \trianglerighteq \rangle L \; ,$$

it follows that

$$L \Rightarrow L' = [\sqsupseteq^L] \langle \trianglerighteq \rangle L' \; .$$

As the factor relation is regular, so is $\sqsupseteq^L$ for regular languages L. Thus, since the Peirce product of a regular relation with a regular language is regular, $L \Rightarrow L'$ is a regular language if L and L' are. Indeed,

$$L \Rightarrow L' = \overline{\Sigma^* \; (\langle \trianglerighteq \rangle L \cap \overline{\langle \trianglerighteq \rangle L'}) \; \Sigma^*} \; .$$

Next, we strengthen the constraint $\boxed{next(\varphi)}\Box \Rightarrow \Box\boxed{\varphi}$ to

$$\boxed{next(\varphi)} \stackrel{\mathsf{a}}{\Rightarrow} \boxed{\varphi} \tag{4}$$

where $L \stackrel{\mathsf{a}}{\Rightarrow} L'$ is pronounced "L' after every L" and

$$s \in L \stackrel{\mathsf{a}}{\Rightarrow} L' \stackrel{\text{def}}{\iff} \begin{array}{l} \text{after every factor of } s \text{ that subsumes } L \\ \qquad \text{is a substring that subsumes } L' \end{array}$$

for every string $s \in \Sigma^*$. Defining

$$s \; after^L \; s' \stackrel{\text{def}}{\iff} (\exists u \trianglerighteq \Box^* L) \; s = us' \; ,$$

we get

$$L \stackrel{\mathsf{a}}{\Rightarrow} L' = [after^L] \; \langle \trianglerighteq \rangle (L' \Box^*) \; .$$

It is not difficult to convert a finite automaton for L into a finite-state transducer for $after_L$. Hence, $L \stackrel{\mathsf{a}}{\Rightarrow} L'$ is regular if L and L' are. In fact,

$$L \stackrel{\mathsf{a}}{\Rightarrow} L' = \overline{\langle \trianglerighteq \rangle (\Box^* L) \; \overline{\langle \trianglerighteq \rangle (L' \Box^*)}} \; .$$

Modulo subsumption $\trianglerighteq$, $L \stackrel{\mathsf{a}}{\Rightarrow} L'$ is one form of Koskenniemi's restrictions [1], a second one being $L \stackrel{\mathsf{b}}{\Rightarrow} L'$, read "$L'$ before every L," defined by

$$L \stackrel{\mathsf{b}}{\Rightarrow} L' \stackrel{\text{def}}{=} [before^L] \; \langle \trianglerighteq \rangle (\Box^* L')$$

where

$$s \; before^L \; s' \stackrel{\text{def}}{\iff} (\exists v \trianglerighteq L\Box^*) \; s = s'v \; .$$

As with $\Rightarrow$ and $\stackrel{\mathsf{a}}{\Rightarrow}$, $L \stackrel{\mathsf{b}}{\Rightarrow} L'$ is regular if L and L' are, with

$$L \stackrel{\mathsf{b}}{\Rightarrow} L' = \overline{\overline{\langle\trianglerighteq\rangle(\Box^* L')} \; \langle\trianglerighteq\rangle(L\Box^*)} \; .$$

We also strengthen $\Box\boxed{prev(\varphi)} \Rightarrow \boxed{\varphi}\Box$ to

$$\boxed{prev(\varphi)} \stackrel{\mathsf{b}}{\Rightarrow} \boxed{\varphi} \; . \tag{5}$$

Both $\Rightarrow$ and $\stackrel{\mathsf{a}}{\Rightarrow}$ are used to analyze *until* through auxiliary formulas $\varphi \; ntil \; \psi$

$$\boxed{\varphi \; until \; \psi} \Rightarrow \boxed{\psi} \mid \boxed{\varphi, \; \varphi \; ntil \; \psi} \tag{6}$$

with

$$\boxed{\varphi \; ntil \; \psi} \stackrel{\mathsf{a}}{\Rightarrow} \boxed{\varphi}^* \boxed{\psi} \; . \tag{7}$$

We can treat *since* similarly, using $\stackrel{\mathsf{b}}{\Rightarrow}$ and auxiliary formulas $\varphi \; sinc \; \psi$

$$\boxed{\varphi \; since \; \psi} \Rightarrow \boxed{\psi} \mid \boxed{\varphi, \; \varphi \; sinc \; \psi} \tag{8}$$

$$\boxed{\varphi \; sinc \; \psi} \stackrel{\mathsf{b}}{\Rightarrow} \boxed{\psi}\boxed{\varphi}^* \; . \tag{9}$$

3.2 Application with minimization and projection

Let $P_\bullet$ be the set of formulas constructed from P using $\top, \wedge, \vee, next, prev, until, since, ntil$ and $sinc$. We define a function $\mathcal{C} : P_\bullet \to Pow(P_\bullet \cup \{now\})^*$ mapping a formula $\varphi \in P_\bullet$ to a language $\mathcal{C}(\varphi)$ over the alphabet $Pow(P_\bullet \cup \{now\})$ by induction on φ, using the constraints we have associated above with the connectives

$$\begin{aligned}
\mathcal{C}(\varphi) &\stackrel{\text{def}}{=} [\blacktriangleright]\overline{\boxed{now}\Box^*\boxed{now}} \quad \text{for } \varphi \in P \cup \{\top\} \\
\mathcal{C}(\varphi \wedge \psi) &\stackrel{\text{def}}{=} \mathcal{C}(\varphi) \cap \mathcal{C}(\psi) \cap (\boxed{\varphi \wedge \psi} \Rightarrow \boxed{\varphi, \psi}) \\
\mathcal{C}(\varphi \vee \psi) &\stackrel{\text{def}}{=} \mathcal{C}(\varphi) \cap \mathcal{C}(\psi) \cap (\boxed{\varphi \vee \psi} \Rightarrow \boxed{\varphi} \| \boxed{\psi}) \\
\mathcal{C}(next(\varphi)) &\stackrel{\text{def}}{=} \mathcal{C}(\varphi) \cap (\boxed{next(\varphi)} \stackrel{\mathsf{a}}{\Rightarrow} \boxed{\varphi}) \\
\mathcal{C}(prev(\varphi)) &\stackrel{\text{def}}{=} \mathcal{C}(\varphi) \cap (\boxed{prev(\varphi)} \stackrel{\mathsf{b}}{\Rightarrow} \boxed{\varphi}) \\
\mathcal{C}(\varphi \; until \; \psi) &\stackrel{\text{def}}{=} \mathcal{C}(\varphi \; ntil \; \psi) \cap (\boxed{\varphi \; until \; \psi} \Rightarrow \boxed{\psi} \| \boxed{\varphi, \; \varphi \; ntil \; \psi}) \\
\mathcal{C}(\varphi \; ntil \; \psi) &\stackrel{\text{def}}{=} \mathcal{C}(\varphi) \cap \mathcal{C}(\psi) \cap (\boxed{\varphi \; ntil \; \psi} \stackrel{\mathsf{a}}{\Rightarrow} \boxed{\varphi}^* \boxed{\psi})
\end{aligned}$$

and similarly for *since* and *sinc*. For each $\varphi \in P_\bullet$, the language $\mathcal{C}(\varphi)$ is regular, as is the language

$$\mathcal{C}(\varphi) \cap \langle \blacktriangleright \rangle \boxed{now, \varphi} = \{s \in \mathcal{C}(\varphi) : s \blacktriangleright \boxed{now, \varphi}\}$$

which we shall abbreviate $\hat{\mathcal{C}}(\varphi)$. The language $\hat{\mathcal{C}}(\varphi)$ can be quite massive, but we can reduce it a few ways. The first is through $\trianglerighteq$-minimization: given a language L, define the set $L_{\trianglerighteq}$ of $\trianglerighteq$-*minimal strings in* L by

$$L_{\trianglerighteq} \stackrel{\text{def}}{=} L - \langle \triangleright \rangle L$$

where $\triangleright$ is $\trianglerighteq$ minus equality

$$s \triangleright s' \stackrel{\text{def}}{\iff} s \trianglerighteq s' \text{ and } s \neq s' \ .$$

For example, $(\Box^*|L)_{\trianglerighteq} = \Box^*$. Notice that $L_{\trianglerighteq}$ is regular if L is. The second way of trimming a language is by projecting every string $a_1 \cdots a_n$ in it to the string

$$\rho(a_1 \cdots a_n) \stackrel{\text{def}}{=} (a_1 \cap (P \cup \boxed{now})) \cdots (a_n \cap (P \cup \boxed{now}))$$

restricting the symbols to subsets of $P \cup \boxed{now}$. For instance, if $p \in P$ then $\rho(\boxed{p, \psi \vee \varphi}\boxed{now, prev(\chi)}) = \boxed{p}\boxed{now}$. In general, if L is a regular language, then so is $\{unpad(\rho(s)) : s \in L\}$. Moreover, an argument by induction on $\varphi \in P_\bullet$ establishes

Theorem 1. *Every $\varphi \in P_\bullet$ is stringwise P-represented by the regular language $\{unpad(\rho(s)) : s \in \hat{\mathcal{C}}(\varphi)_{\trianglerighteq}\}$.*

Remark The projection ρ drops $\top$. Writing $\mathsf{F}\varphi$ for $\top$ *until* φ as usual (and $\mathsf{P}\varphi$ for $\top$ *since* φ), one might try to replace $\boxed{\mathsf{F}q}\Box$ by $\boxed{q}\Box \mid \Box\boxed{\mathsf{F}q}$. But doing so in $\boxed{p, \mathsf{F}q}^+\boxed{r}\Box^*$ (where $L^+ \stackrel{\text{def}}{=} L^*L$) can lead to non-regularity, as intersection with the regular language $\boxed{p}^+\boxed{r}\boxed{q}^+$ yields the non-regular language

$$\{\boxed{p}^n\boxed{r}\boxed{q}^m : n \geq m \geq 1\}$$

with regular sublanguage $\boxed{p}^+\boxed{r}\boxed{q}$ obtained by $\trianglerighteq$-minimization and *unpad*.

4 Negation and paths for infinite strings

We turn next to negation and formulas such as $\mathsf{G}\varphi$ left out of Theorem 1.

4.1 Negation

The obvious constraints to associate with Boolean negation $\neg$

$$\langle V, x\rangle \models \neg\varphi \stackrel{\text{def}}{\iff} \text{not } \langle V, x\rangle \models \varphi$$

are non-contradiction $[\blacktriangleright]\,\overline{\boxed{\varphi, \neg\varphi}}$ and excluded middle

$$\Box \Rightarrow \boxed{\varphi} \mid \boxed{\neg\varphi}\ .$$

A popular alternative that we will adopt is to treat negation as a function $\varphi \mapsto \overline{\varphi}$ on formulas φ such that $\overline{\overline{\varphi}} = \varphi$ and $\overline{p} \in P$ for every $p \in P$ (if necessary, doubling P to $P \times \{+,-\}$ with $\overline{(p,+)} = (p,-)$ and $\overline{(p,-)} = (p,+)$). The functions (valuations) V are then required to satisfy $V(p) \cap V(\overline{p}) = \emptyset$, suggesting

$$[\blacktriangleright]\,\overline{\boxed{p, \overline{p}}}\ , \tag{10}$$

and $V(p) \cup V(\overline{p}) = \mathbb{Z}$. Every n-ary connective θ is paired with an n-ary connective $\overline{\theta}$ so that $\overline{\overline{\theta}} = \theta$ and

$$\overline{\theta(\varphi_1, \ldots, \varphi_n)} \stackrel{\text{def}}{=} \overline{\theta}(\overline{\varphi_1}, \ldots, \overline{\varphi_n})\ .$$

De Morgan's laws suggest $\overline{\vee} \stackrel{\text{def}}{=} \wedge$, $\overline{\theta} \stackrel{\text{def}}{=} \theta$ for $\theta \in \{\mathit{next},\ \mathit{prev}\}$, $\overline{\top} \stackrel{\text{def}}{=} \bot$ with

$$[\blacktriangleright]\,\overline{\boxed{\bot}} \tag{11}$$

(as $\langle V, x\rangle \not\models \bot$) and $\overline{\mathit{until}} \stackrel{\text{def}}{=} \mathit{release}$ where

$$\begin{aligned}\langle V, x\rangle \models \varphi\ \mathit{release}\ \psi \iff\ & (\forall y \geq x)\ \langle V, y\rangle \models \psi \text{ or} \\ & (\exists z < y)\ z \geq x \text{ and } \langle V, z\rangle \models \varphi\end{aligned}$$

covered by

$$\boxed{\varphi\ \mathit{release}\ \psi} \Rightarrow \boxed{\psi} \tag{12}$$

$$\boxed{\varphi\ \mathit{release}\ \psi}\Box \Rightarrow \boxed{\varphi}\Box \mid \Box\boxed{\varphi\ \mathit{release}\ \psi}\ . \tag{13}$$

We treat $\overline{\mathit{ntil}}$ and $\overline{\mathit{since}}$ similarly.

4.2 Paths

$\mathsf{G}\varphi$ is $\bot\ \mathit{release}\ \varphi$ and amounts to the infinite conjunction

$$\varphi \wedge \mathit{next}(\varphi) \wedge \mathit{next}(\mathit{next}(\varphi)) \wedge \cdots$$

which we shall analyze as follows. Given a language L, we say a language X is an *L-path* if $\emptyset \neq X \subseteq L$ and

(i) for all $s \in X$, there exists $s' \in X$ such that $s' \trianglerighteq \Box^+ s \Box^+$
(ii) for all $s, s' \in X$, there exists $s'' \in X$ such that $s'' \underline{\blacktriangleright} s$ and $s'' \underline{\blacktriangleright} s'$.

For example, for each $s \in L$, $\Box^* s \Box^*$ is a $\Box^* L \Box^*$-path (although a $\Box^* L \Box^*$-path need not be a subset of $\Box^* L \Box^+$ or $\Box^+ L \Box^*$). An L-path X is said to be *principal* if for some string s, $X \subseteq \Box^* s \Box^*$.

Definition. A set L of now_Φ-pointed strings *pathwise Φ-represents* φ if φ is equivalent to the disjunction over L-paths X of conjunctions $\bigwedge\{fmla(s) : s \in X\}$ in that

$$\langle V, x\rangle \models \varphi \iff (\exists L\text{-path } X)(\forall s \in X)\ \langle V, x\rangle \models fmla(s)$$

for all $V : P \to Pow(\mathbb{Z})$ and $x \in \mathbb{Z}$.

We can then prove an analog of Theorem 1 for pathwise (as opposed to stringwise) P-representations of formulas from a set P_∞ extending $P_\bullet$ with dual connectives $\bot, \mathit{release}, \overline{\mathit{ntil}}, \overline{\mathit{since}}$ and $\overline{\mathit{sinc}}$. $\mathcal{C}(\varphi)$ is revised to $\mathcal{D}(\varphi)$, bringing in the two forms (10) and (11) of non-contradiction,

$$\mathcal{D}(p) \stackrel{\text{def}}{=} [\blacktriangleright]\overline{\boxed{now}\Box^*\boxed{now} \mid \boxed{p, \overline{p}}} \qquad \text{for } p \in P$$

$$\mathcal{D}(\varphi) \stackrel{\text{def}}{=} [\blacktriangleright]\overline{\boxed{now}\Box^*\boxed{now} \mid \boxed{\bot}} \qquad \text{for } \varphi \in \{\top, \bot\}\ ,$$

treating $\wedge, \vee, \mathit{next}, \mathit{prev}, \mathit{until}, \mathit{since}, \mathit{ntil}$ and sinc as does $\mathcal{C}$

$$\mathcal{D}(\varphi \wedge \psi) \stackrel{\text{def}}{=} \mathcal{D}(\varphi) \cap \mathcal{D}(\psi) \cap (\boxed{\varphi \wedge \psi} \Rightarrow \boxed{\varphi, \psi})$$

$$\mathcal{D}(\varphi \vee \psi) \stackrel{\text{def}}{=} \mathcal{D}(\varphi) \cap \mathcal{D}(\psi) \cap (\boxed{\varphi \vee \psi} \Rightarrow \boxed{\varphi} \| \boxed{\psi})$$

$$\mathcal{D}(\mathit{next}(\varphi)) \stackrel{\text{def}}{=} \mathcal{D}(\varphi) \cap (\boxed{\mathit{next}(\varphi)} \stackrel{\mathsf{a}}{\Rightarrow} \boxed{\varphi})$$

etc, and building (12) and (13) into *release*

$$\begin{aligned}\mathcal{D}(\varphi\ \mathit{release}\ \psi) \stackrel{\text{def}}{=}\ & \mathcal{D}(\varphi) \cap \mathcal{D}(\psi) \cap \\ & (\boxed{\varphi\ \mathit{release}\ \psi} \Rightarrow \boxed{\psi})\ \cap \\ & (\boxed{\varphi\ \mathit{release}\ \psi}\Box \Rightarrow \boxed{\varphi}\Box \mid \Box\boxed{\varphi\ \mathit{release}\ \psi})\end{aligned}$$

and similarly for $\overline{\mathit{since}}$. Finally, we set

$$\hat{\mathcal{D}}(\varphi) \stackrel{\text{def}}{=} \mathcal{D}(\varphi) \cap \langle\blacktriangleright\rangle\boxed{now, \varphi}$$

and refrain from the unpadding in Theorem 1.

Theorem 2. *Every $\varphi \in P_\infty$ is pathwise P-represented by the regular language $\{\rho(s) : s \in \hat{\mathcal{D}}(\varphi)_{\trianglerighteq}\}$.*

5 Conclusion

The main results of the preceding account of temporal propositions as regular languages are Theorems 1 and 2 (from §§3.2 and 4.2). The theorems essentially implement well-known tableau constructions for Linear Temporal Logic

[3] through finite-state methods. As we shall see next, these methods extend to constructs in Hybrid Logic [10]. Beyond any application to a particular formal system, the methods feature notions (such as bounded entailment $\vdash_C$ defined in §2.2) that are part of a tool-kit for an approach to natural language semantics representing events as strings so that entailments can be read directly off the event representations. We close by outlining an approach based on these methods to changes over time against an inertial background [11].

5.1 Hybrid Logic

A basic notion in Hybrid Logic is that of a *nominal*, the collection of which we shall assume form a designated subset $P_0 \subseteq P$ of atomic propositions that a valuation V is required to map to singleton sets

$$(\forall n \in P_0)\ \ V(n) \text{ has cardinality } 1\ .$$

The uniqueness requirement on nominals n is built into the language

$$[\underline{\blacktriangleright}]\ \overline{\boxed{n}\Box^*\boxed{n}}$$

which we may assume is part of the constraints for $n \in P_0$. For arbitrary languages L and L' over Σ, let us write $L \overset{\blacktriangleright}{\rightarrow} L'$ for the set of strings that weakly subsume L' whenever they weakly subsume L

$$L \overset{\blacktriangleright}{\rightarrow} L' \overset{\text{def}}{=} \{s \in \Sigma^* \ :\ \text{if } s\underline{\blacktriangleright}L \text{ then } s\underline{\blacktriangleright}L'\}\ .$$

The special case of $L' = \emptyset$ reduces to $[\underline{\blacktriangleright}]\ \overline{L}$

$$[\underline{\blacktriangleright}]\ \overline{L} = L \overset{\blacktriangleright}{\rightarrow} \emptyset\ .$$

The operation $\overset{\blacktriangleright}{\rightarrow}$ preserves regularity, as

$$\begin{aligned} L \overset{\blacktriangleright}{\rightarrow} L' &= \overline{\langle\underline{\blacktriangleright}\rangle L \cap \overline{\langle\underline{\blacktriangleright}\rangle L'}} \\ &= [\{(s,s) : s\underline{\blacktriangleright}L\}]\langle\underline{\blacktriangleright}\rangle L'\ . \end{aligned}$$

Forming $L \overset{\blacktriangleright}{\rightarrow} L'$ with $L' \neq \emptyset$ pays off when analyzing a couple of constructs, @ and E, in Hybrid Logic. These constructs allow us to say of a temporal proposition φ that it holds at a nominal n

$$\langle V, x\rangle \models @_n\varphi \iff^{\text{def}} \langle V, n_V\rangle \models \varphi \quad \text{where } V(n) = \{n_V\} \tag{14}$$

or that it holds somewhere

$$\langle V, x\rangle \models \mathsf{E}\varphi \iff^{\text{def}} (\exists y)\ \langle V, y\rangle \models \varphi\ . \tag{15}$$

We can capture (14) as

$$\boxed{@_n\varphi} \overset{\blacktriangleright}{\rightarrow} \boxed{n, \varphi}$$

and (15) as

$$\boxed{\mathsf{E}\varphi} \xrightarrow{\blacktriangleright} \boxed{\varphi}\ .$$

To define negation via De Morgan duals, we set

$$\overline{@} = @$$
$$\overline{\mathsf{E}} = \mathsf{A}$$

and associate with $\mathsf{A}\varphi$ the constraints

$$\Box^+\boxed{\mathsf{A}\varphi} \Rightarrow \boxed{\varphi}\Box^+$$
$$\boxed{\mathsf{A}\varphi} \Rightarrow \boxed{\varphi}$$
$$\boxed{\mathsf{A}\varphi}\Box^+ \Rightarrow \Box^+\boxed{\varphi}$$

supporting a reading of $\mathsf{A}\varphi$ as "at all times (the past, the present and the future), φ."

Another construct from Hybrid Logic is the binder $\downarrow$ that we will assume combines a nominal $n \in P_0$ with a temporal formula φ in which '$\downarrow n$' does *not* occur. The resulting formula $\downarrow n.\varphi$ is then interpreted according to

$$\langle V, x\rangle \models \downarrow n.\varphi \stackrel{\text{def}}{\iff} \langle V_{x/n}, x\rangle \models \varphi$$

where $V_{x/n}$ is V except that it maps the nominal n to $\{x\}$. The corresponding constraint is

$$\boxed{\downarrow n.\varphi} \Rightarrow \boxed{n, \varphi}$$

(with the proviso that '$\downarrow n$' does not occur in φ).

5.2 Non-monotonic inertial reasoning

Finally, consider a temporal formula φ that, in the absence of a force against it, persists over time. A simple way of formulating this idea is to introduce a temporal formula $\mathsf{f}\overline{\varphi}$ intuitively saying that "a force is applied to make $\overline{\varphi}$ true (at the next step)" so that the constraint

$$\boxed{\varphi}\Box \Rightarrow \Box\boxed{\varphi} \mid \boxed{\mathsf{f}\overline{\varphi}}\Box \tag{16}$$

can be read as: φ persists (to the next step) unless a force is applied against it. Turning the force around to one $\mathsf{f}\varphi$ *for* (rather than against) φ, we obtain the backward form of persistence

$$\Box\boxed{\varphi} \Rightarrow \boxed{\varphi}\Box \mid \boxed{\mathsf{f}\varphi}\Box \tag{17}$$

making φ persist backward unless it was previously forced [9]. Together, (16) and (17) imply "no change without force." Distinguishing $\mathsf{f}\varphi$ from $\mathsf{f}\overline{\varphi}$ allows us to formulate the constraint

$$\boxed{\mathsf{f}\varphi}\Box \Rightarrow \Box\boxed{\varphi} \mid \boxed{\mathsf{f}\overline{\varphi}}\Box \tag{18}$$

saying that an unopposed force for φ brings φ about at the next step. The non-determinism expressed in the righthand sides of (16), (17) and (18) by choice $\mid$ opens the door to non-monotonicity as soon as we apply bias to choosing between the opposite sides of $\mid$. For instance, the assumption

(†) no force is applied unless it is explicitly mentioned

boosts the inference

$$\boxed{\varphi}\Box \vdash_{(16)} \Box\boxed{\varphi} \mid \boxed{\mathsf{f}\overline{\varphi}}\Box$$

to:

(‡) from $\boxed{\varphi}\Box$, infer $\Box\boxed{\varphi}$

(as no force is mentioned in $\boxed{\varphi}\Box$). The inference (‡) is soft inasmuch as the principle (†) licensing it is. (‡) is non-monotonic in that we lose the conclusion $\Box\boxed{\varphi}$ if we enrich the premise $\boxed{\varphi}\Box$ to $\boxed{\varphi, \mathsf{f}\overline{\varphi}}\Box$ (which subsumes $\boxed{\varphi}\Box$).

More precisely, recall that

$$L \vdash_C L' \iff C \cap \langle\underline{\blacktriangleright}\rangle L \ \underline{\blacktriangleright}\ L' \ . \tag{19}$$

If in (19) we were to refine the Peirce product

$$\langle\underline{\blacktriangleright}\rangle L = L_\Box \ \& \ \Sigma^*$$

(where $L_\Box$ is $\Box^* unpad(L)\Box^*$) by &-superposing $L_\Box$ not with Σ^* but with a sublanguage H such as

$$Pow(\Phi - \{\mathsf{f}\varphi, \mathsf{f}\overline{\varphi}, \ldots\})^*$$

then there would be more languages L' such that

$$C \cap (L_\Box \ \& \ H) \ \underline{\blacktriangleright}\ L' \tag{20}$$

than such that $L \vdash_C L'$. As far as computability is concerned, the important point about (20) is that it is as much an inclusion between regular languages as $L \vdash_C L'$ is. What (20) offers is a handle H on what to &-superpose with $L_\Box$ before intersecting it with the constraints C to see what is weakly subsumed. Under (20), there are two distinct ways to enrich $L_\Box$: by intersection with hard constraints C and by superposition with permissible hypotheses H. The non-monotonicity in (‡) above can be traced to a choice in (†) of H short of the full space Σ^* of possibilities entertained in $\vdash_C$. Bias is injected into the choice

$$\Box\boxed{\varphi} \mid \boxed{\mathsf{f}\overline{\varphi}}\Box$$

by including the left side $\Box\boxed{\varphi}$ in H, while excluding the right side $\boxed{\mathsf{f}\overline{\varphi}}\Box$ from H. Equally, we could pick an H' that throws *out* $\Box\boxed{\varphi}$ and lets in $\boxed{\mathsf{f}\overline{\varphi}}\Box$ to *explain* the failure of φ to persist in $\boxed{\varphi}\Box$.

References

1. Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI Publications, Stanford (2003)
2. Emerson, E.A.: Temporal and modal logic. In Leeuwen, J.v., ed.: Handbook of Theoretical Computer Science. Volume B: Formal Methods and Semantics. MIT Press (1992) 995–1072
3. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
4. Fernando, T.: A finite-state approach to events in natural language semantics. Journal of Logic and Computation **14**(1) (2004) 79–92
5. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Computer Aided Verification. Springer LNCS 1855 (2000) 403–418
6. Brink, C., Britz, K., Schmidt, R.: Peirce algebras. Formal Aspects of Computing **6**(3) (1994) 339–358
7. Harel, D.: Dynamic logic. In Gabbay, D., Guenthner, F., eds.: Handbook of Philosophical Logic. Volume 2. Reidel, Dordrecht (1984) 497–604
8. Karttunen, L.: www.stanford.edu/~laurik/fsmbook/examples/Yale Shooting.html (2005)
9. Fernando, T.: Finite-state temporal projection. In: Proc. 11th International Conference on Implementation and Application of Automata. Springer LNCS 4094 (2006) 230–241
10. Areces, C., ten Cate, B.: Hybrid logics. In Blackburn, P., Wolter, F., van Benthem, J., eds.: Handbook of Modal Logics. X (2005) (In Preparation).
11. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, M., Michie, D., eds.: Machine Intelligence 4. Edinburgh University Press (1969) 463–502

Phrase-based finite state models *

Jorge González and Francisco Casacuberta

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Abstract. In the last years, statistical machine translation has already demonstrated its usefulness within a wide variety of translation applications. In this line, phrase-based alignment models have become the reference to follow in order to build competitive systems. Finite state models are always an interesting framework because there are well-known efficient algorithms for their representation and manipulation. This document is a contribution to the evolution of finite state models towards a phrase-based approach. The inference of stochastic transducers that are based on bilingual phrases is carefully analysed from a finite state point of view. Indeed, the algorithmic phenomena that have to be taken into account in order to deal with such phrase-based finite state models when in decoding time are also in-depth detailed.

1 Introduction

Machine Translation (MT) is an emerging area of research in computational linguistics which investigates the use of computer software to translate text or speech from one natural language to another. The goal of MT is very ambitious because it would allow for a reduction of the linguistic barriers which all the people have been ever involved with.

Statistical machine translation represents an interesting framework because the translation software being developed is language-independent, that is, different MT systems are built if different parallel training corpora are supplied.

Given a source sentence $\mathbf{s} = \mathbf{s}_1 \dots \mathbf{s}_J$, the goal of statistical machine translation is to find a target sentence $\hat{\mathbf{t}} = \mathbf{t}_1 \dots \mathbf{t}_{\hat{I}}$, among all possible target strings $\mathbf{t}$, that maximises the posterior probability, according to a source-channel model:

$$\hat{\mathbf{t}} = \operatorname{argmax}_{\mathbf{t}} \Pr(\mathbf{t}|\mathbf{s}) \tag{1}$$

Source-channel models are often applied the Bayes rule [1] to break them down into two different statistical models: a translation model to learn translations, and a language model, to score the quality of the proposed hypotheses [2, 3]:

$$\hat{\mathbf{t}} = \operatorname{argmax}_{\mathbf{t}} \Pr(\mathbf{s}|\mathbf{t}) \cdot \Pr(\mathbf{t}) \tag{2}$$

* This work is supported by the EC (FEDER) and the Spanish MEC under grant TIN2006-15694-C02-01.

The conditional probability $\Pr(\mathbf{t}|\mathbf{s})$ can also be approximated by a joint probability distribution $\Pr(\mathbf{s}, \mathbf{t})$ in order to be modelled by means of stochastic finite state transducers [4, 5]:

$$\hat{\mathbf{t}} = \operatorname{argmax}_{\mathbf{t}} \Pr(\mathbf{s}, \mathbf{t}) \tag{3}$$

These models can integrate the probabilistic information that *state-of-the-art* phrase-based models [6–9] are used to explicitly separate into two distributions, that is, a target language model and a phrase translation dictionary.

This paper presents a natural evolution for finite state models in order to be based on bilingual phrases. Training and decoding algorithms are conveniently adapted to deal with such phrase-based finite state models. The main contributions are reflected on the translation results, which are clearly favourable to these phrase-based models, with respect to the original word-based approaches.

The organization of this document is as follows: next section presents a review of finite state models; sections 3 and 4 deal with, respectively, word-based and phrase-based finite state models; the experimental setup and results are described in section 5; and, finally, conclusions are summed up at the last section.

2 Finite state models

A weighted finite-state automaton is a tuple $\mathcal{A} = (\Gamma, Q, i, f, P)$, where Γ is an alphabet of symbols, Q is a finite set of states, functions $i : Q \to \mathbb{R}^+$ and $f : Q \to \mathbb{R}^+$ give a weight to the possibility of each state to be, respectively, initial and final, and parcial function $P : Q \times \{\Gamma \cup \lambda\} \times Q \to \mathbb{R}^+$ defines a set of transitions between pairs of states in such a way that each transition is labelled with a symbol from Γ (or the empty string λ), and is assigned a weight. An example of a weighted finite-state automaton can be observed in figure 1.

A weighted finite-state transducer [10] is defined similarly to a weighted finite-state automaton, with the difference that transitions between states are labelled with pairs of symbols that belong to the cartesian product of two different (input and output) alphabets, $(\Sigma \cup \{\lambda\}) \times (\Delta \cup \{\lambda\})$.

When weights are probabilities, the range of functions i, f, and P is constrained to $[0, 1]$. Moreover, probabilistic models have to respect the *consistency* property in order to define a distribution of probabilities on the free monoid. In that case they are called stochastic finite-state models. Consistent probability distributions can be obtained by requiring a series of local constraints, that is:

- $\sum i(q) = 1$
- $\forall q \in Q : \sum P(q, u, q') + f(q) = 1$

Then, given some input/output strings $\mathbf{s}$ and $\mathbf{t}$, a stochastic finite-state transducer is able to associate them a joint probability $\Pr(\mathbf{s}, \mathbf{t})$.

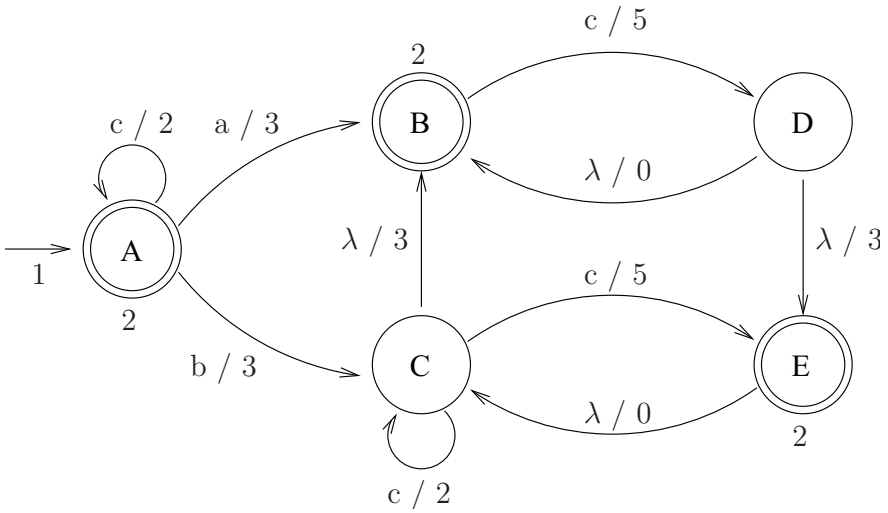


Fig. 1. A weighted finite-state automaton

2.1 Inference of stochastic transducers

The GIATI paradigm [11] has been revealed as an interesting approach to infer stochastic finite-state transducers through the modelling of languages. Rather than learning translations, GIATI first converts every pair of parallel sentences from the training corpus into only one string in order to, afterwards, infer a language model from.

More concretely, given a parallel corpus consisting of a finite sample C of string pairs: first, each training pair $(\bar{x}, \bar{y}) \in \Sigma^\star \times \Delta^\star$ is transformed into a string $\bar{z} \in \Gamma^\star$ from an extended alphabet, yielding a string corpus S; then, a stochastic finite-state automaton $\mathcal{A}$ is inferred from S; finally, transition labels in $\mathcal{A}$ are turned back into pairs of strings of source/target symbols in $\Sigma^\star \times \Delta^\star$, thus converting the automaton $\mathcal{A}$ into a transducer $\mathcal{T}$.

The first transformation is modelled by some labelling function $\mathcal{L} : \Sigma^\star \times \Delta^\star \rightarrow \Gamma^\star$, whereas the last transformation is defined by an inverse labelling function $\Lambda(\cdot)$, such that $\Lambda(\mathcal{L}(C)) = C$. Building a corpus of extended symbols from the original bilingual corpus allows for the use of many useful algorithms for learning stochastic finite-state automata (or equivalent models) that have been proposed in the literature about grammatical inference.

Every extended symbol from Γ has to condense somehow the meaningful relationship that exists between the words in the input and output sentences. Discovering these relations is a problem that has been thoroughly studied in statistical machine translation and has well-established techniques for dealing with it. The concept of statistical alignment [1] formalises this problem. An alignment is a mapping between words from a source sentence and words from a target sentence. Whether this function is constrained to a one-to-one, a one-to-many or a many-to-many correspondence depends on the particular assumptions that we

make. Constraining the alignment function simplifies the learning procedure but causes the model to lessen its expressive power. The available algorithms try to find a trade-off between complexity and expressiveness.

2.2 The search problem

Equation 3 expresses the MT problem in terms of a finite state model that is able to compute the expression $\Pr(\mathbf{s}, \mathbf{t})$. Given that only the input sentence is known, the model has to be parsed, taking into account all possible $\mathbf{t}$ that are compatible with $\mathbf{s}$. The best output hypothesis $\hat{\mathbf{t}}$ would be that one which corresponds to a path through the transduction model that, with the highest probability, accepts the input sequence as part of the input language of the transducer.

Although the navigation through the model is constrained by the input sentence, the search space can be extremely large. As a consequence, only the most scored partial hypotheses are being considered as possible candidates to become the solution. This search process is very efficiently carried out by the well known Viterbi algorithm [12].

3 Word-based finite state models

As it has been already mentioned, the inference of transducers will be done through the transformation of the bilingual training corpus into a corpus of strings, which a language model will be inferred from. This transformation will be based on the alignment function defined between every pair of bilingual sentences. According to the alignment degree, these transducers could be classified as word-based or phrase-based finite state models.

One-to-one and one-to-many alignment functions would produce word-based models, whereas many-to-many correspondences would bring to phrase-based models.

On the one hand, one-to-one models do not seem a very appropriate approach since they would require that source-target aligned sentences had exactly the same number of words. On the other hand, one-to-many alignment models have been a reference in statistical machine translation until the phrase-based tendencies took place at the research community. Word-based models constrain alignments so that one target word has to be aligned to only one source word.

The conversion of every pair of parallel sentences into an extended symbol string follows this algorithm:

```
for i = 1, j = 1, 2, ... J
      throw s[j]
      while ((i <= I) && (alignments[i] <= j))
          add t[i]
          i++;
while (i <= I)
      add t[i]
      i++;
```

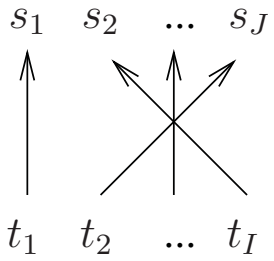


Fig. 2. An alignment situation

which means that compound symbols are left-to-right generated (`throw`), and where a target word t_i is merged (`add`) with its corresponding source word s_{a_i} iff their alignment $t_i \rightarrow s_{a_i}$ does not cross over any other alignment that has not been explored yet. If this is not possible, then the appearance of t_i is delayed until j reaches, then attached to, the last source word that is implied within the group of alignment crossing. Spurious source and target words are placed at their right position, given that a monotonous word order is always demanded. This procedure ensures that every extended symbol is composed of one and only one source symbol, optionally followed by an arbitrary number of target symbols. For example, the alignment in figure 2 would cause the string "$s_1 t_1$, s_2, s_3, ..., $s_J t_2 t_3 \ldots t_I$" to be produced. If a more detailed description about the labelling function is preferred, see [11].

A smoothed n-gram model may be inferred from the string corpus previously generated. Such a model can be expressed in terms of a stochastic finite-state automaton [13]. Figure 3 shows a general scheme for the representation of n-gram models through finite state machines.

No-backoff transitions jump from states in a determined layer to the one immediately above, thus increasing the history levels. Once the top level has been reached, n-gram transitions allow for movements inside this layer, from state to state, updating the history to the last $n-1$ seen events. Backoff transitions to lower history levels are taken if no way is found from a specific state for a given symbol $\mathbf{s}_j$. If the lowest level is reached and no unigram transition is found for $\mathbf{s}_j$, then a transition to the <unk> state is fired, thus considering $\mathbf{s}_j$ as an unknown word. There is only one initial state, which is denoted as <s>, and it is placed at the history level 1.

Since every unigram, bigram, etc., is represented as a transition consuming their last symbol, and given that all these extended symbols are composed of exactly one source word, the inverse labelling function can be straight-forwardly applied. This way, transition labels are turned back into pairs of source/target words to become a transducer.

Again, since every consuming transition implies that only one source symbol needs to be parsed, the beam-search Viterbi algorithm can be appropriately employed for decoding purposes.

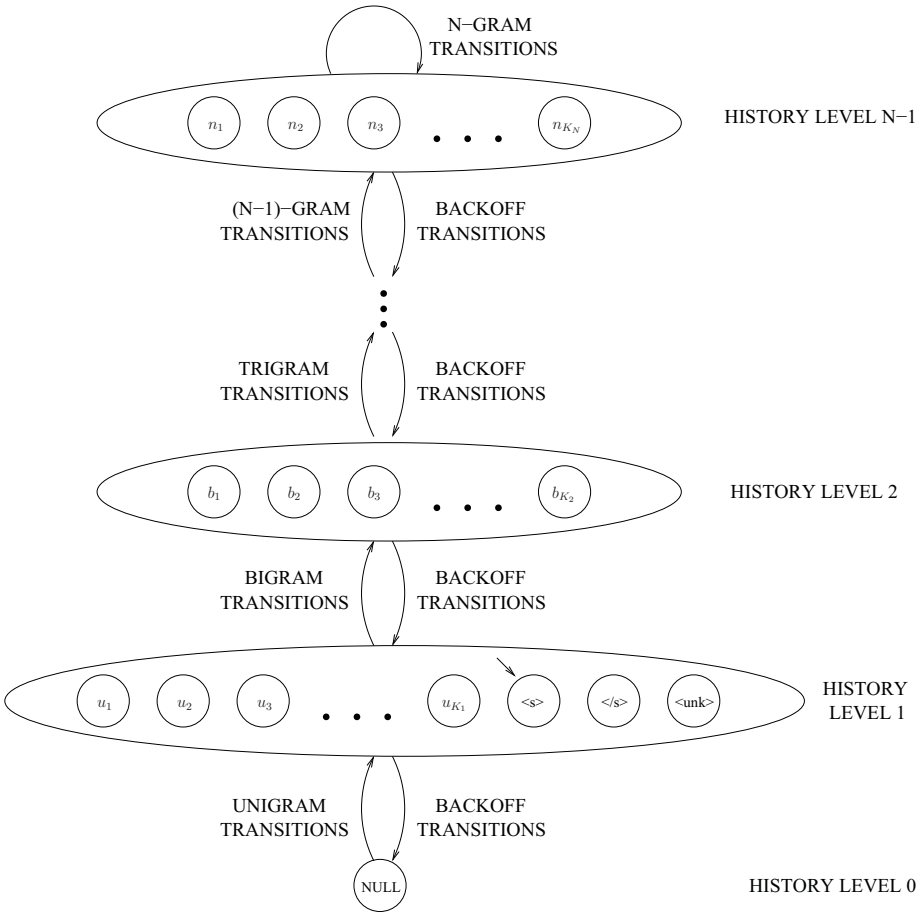


Fig. 3. A finite-state n-gram model

4 Phrase-based finite state models

Some recent researching lines are trying to merge the phrase-based methodology within a finite state framework [14]. There, a generative translation process, which is composed of several transduction models, is applied. Each constituent distribution of the model, including some well-known aspects in SMT, such as phrase reordering or spurious word insertion, is implemented as a weighted finite state transducer. The GIATI paradigm, however, tries to merge all these operations into only one transduction model.

Phrase-based finite state models come from the concept of monotonous bilingual segmentation, where it is assumed that only segments of contiguous words are considered, that every pair of source/target sentences is split up into the very same number of segments, and that they are one-to-one monotonously aligned.

On this occasion, extended symbol strings would be composed of their corresponding sequences of bilingual segments.

A bilingual segmentation of the training corpus can be approximated through a phrase-based statistical machine translation approach. In general terms, a statistical phrase-based model consist of a stochastic phrase translation table. From this table, those phrase pairs that best match a parallel training sample can be selected to approximate a bilingual segmentation. Such a phrase selection can be monotonously generated by translating the source-training sentences with that phrase-based model, since decoding implies looking for the best segmentation.

Again, a smoothed n-gram model can be inferred from the extended symbol corpus. Nevertheless, last step of GIATI cannot be applied as directly as word-based models do. As figure 4 shows, no-backoff transitions are labelled with a many-to-many extended symbol and they are assigned only one probability.

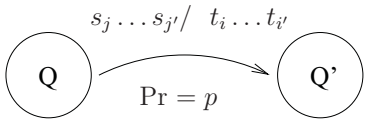


Fig. 4. Phrase-based automata transitions

If a transition label only contains one source symbol, the transformation is the same as for word-based models. However, the inverse labelling algorithm needs to divide all transitions including more than one source symbol.

These transitions are divided by the length of the source segment, putting only one source symbol on every resulting transition. The output segments are delayed to their last transition, which is reaching Q', thus forcing the previous ones to produce the empty string λ. Finally, probabilities are placed at their first transition, leaving 1-probability to the others. Figure 5 shows how this algorithm works.

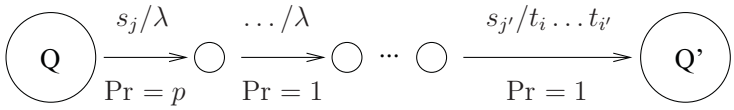


Fig. 5. Phrase-based transducer transitions

Intermediate states are artificially created on-the-fly and do not belong to the original automaton model. They are non-final states that should be parsed through until a real state is being reached, i.e. Q' in figure 5.

Actually, these transition sequences have to be seen as a unique transition: the one corresponding to $\Pr(s_j \ldots s_{j'}, t_i \ldots t_{i'}|\mathrm{Q})$, that is, the phrase translation probability after a given history Q.

When in decoding time, the search algorithm takes into account such a special situation, thus trying to follow all the paths coming from a determinate state which are compatible with the input string that has not been analysed yet. This parsing behaviour, i.e. the non-stop at intermediate states thing, can be easily implemented adding some extra conditions to the Viterbi algorithm and including more information within the trellis structure that is commonly employed.

Yet another change to the search algorithm is needed because of the phrase-based nature of the proposed approach. Given a starting state Q, a successful path from Q to any Q' would take into account a phrase-based n-gram event, that is, a phrase pair $(s_j \dots s_{j'}, t_i \dots t_{i'})$ that was seen during training after a given history Q. However, if only these paths are explored, the model may not be as effective as it would be able to be.

As a result, backoff transitions must be always allowed in order to cover all compatible phrases in the model, not only the ones which have been seen after a given history, but from lower levels as well. One more constraint has to be included into the parsing algorithm: any directly reaching state Q' is unable to be reached through a path that implies a backoff transition between Q and Q'. Backoff transitions are followed in order to consider all the possible segmentations of the input sentence.

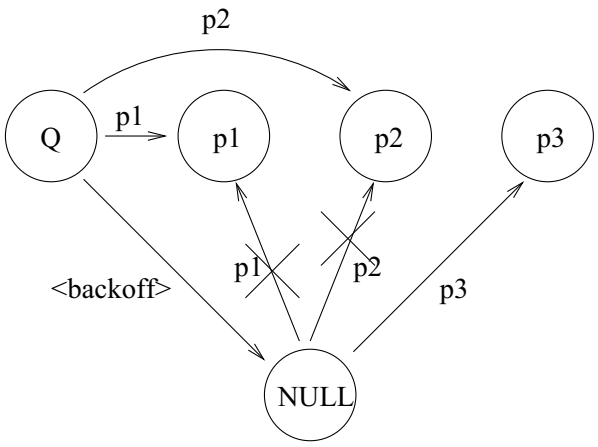


Fig. 6. Compatible transitions for a phrase-based bigram model

Figure 6 shows a parsing example over a finite-state representation of a bigram model. Given a reaching state Q, phrases *p1*, *p2* and *p3* are all compatible with the portion of the input sentence that has not been parsed yet. However, the bigram (Q, *p3*) did not occur throughout the training corpus, therefore there is no a direct transition from Q to *p3*. A backoff transition enables the access to *p3* because the bigram (Q, *p3*) turns into a unigram event that is actually inside

the model. Again, unigram transitions to *p1* and *p2* must be ignored because their corresponding bigram events were successfully found one level above.

5 Experiments

This approach has been applied to the EuroParl corpus, that is, the benchmark corpus of the NAACL 2006 and 2007 shared tasks of the Workshops on Machine Translation of the Association for Computational Linguistics.

The EuroParl corpus is built on the proceedings of the European Parliament, which are published on its web and are freely available. Because of its nature, this corpus has a large variability and complexity, since the translations into the different official languages are performed by groups of human translators. The fact that not all translators agree in their translating criteria implies that a given source sentence can be translated in various different ways throughout the corpus.

Since the proceedings are not available in every language as a whole, a different subset of the corpus is extracted for every different language pair, thus evolving into somewhat different corpora for each pair.

5.1 Corpus characteristics

Several shared tasks involving, among others, French, English and Spanish languages, were proposed during the NAACL 2006 and 2007 Workshops on Machine Translation.

French→English and Spanish↔English experiments were carried out over the 2006 EuroParl benchmark corpus, whereas only Spanish↔English translation was tackled from the 2007 data.

The characteristics of these corpora can be seen in Table 1.

Table 1. *Characteristics of the EuroParl corpora*

		2006				2007	
		Fr	**En**	**Sp**	**En**	**Sp**	**En**
Training	Sentences	688031		730740		964791	
	Run. words	15.6 M	13.8 M	15.7 M	15.2 M	20.9 M	20.3 M
	Vocabulary	80348	61626	102216	64070	113026	81754
Dev-Test	Sentences	2000		2000		2000	
	Run. words	66200	57951	60332	57951	60243	58059

5.2 System evaluation

We evaluated the quality of a statistical machine translation system by using the following evaluation measures:

BLEU *(Bilingual Evaluation Understudy) score*: This indicator computes the precision of unigrams, bigrams, trigrams, and tetragrams with respect to a set of reference translations, with a penalty for too short sentences [15]. BLEU measures accuracy, not error rate.

WER *(Word Error Rate)*: The WER criterion calculates the minimum number of editions (substitutions, insertions or deletions) needed to convert the system hypothesis into the sentence considered ground truth. Because of its nature, this measure is considered to be a pessimistic indicator.

5.3 Translation results

On the one hand, word-based finite state models are based on statistical alignments, which were obtained from the application of the public available tool GIZA++ [16] to the corresponding training corpora. On the other hand, phrase-based finite state models are required to operate with a bilingual segmentation of the training corpus. These bilingual segmentations were provided by means of a statistical phrase-based machine translation system such as Pharaoh [17].

Table 2. *Translation results over the EuroParl corpora*

	Word-based		**Phrase-based**	
Corpus	BLEU	WER	BLEU	WER
2006 fr→en	20.0	64.1	28.0	61.9
2006 sp→en	20.6	63.9	27.6	61.6
2006 en→sp	16.8	67.9	26.4	62.3
2007 sp→en	21.9	62.9	28.0	59.6
2007 en→sp	20.1	64.9	25.3	60.8

From the translation results that are presented in Table 2, it can be concluded that phrase-based finite-state models clearly outperform the models that are strictly based on words, within the context of such a EuroParl translation task. Phrase-based finite state models are almost achieving a relative improvement of 35% of BLEU over the language pairs and translation directions that have been tested on.

6 Conclusions and further work

Phrase-based alignment models have become the predominant technology in statistical machine translation. However, finite state models are always an interesting approach to be taken into account in translation matters because they present some advantages with respect to the use of pure source-channel models.

The idea of using phrase-based (rather than word-based) dictionaries can also be brought to a finite state framework. This paper has presented the implementation details that are needed to build a phrase-based finite state model from

a bilingual segmentation of the training corpus. Indeed, the algorithmic phenomena that have to be taken into account in order to deal with such phrase-based finite state models when in decoding time have also been in-depth described.

Experiments concerning several language pairs from the EuroParl corpus have been carried out. Translation results from phrase-based finite-state models are clearly outperforming the ones from a word-based finite state framework. An approximate relative improvement of 35% over the BLEU metric is observed for most of the language pairs and translation directions that have been tested on.

Phrase-based finite state models come from the concept of monotonous bilingual segmentation. The experiments reported here are based on a single bilingual segmentation per every pair of training sentences. That is, any other way of splitting a given pair to produce a different monotonous bilingual segmentation is therefore discarded. Learning from all the possible segmentations (rather than from the most likely one) that are compatible with a given alignment of a training pair will probably enrich the models, since the useful information that is extracted from the training data increases. This will be part of our future work.

References

1. Brown, P.F., Cocke, J., Pietra, S.D., Pietra, V.J.D., Jelinek, F., Lafferty, J.D., Mercer, R.L., Roossin, P.S.: A statistical approach to machine translation. Computational Linguistics **16**(2) (1990) 79–85
2. Brown, P.F., Pietra, S.D., Pietra, V.J.D., Mercer, R.L.: The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics **19**(2) (1993) 263–311
3. Ney, H., Nießen, S., Och, F.J., Tillmann, C., Sawaf, H., Vogel, S.: Algorithms for Statistical Translation of Spoken Language. IEEE Trans. on Speech and Audio Processing, Special Issue on Language Modeling and Dialogue Systems **8** (January 2000) 24–36
4. Casacuberta, F., Ney, H., Och, F.J., Vidal, E., Vilar, J.M., Barrachina, S., García-Varea, I., Llorens, D., Martínez, C., Molau, S.: Some approaches to statistical and finite-state speech-to-speech translation. Computer Speech & Language **18**(1) (2004) 25–47
5. Casacuberta, F., Vidal, E.: Machine translation with inferred stochastic finite-state transducers. Computational Linguistics **30**(2) (2004) 205–225
6. Tomás, J., Casacuberta, F.: Monotone statistical translation using word groups. In: Procs. of the Machine Translation Summit VIII. (2001) 357–361
7. Marcu, D., Wong, W.: A phrase-based, joint probability model for statistical machine translation. In: Procs. of the Conference on Empirical Methods in Natural Language Processing and Very Large Corpora EMNLP-02. (2002)
8. Zens, R., Och, F.J., Ney, H.: Phrase-based statistical machine translation. In Jarke, M., Koehler, J., Lakemeyer, G., eds.: KI. Volume 2479 of Lecture Notes in Computer Science., Springer (2002) 18–32
9. Zens, R., Ney, H.: Improvements in phrase-based statistical machine translation. In: HLT-NAACL. (2004) 257–264
10. Mohri, M., Pereira, F., Riley, M.: Weighted finite-state transducers in speech recognition. Computer Speech & Language **16**(1) (2002) 69–88

11. Casacuberta, F., Vidal, E., Picó, D.: Inference of finite-state transducers from regular languages. Pattern Recognition **38**(9) (2005) 1431–1443
12. Jelinek, F.: Statistical Methods for Speech Recognition. The MIT Press (January 1998)
13. Llorens Piñana, D.: Suavizado de autómatas y traductores finitos estocásticos. PhD Thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia (2000)
14. Kumar, S., Deng, Y., Byrne, W.: A weighted finite state transducer translation template model for statistical machine translation. Nat. Lang. Eng. **12**(1) (2006) 35–75
15. Papineni, K., Roukos, S., Ward, T., Zhu, W.: Bleu: a method for automatic evaluation of machine translation (2001)
16. Och, F.J., Ney, H.: A systematic comparison of various statistical alignment models. Computational Linguistics **29**(1) (2003) 19–51
17. Koehn, P.: Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In Frederking, R.E., Taylor, K., eds.: AMTA. Volume 3265 of Lecture Notes in Computer Science., Springer (2004) 115–124

ME-CSSR: an Extension of CSSR using Maximum Entropy Models

Muntsa Padró and Lluís Padró

TALP Research Center
Universitat Politècnica de Catalunya

Abstract. In this work an extension of CSSR algorithm using Maximum Entropy Models is introduced. Preliminary experiments to perform Named Entity Recognition with this new system are presented.

1 Introduction

The Causal State Splitting Reconstruction (CSSR) algorithm [1] infers the causal states of a process from data, building a deterministic automaton that is expected to capture the patterns of data. These data are sequences of symbols drawn from a discrete alphabet Σ. Consider, for example $\Sigma = \{M, m\}$ to represent capitalized words (M) and not capitalized words (m). A history x is defined as a suffix formed by alphabet symbols (i.e. Mmm, $MMmM$, etc). CSSR studies each possible history (up to a preestablished maximum length l_{max}), comparing them in terms of their future probability distributions $P(Z|x)$, where Z is a random variable taking any value in Σ. Two histories, x and y, are equivalent when $P(Z|x) = P(Z|y)$, i.e. when they have the same probability distribution for the future. The different future distributions build the equivalence classes, named *causal states*. CSSR iteratively builds these causal states. The algorithm performs the comparison between probability distributions performing a hypothesis test.

CSSR has been applied to different research areas. For example, it has been used to learn the patterns of physical systems in crystallography [2] and to anomaly detection in dynamical systems [3]. These systems use CSSR to capture patterns representing data that can be then used for different purposes.

This algorithm has been also used in the field of Natural Language Processing (NLP) to learn automata that can be afterwards used to tag new data in tasks such as Named Entity Recognition (NER) and Chunking [4]. The results obtained in those experiments show that this technique can provide state-of-the-art results in some NLP tasks. Given these results, the challenge is to improve them, developing systems rivalling best state-of-the-art systems. In this work, we propose an approach to combine CSSR with Maximum Entropy (ME) models in order to introduce more information into the system and study if the performance improves. For these preliminary experiments we focus on NER task.

To apply CSSR to NER and to other NLP tasks, it is necessary to encode each word as a symbol of the alphabet Σ. This symbol has to take into account the relevant features for the task as well as the hidden information about whether

the word belongs to a named entity (NE)[1]. For example, if the only features taken into account are if the word is capitalized or not (M or m), the alphabet will be the combination of each feature with the corresponding "B-I-O" tag: $\Sigma = \{M_B, M_I, M_O, m_B, m_I, m_O\}$

This approach is rather limited, since all information we want to take into account has to be encoded in the alphabet. Furthermore, the amount of necessary data to build a correct automaton grows exponentially with the alphabet size. For that reason, a method to introduce more information about the words independently of the alphabet has been devised.

2 Introducing ME models into CSSR

The main idea of the proposed approach is based on generalizing the concept of history. Instead of considering histories as sequences of alphabet symbols corresponding to the last l_{max} words, we define histories as sets of relevant information about the last l_{max} words. Thus, histories can be encoded as collections of features of the words in a window of size l_{max}. In this way, causal states can still be defined as sets of histories with the same distribution for the future and can be calculated following the structure of CSSR.

This work uses ME models [6] to compute the probability distribution of the future. The classes of ME models are the alphabet symbols used with CSSR, and they define the possible transitions of each state in the automaton. The relevant information associated to each word is encoded as different features, and ME models are used to compute the probability distribution of the next symbol given the active features. If with CSSR the probability to be computed had the form of $P(m_B|M_B M_I m_O)$ now the probability will be computed as $P(m_B|h)$ where h is a history including relevant features of last words.

We present three different approaches to use this extended concept of history and ME models in combination with CSSR:

1. **Plain ME:** Using the learned ME models, compute the probability of each word in test corpus of having the tag B, I or O (taking into account that there is a known part of the symbol, i.e. we know if it is M or m), and compute the best sequence of tags using the Viterbi algorithm. Note that this first approach doesn't use CSSR in any way, but it can be used as a baseline of ME models performance.
2. **ME-over-CSSR:** Use CSSR to learn an automaton as in [4], using a simple alphabet. The ME model is used only during the tagging task, and its predicted probabilities are combined with the transition probabilities learned by the automaton. This is a simple way to introduce more complicated features without changing CSSR algorithm.
3. **ME-CSSR:** An extended version of CSSR algorithm that defines histories as sets of features instead of simple symbol suffixes. In this way all the

[1] This information is encoded using "B-I-O" approach [5]: B for words at NE beginning, I for words internal to a NE, and O for words outside a NE

information encoded in the features is taken into account when building the automaton and the automaton is expected to better capture the patterns of sequences since it has more information.

2.1 Experiments and Results

Different experiments with these three different methods were performed. These are preliminary experiments as the system is still under development.

The used alphabet and data are the same used in [4]. The alphabet has 5 symbols combining different orthographical and syntactic information, which combined with the B-I-O tags lead to a 15 symbol alphabet. The data are those of CoNLL-2000 shared task [7].

The experiments presented in this work were performed with two different feature sets. These sets include few and simple features, and will be extended in further work. First feature set (FS_1) takes into account just the alphabet symbol and the PoS tag. The second one (FS_2) includes the same features than FS_1 plus 4 more boolean features: capitalized word, word containing numbers, all letters capitalized, and auxiliary word (words that often appear inside NEs). Note that the feature corresponding to the alphabet symbol includes the hidden B-I-O information which is not available in the test corpus. When performing tagging step this feature is set to the symbol assumed by the Viterbi algorithm in the currently analyzed path. All these features are taken into account for each word in a window of size l_{max} to the left of the current word. To maintain the idea of histories it is necessary to consider the same maximum length for all features which will be the length used by CSSR to learn the automaton. Both feature sets also include the known part of the symbol (i.e. m or M) and the PoS tag of the current word.

The implication of taking into account different lengths for different features, of introducing features of future words, and how to combine it with CSSR algorithm, will be studied in the future.

The experiments were conducted with both feature sets and with l_{max} from 2 to 4. Table 1 shows the best F_1 scores obtained. The results with $l_{max} = 4$ are not presented as they are far behind the other results, since the available training data is insufficient to learn reliable automata with this history length.

System	FS_1		FS_2	
	l=02	**l=03**	**l=02**	**l=03**
Plain ME	87.00	86.37	86.56	86.28
ME-over-CSSR	**88.51**	86.63	88.26	86.61
ME-CSSR	85.89	85.61	85.97	85.18

Table 1. Obtained F_1 results with different feature sets and different approaches

From these results it can be seen that the simple combination ME-over-CSSR leads to better results than using plain ME models with the Viterbi algorithm,

and that the proposed ME-CSSR method leads to worse results. The best result of using only CSSR reported in [4] is $F_1 = 88.96\%$ which is not significantly different (at 95% confidence degree) from the best result presented here. Also the figures show that increasing l_{max} or the number of features leads to a lose in performance, which is surprising specially in the case of using plain ME models. This can be due to the sparseness of data, or to using over-simplistic feature sets, and further research is required on this issue.

Another point requesting further study is the trade-off between the data-sparseness caused by the fact of viewing histories as feature sets. Since the richer feature set we use, the less occurrences we'll have of each particular history, the CSSR algorithm will have less evidence to accurately build the causal states. On the other hand, richer feature sets should produce better ME models, which can compensate this lack of evidence.

3 Conclusions and Further Work

An extension of CSSR using ME models has been presented. The best results obtained are similar to the ones obtained with CSSR without ME models, but the experiments are very preliminary and the used features very simple, so there is still room for improvement. We expect to attain better performance when introducing more complicated features into the system, as ME models estimate better the probability distributions when rich feature sets are taken into account.

While the ME-over-CSSR approach yields better results than using only plain ME models, the ME-CSSR proposal leads to worse results in the performed experiments. One reason for this can be that the hypothesis test to determine if two probability distributions are different is performed using χ^2 statistics, and this may not be adequate when dealing with histories containing many features, as the number of occurrences for each history will be low, and χ^2 test depends on the counts of seen events being a poor test if the counts are low. Additionally, since ME models provide conditional probability distributions, a test comparing distributions regardless of the counts behind would be much more appropriate.

In the future, experiments introducing more features into the combined systems will be performed, searching for better results of the approaches combining CSSR and ME models. Also, other hypothesis tests have to be checked to learn automata with ME-CSSR, as χ^2 seems not to be the most adequate.

References

1. Shalizi, C.R., Shalizi, K.L.: Blind construction of optimal nonlinear recursive predictors for discrete sequences. In: Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference. (2004)
2. Varn, D.P., Crutchfield, J.P.: From finite to infinite range order via annealing: The causal architecture of deformation faulting in annealed close-packed crystals. Physics Letters A **324** (2004) 299–307
3. Ray, A.: Symbolic dynamic analysis of complex systems for anomaly detection. Signal Process. **84**(7) (2004) 1115–1130

4. Padró, M., Padró, L.: Applying a finite automata acquisition algorithm to named entity recognition. In: Proceedings of FSMNLP'05, Helsinki (2005)
5. Ramshaw, L., Marcus, M.P.: Text chunking using transformation-based learning. In: Proceedings of the Third ACL Workshop on Very Large Corpora. (1995)
6. Berger, A., Pietra, S.D., Pietra, V.D.: A maximum entropy approach to natural language processing. Computational Linguistics **22** (1996) 39–71
7. Tjong Kim Sang, E.F., Buchholz, S.: Introduction to the conll-2000 shared task: Chunking. In: Proceedings of CoNLL-2000, Lisbon, Portugal (2000)

ExPRESS – Extraction Pattern Recognition Engine and Specification Suite

Jakub Piskorski

Joint Research Center of the European Commission
Web Mining and Intelligence Action
Institute for the Protection and Security of the Citizen
Via Fermi 1, 21027 Ispra (VA), Italy

Abstract. The emergence of information extraction (IE) oriented pattern engines has been observed during the last decade. Most of them exploit heavily finite-state devices. This paper introduces ExPRESS – a new extraction pattern engine, whose rules are regular expressions over flat feature structures. The underlying pattern language is a blend of two previously introduced IE oriented pattern formalisms, namely, JAPE, used in the widely known GATE system, and the unification-based XTDL formalism used in SProUT. A brief and technical overview of ExPRESS, its pattern language and the pool of its native linguistic components is given. Furthermore, the implementation of the grammar interpreter is addressed too.

1 Introduction

The task of information extraction (IE) is centered around extracting specific structured information from free-text documents. The classical IE tasks focus on detecting entities, identifying relations which hold among them, and extracting events. Typically, the major step in the process of retrieving the sought-after information consists of applying a cascade of so called extraction patterns. Recently, the emergence of IE-oriented pattern specification languages has been observed. These languages utilize various types of formalisms, ranging from character-level regular expressions to unification-based formalisms. Due to efficiency reasons, finite-state based pattern engines are the most prominent ones being used.

This paper introduces ExPRESS (Extraction Pattern Recognition Engine and Specification Suite) – a new extraction pattern engine, whose rules are regular expressions over flat feature structures, i.e., non-recursive feature structures, where features are string valued. The rule specification language is a blend of two previously introduced IE-oriented grammar formalisms, namely, JAPE [1] used in the widely known GATE platform and the unification-based formalism XTDL deployed in SProUT [2]. The main motivation beyond the development of ExPRESS comes from: (a) a need of an efficient pattern engine for extracting facts from vast amount of news articles collected on a daily basis from the web

by Europe Media Monitor[1] (EMM) system [3], and (b) due to efficiency problems encountered when using other freely available IE-oriented pattern engines, including the two aforementioned ones.

The rest of this paper is organized as follows. We start in section 2 with some basic definitions and notions used throughout this paper. Next, in section 3 a brief overview of the related work is given. Subsequently, in section 4 ExPRESS, its pattern specification language and its core native linguistic components are described. Efficiency issues in the context of compiling and processing the grammars are addressed in section 5. Section 6 gives technical details about implementation and provides some figures concerning the run-time behavior. We provide a concluding summary in section 7.

2 Basic Definitions and Notions

This section introduces the basic definitions and notions used in this paper. A *deterministic finite-state automaton* (DFSA) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite *set of states*, Σ is the *alphabet* of M, $\delta : Q \times \Sigma \to Q$ is the *transition function*, q_0 is the *initial state* and $F \subseteq Q$ is the *set of final states*. The transition function can be extended to $\delta^* : Q \times \Sigma^* \to Q \cup \{\perp\}$ by defining $\delta^*(q, \epsilon) = q$, $\delta^*(q, a) = \delta(q, a)$ if $\delta(q, a)$ is defined or $\delta^*(q, a) = \perp$ otherwise, and $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for $a \in \Sigma$ and $w \in \Sigma^*$. The *language accepted* by a DFSA M is defined as $L(M) = \{w \in \Sigma^* | \delta^*(q_0, w) \in F\}$. Languages accepted by finite-state automata are also called *regular*. The *union* and *concatenation* of two regular languages L_1 and L_2 is denoted as $L_1 \cup L_2$ and $L_1 \cdot L_2$ respectively. A *path* in a DFSA M is a sequence of triples $\langle (p_0, a_0, p_1), \ldots, (p_{k-1}, a_{k-1}, p_k) \rangle$, where $(p_{i-1}, a_{i-1}, p_i) \in Q \times \Sigma \times Q$ and $\delta(p_i, a_i) = p_{i+1}$ for $1 \leq i < k$. The string $a_0 a_1 \ldots a_k$ is the *label* of the path. Among all DFSAs recognizing the same language, there is always one which has the minimal number of states. We call such an automaton *minimal* (MDFSA). The definition of *nondeterministic finite-state automata* (NFSA) is analogous, with the difference that transition function is set-valued, i.e., more than one transition from a given state q labeled with a symbol $a \in \Sigma$ might exist.

Next, we define flat feature structures, which are frequently referred to in this paper. A type space is a triple $\Phi = (\Sigma_T, \Sigma_F, \Delta)$, where Σ_T is a finite *set of types*, Σ_F is a finite *set of features* and $\Delta : \Sigma_T \to 2^{\Sigma_F}$ is the total *type specification function*, i.e., Δ maps types to their features. We say that a feature $f \in \Sigma_F$ is *appropriate* for the type α if $f \in \Delta(\alpha)$, otherwise f is *inappropriate* for the type α. A *flat feature structure* (FFS) in the type space $\Phi = (\Sigma_T, \Sigma_F, \Delta)$ is a pair $s = (\alpha, val)$, where $\alpha \in \Sigma_T$ (α is a type), and $val : \Delta(\alpha) \to {\Sigma^+}_V \cup \{\top\}$ is a *feature-value mapping*, where Σ_V is a finite set of symbols. The symbol $\top$ is used to denote unspecified (undefined) feature values, i.e., $val_s(f) = \top$ means that the value of f is unspecified for s. We say, that two FFSs $s = (\alpha_s, val_s)$ and $t = (\alpha_t, val_t)$ *match* in the type space Φ if and only if: (a) $\alpha_s, \alpha_t \in \Sigma_T$,

[1] http://emm.jrc.it/overview.html

(b) $\alpha_s = \alpha_t$, and (c) $\forall f \in \Delta(\alpha_s) : val_s(f) = val_t(f)$ or $val_s(f) = \top$ or $val_t(f) = \top$. For the sake of simplicity, we denote a FFS $s = (\alpha, val)$ also as $[f_1 : v_1 \ldots f_k : v_k]_\alpha$, where $\forall 1 \leq i \leq k : f_i$ is appropriate for α and $v_i = val(f_i)$.

In this paper, we also refer to *typed feature structures* (TFS), which are related to record structures in programming languages and are widely used as a data structure for NLP. Their formalizations [4] include multiple inheritance and subtyping, which allow for terser descriptions.

3 Related Work

The idea of using regular expressions over more complex structures is not new and has been considered by several authors, e.g., [5] uses regular grammars with predicates over morphologically analyzed tokens. Furthermore, [6] introduces finite-state transducers with arbitrary predicates over symbols and discusses various operations on such finite-state devices. In particular, during the last decade, several high-level IE-oriented specification languages for creating patterns have been developed, e.g., [7] introduced CPSL designed as a language for specifying finite-state grammars over arbitrary annotations. The widely-known GATE platform, exploited heavily for development of IE components, comes with JAPE – Java Annotation Pattern Engine [1], which is similar in spirit to CPSL. A JAPE grammar consists of pattern-action rules. The left-hand side (LHS) of a rule is a regular expression over arbitrary atomic feature-value constraints, while the right-hand side (RHS) constitutes a so-called *annotation manipulation statement* which specifies the output structures to be produced once the pattern matches. Additionally, the RHS may call native code, which on the one hand provides a gateway to the outer world, but on the other hand makes pattern writing difficult for non-programmers.

A somewhat more declarative and linguistically-oriented pattern specification formalism called XTDL is used in SProUT [2], a lesser known IE framework. It can be seen as an amalgam of finite-state and unification-based grammar formalisms. In XTDL the LHS of a rule is a regular expression over typed feature structures (TFS) with functional operators and coreferences[2], and the RHS is a TFS, specifying the output production. Functional operators are primarily utilized for forming the slot values in the output structures and, secondly, they can act as Boolean-valued predicates, which allows for introducing complex constraints in the rules. The aforementioned features make XTDL more amenable formalism than JAPE since writing 'native code' is eliminated and coreferencing allows for terser descriptions.

Clearly, rich annotations on automata edges allow for compact descriptions, but standard finite-state optimization and processing methods are hardly applicable. Although, efficient processing techniques for both JAPE [8] and XTDL [9] have been developed, to the authors knowledge and experience processing even

[2] Coreferences express structural identity, create dynamic value assignments, and serve as means of data transfer from LHS to RHS of a pattern

moderate-size grammars with the aforementioned engines remains a bottle-neck.[3] In particular, processing XTDL patterns involves unification, a rather expensive operation.

Some other IE-oriented pattern languages are surveyed in [10], but since most of them are bound to a specific type of information and exhibit somewhat black-box character, we do not discuss them any further.

4 ExPRESS

4.1 Overview

ExPRESS is a pattern engine which allows for specifying and processing cascaded finite-state grammars, where grammar rules are regular expressions over feature structures. It has been mainly designed for tackling IE tasks. ExPRESS consists basically of a grammar parser and a cascaded-grammar interpreter. A cascaded grammar specification is divided into three parts: (a) *types declaration*, (b) a set of *grammar definitions* and (c) a *workflow specification*. The *types declaration* part is a list of all types and appropriate features for these types, which are used in the grammar(s). In the type declaration example in figure 1, three types are introduced, namely `person`, `person_group` and `violent_event`, where for each of them a list of appropriate features is specified.

```
person:=[NAME,FIRST_NAME,LAST_NAME,INITIAL,AMOUNT,SEX]
person_group:=[NAME,AMOUNT,QUANTIFIER]
violent_event:=[TYPE,METHOD,ACTOR,VICTIM]
```

Fig. 1. Type declaration in ExPRESS

A single *grammar definition* consists of two parts: a *grammar configuration* part and a *rule definition* part. In the configuration part, a list of arbitrary processing resources can be specified, which will be applied before the interpreter applies the grammar. These components provide the grammar interpreter with a stream of input flat feature structures represented as a list of disjunctions of FFSs. The list of available components and the task of integration of external components is addressed in section 4.3. Further, for each grammar a different search strategy can be chosen. Currently the following strategies are supported: (a) *longest-match*, (b) *all-matches*, and (c) *all-longest-matches* (*longest-match* strategy applied at each position in the input). Finally, the last item in the configuration part specifies the output production option. Three alternative options

[3] There are several implementations of JAPE. We did not test the recently developed Japec version [8], which is supposed to be 2-5 times faster than the original implementation.

are provided: (a) return only structures produced via grammar application, (b) additionally to (a) return also feature structures produced by other processing modules applied at the same level, (c) like (b), with the difference that only those feature structures produced by other processing modules are returned, which were not consumed by the application of the grammar. The simplified example in figure 2 gives an idea of the syntax of a single grammar. The rule specification format is described in detail in section 4.2.

```
SETTINGS:
 { MODULES: <Tokenizer>, <Morphology>, <Gazetteer>
   SEARCH_MODE: longest_match
   OUTPUT: grammar_only
 }
RULES:
 R1
 .
 .
 RN
```

Fig. 2. Syntax of a single grammar

Finally, the last part of the input to the parser, namely *workflow specification*, is a sequence of grammar names, which defines the order in which the grammars are applied by the interpreter. In addition, each grammar name, may be accompanied by a file, which specifies the priorities for the rules in the corresponding grammar. Thus, experimenting with different prioritization set-ups is more elegant than in JAPE, where priorities are encoded directly in the rules (XTDL also separates the prioritization settings from the grammars). If there are several rules which match and have the same priority, then all output structures are returned by the grammar interpreter, unless the output structures are identical. In the latter case only one instance is returned.

4.2 Rule Specification Language

This subsection focuses on the particularities of the rule specification formalism of ExPRESS, which is similar in spirit to JAPE, but also encompasses some features and syntax borrowed from XTDL. The LHS of a rule is a regular expression over flat feature structures (FFS), i.e., non-recursive TFS without coreferencing, where features are string-valued and unlike in XTDL types are not ordered in a hierarchy (see 2). On the LHS of a rule variables can be tailored to the string-valued attributes in order to facilitate information transport into the RHS, etc. Further, like in XTDL, functional operators are allowed on the RHSs for manipulating slot values and for establishing contact with the 'outer world'. They can also be deployed as boolean-valued predicates. There is

a predefined set of available functional operators, and new ones can be added by simply implementing an appropriate programming interface by the grammar developer. Finally, we adapted the JAPE's feature of associating patterns on the LHSs with multiple actions (*labeling*), i.e., producing more than one annotation (eventually nested) for a given text fragment.[4] A rule for matching person names presented in figure 3 illustrates the syntax. This rule matches a sequence

```
person  :> ((dictionary & [TYPE: "first_name",
                           SURFACE: #first])
            (dictionary & [TYPE: "initial",
                           SURFACE: #in]
             token & [SURFACE: "."]) ?
            (token & [TYPE: "firstCapital",
                      SURFACE: #last])):name
-> name: person & [NAME: #full_name,
                   FIRST_NAME: #first,
                   LAST_NAME: #last,
                   INITIAL: #in
                   AMOUNT: "1"]
                & #full_name := ConcWithBlanks(#first,#in,#last)
                & ValidatePersonName(#full_name).
```

Fig. 3. A rule for recognition of person names

consisting of: a structure of type `dictionary` (output of the dictionary look-up tool) representing the first name, followed by an optional initial (a sequence of `dictionary` and `token` structures), and another structure representing a capitalized token (last name). The symbol `&` links a type name of the FFS with a list of feature-value pairs representing the constraints which have to be fulfilled. It should not be confused with the same symbol denoting unification in XTDL. The symbols `#first`, `#in` and `#last` establish variable bindings to the surface forms of the matched text fragments. Further, the label `name` on the LHS specifies the start/end position of the action defined on the RHS of the rule. This action produce a structure of type `person`, where the value of the slots `FIRST_NAME` , `LAST_NAME` and `INITIAL` is created via accessing the variables `#first`, `#in` and `#last` resp. The value of the `NAME` slot is computed via a call to a functional operator `ConcWithBlanks()` which concatenates its arguments and inserts a space character between them. Finally, the RHS contains a call to a functional operator `ValidatePersonName()` which acts as a boolean predicate and contacts some

[4] XTDL allows only for producing single output structures and does not provide the labeling facility, i.e., output structure correspond to the entire text fragment matched by the LHS pattern. However, there is a 'dirty' workaround consisting of accessing positional information of single feature structures matched by the LHS and using such information for redefining start/end position of the output structure.

external mechanism (i.e., morphological person name filtering) which estimates whether the current name is likely to be a person name or not, and returns an appropriate value. It is important to note that in order for a rule to match, all boolean-valued predicates in the RHS of the rule must hold.

```
killing_event :> ((person_group & [NAME: #n1,
                                   AMOUNT: #a1,
                                   QUANTIFIER: #q1]
                   | person & [NAME: #n1,
                               AMOUNT: #a1]):victim

                  (dictionary & [TYPE: "death_trigger",
                                 FORM: "passive"
                                 METHOD: #m])

                  (person_group & [NAME: #n2,
                                   AMOUNT: #a2,
                                   QUANTIFIER: #q2]
                   | person & [NAME: #n2,
                               AMOUNT: #a2]):killer
                 ):event
-> killer: actor & [NAME: #n2,
                    AMOUNT: #a2,
                    QUANTIFIER: #q2],
   victim: dead & [NAME: #n1,
                   AMOUNT: #a1,
                   QUANTIFIER: #q1]
                & IsNonZeroQuantifier(#q1),
   event: violent_event & [TYPE: "killing",
                           METHOD: #m,
                           ACTOR: #n2,
                           VICTIM: #n1].
```

Fig. 4. A rule for violent event recognition

Another example of a rule that matches information concerning actors and victims in violent events, where a person or a group thereof is killed by another human body, is given in figure 4. This rule matches a sequence consisting of: a FFSs of type `person` or `person_group` (the disjunction is denoted with '`|`') representing a human(s) who is (are) the *victim* of the event, followed by a phrase in passive form, which triggers a 'killing' event (dictionary look-up), and another structure representing the *actor* (person or group of persons). There are three labels on the LHS, namely `victim`, `killer`, and `event`, which produce structures of type `dead`, `actor` and `violent_event` respectively. In case of the `dead` structure, the quantifier (variable `#q1`) must not be a 'zero' quantifier. This con-

```
dead & [NAME: "Talibani", AMOUNT: "230", QUANTIFIER: "Most of"]
actor & [NAME: "US troops"]
violent_event & [TYPE: "killing",
                 METHOD: "shooting",
                 ACTOR: "US troops",
                 VICTIM: "Talibani"].
```

Fig. 5. The output structures produced by the rule in figure 4 when matching the text fragment *Most of the 230 Talibani were shot by the US troops*

straint is expressed in the rule via the boolean predicate `IsNonZeroQuantifier`. The rule described above matches the text fragment *Most of the 230 Talibani were shot by the US troops* and produces three output structures depicted in figure 5. On the contrary, the text fragment *None of the Taliban were killed by UN troops* would not be matched since `IsNonZeroQuantifier` predicate ("*None of*") does not hold.

The handling of Kleene constructions has to be clarified briefly. If a structure containing a variable within a Kleene construction is matched more than once, then (optionally) a local instances of the variable is created for each such submatch, and the local bindings are accumulated into a concatenation thereof. This resembles the weak unidirectional coreferences in XTDL [9]. Further, labels are not allowed within Kleene constructions, and labeled construction are not allowed to consume empty input streams.

The full syntax of ExPRESS extraction rule formalism is given in BNF format in figure 6. Some constructs known from other pattern languages are missing, e.g., negation, but it can be simulated via non-productive rules and prioritization [1].

4.3 Native and External Linguistic Components

In order to facilitate writing grammars ExPRESS comes with a pool of native basic Unicode-aware IE-oriented linguistic processing resources, which includes: (a) a basic tokenizer which segments text based on a list of white spaces and token separators, (b) a tokenizer which additionally performs fine-grained token classification (circa 40 IE-oriented default token classes are provided, e.g. email addresses, URLs, hyphenated constructions, etc.), (c) simple morphological analyzer based on full-form lexica encoded in the MULTEXT[5] format [11], and (d) a space and time efficient dictionary look-up tool which allows for storing huge

[5] MULTEXT was a EU-funded project aiming at developing a set of generally usable software tools to manipulate and analyze text corpora, together with lexicons and multilingual corpora in several European languages. In particular, harmonized specifications for encoding computational lexicons have been established, i.e., same tagset and features are used for all languages.

```
Rules         -> Rule (Rule)*
Rule          -> RuleName ":=" Pattern "->" (Actions)? "."
RuleName      -> Identifier

Pattern       -> "(" Concat ")" (":" Label)?
Label         -> Identifier
Concat        -> Disjunction (Disjunction)*
Disjunction   -> Kleene ("|" Kleene )*
Kleene        -> Element ("+" | "*" | "?")?
Element       -> (BasicElement | Pattern)
BasicElement  -> Type ("&" FeatStruct)?
Type          -> Identifier
FeatStruct    -> "[" Attribute ":" Value ("," Attribute ":" Value)* "]"
Attribute     -> Identifier
Value         -> (SimpleValue (Variable)?) | (Variable)
SimpleValue   -> Identifier
Variable      -> "#"Identifier

Actions       -> Action ("," Action)*
Action        -> Label ":" Type ("&" OutputStruct ("&" FuncOp)* )?
OutputStruct  -> "[" Attribute ":" OVal ("," Attribute ":" OVal)* "]"
Attribute     -> Identifier
OVal          -> (SimpleValue | Variable)?
FuncOp        -> (Variable ":=")? FuncOpName "(" Arg ("," Arg)* ")"
FuncOpName    -> Identifier
Arg           -> (SimpleValue | Variable)
```

Fig. 6. ExPRESS Syntax

amount of entries, where each of them can be associated with arbitrary feature-value pairs. The latter two components exploit the finite-state compression and compilation techniques described in [12], [13] and [14].

Additional external processing components can be easily integrated via implementing a special programming interface. Basically this boils down to providing a function which converts components specific native output format into a stream of disjunctions of FFSs with positional information, and providing functions which return a list of types of output structures returned by this component and features which are appropriate for these types. The latter ones are utilized for performing a strict compatibility check with the types declared in the grammar cascade.

5 Compiling and Processing Grammars

Since the reservoir of FFSs used in extraction rules is potentially infinite, converting ExPRESS grammars into a single and optimized for processing finite-state network is not straightforward. Typically, in a grammar consisting of regular patterns over some feature structures the latter ones are replaced by some unique symbols representing references to these feature structures, i.e., they are treated in a symbolic way (naive implementation). Subsequently, single extraction patterns are merged into a single MDFSA via application of standard finite-state optimization techniques. Although such finite-state device is deterministic in a strict sense, it clearly is not deterministic when we consider the real semantics of its transition labels, i.e., feature structures. Consequently, while processing such automata (being the result of merging the elementary rule automata into one MDFSA), in each step, all outgoing transition from a given state are inspected one by one whether their label matches with the current input feature structures. Since distinct feature structures (even pairs of matching feature structures) are represented as different symbols, some states of the automaton, might have a quite high number of outgoing transitions. This applies in particular for the initial state and in its direct proximity. Inspecting all outgoing transition each time the initial state is visited clearly deteriorates the run-time performance.

The rest of this section describes a method for efficiently processing ExPRESS grammars. First, in in subsection 5.1, the pattern matching algorithm sketched above is described in a more formal manner. Next, in subsection 5.2, some enhancements thereof are introduced, which mainly consist of flattening FFSs in the patterns and input FFSs into character-level regular expressions and strings respectively, so that matching input FFSs with the grammar automaton can be performed efficiently.

5.1 Pattern Matching Algorithm

Let G be a grammar consisting of regular patterns $r_1 \ldots r_n$ over FFSs, where each pattern r_i is represented by a regular expression R_i. FFSs are replaced

in each R_i by symbols representing references to these FFSs. Next, we construct a DFSA M (representing the whole grammar) which accepts the language $R_1 \cdot \{\$_1\} \cup \ldots \cup R_n \cdot \{\$_n\}$, where $\$_1 \ldots \$_n$ are unique symbols representing rule identifiers. Additionally, we turn each state q into a final state if it has an outgoing transition labeled with one of the symbols in $\{\$_1, \ldots, \$_n\}$. All other states are non-final. Further, let us assume, that the stream of input FFSs is represented as a directed labeled graph $InputFS = (V, E)$, where all nodes in V correspond to start/end positions of text spans associated with the input FFSs. An edge in E is a 3-tuple (v, a, u), where v and u are source/target nodes, and a is the label which points to some FFS.

An algorithm that takes automaton M and finds all matches in $InputFS$ (an input stream of flat feature structures) is presented in figure 7. Please note that M is is not deterministic when we consider the real semantics of its transition labels. The variable *node* (initialized in line 1) points to the current node in $InputFS$, i.e., the node from which the algorithm tries to find the next potential match. The main **while** loop of the algorithm (lines 3-20) is executed until the current node is the last node in $InputFS$. Since there is potentially more than one path from the node u in $InputFS$ which matches with the automaton M and due to the fact that even one single path in $InputFS$ might match with different paths in M, we store in the set *Active* all 'current' configurations of M. A single *configuration* of M is a triple (q, π, v), where q denotes the current state of M, π is a sequence of input FFSs which match a path in M from q_0 to q, and v denotes the next node in $InputFS$ from which subsequent matches in the input stream will be sought. Analogously, in *Accepting* we store all *accepting configurations* of M (ones whose current state is final). Initially this set is empty (line 5). In the **while** loop in lines 6-15 all possible configurations of M that match some path in $InputFS$ starting in the node *node* are computed. This process resembles breadth-first-search in graphs. In particular, in the inner loop (lines 8-14) for each $(q, \pi, v) \in Active$ we compute all 'subsequent' configurations, i.e, the ones being the result of matching some input FFS a starting in node v with a FFS a' in the set of transitions for state q, so that $\delta(q, a') \neq \perp$. Matching test is done via a call to the function MATCHES (line 13). Note that for a single input FFS there might be potentially more than one matching transition in M (**for** loop in lines 12-13). Once all 'new' configurations have been computed, we select from the set of accepting configurations one which fulfills selection criteria (line 17). Selection criteria may vary, depending on the search strategy. For instance, in the *longest-match strategy*, one simply takes the configuration which covers the longest text span. If more than one such configuration exists, then the one being a result of application of a rule with highest priority is chosen, etc.[6] Once an accepting configuration is chosen, an appropriate action is performed (line 18), e.g., output structure(s) is produced. We can restore the rules that matched via inspecting transition labels from final states. Finally, the value of the current

[6] In some applications, it is convenient to select more than one accepting configuration, but the modification to the presented algorithm is straightforward so it is not discussed any further.

node in the input graph is then modified accordingly in the line 19. If no accepting configurations were found, the current node is set to the closest node in $InputFS$ that has an outgoing edge (line 20).

```
FIND-MATCHES(M = (Q, Σ, δ, q_0, F), InputFS)
 1  node ← GETFIRSTNODE(InputFS)
 2  lastNode ← GETLASTNODE(InputFS)
 3  while node ≠ lastNode
 4  do Active ← {(q_0, ε, node)}
 5     Accepting ← ∅
 6     while Active ≠ ∅
 7     do Next ← ∅
 8        for (q, π, v) ∈ Active
 9        do if q ∈ F
10              then Accepting ← Accepting ∪ {(q, π, v)}
11           for (v, a, u) ∈ InputFS
12           do for a' ∈ Σ : δ(q, a') ≠ ⊥
13              do if MATCHES(a, a')
14                    then Next ← Next ∪ {(δ(q, a'), π · a', u)}
15        Active ← Next
16     if Accepting ≠ ∅
17       then (q, π, v) ← SELECTACCEPTINGCONFIG(Accepting)
18            EXECUTEACTION(M, q, π)
19            node ← v
20       else node ← GETNEXTNODE(InputFS, node)
21  return
```

Fig. 7. Pattern matching algorithm

Intuitively, the most time-consuming part of the algorithm in figure 7 is the **for** loop in lines 12-14. In the naive implementation one has to inspect all outgoing transitions from the state q whether their label (a') matches with the current input FFS (a). Inspecting all outgoing transition for frequently visited states, e.g., the initial state and its direct proximity, clearly deteriorates the run-time performance.

For alleviating the aforementioned problem JAPE applies a solution which exploits the fact that the feature structures being labels of outgoing transitions from a given state have shared parts. In particular, all such structures are partitioned into disjoint partial feature structures which do not intersect and they are reordered accordingly in order to avoid redundant computations while matching the stream of input feature structures.

In XTDL, where the recognition part of the rules consists of TFSs, a similar technique for ordering the outgoing transitions is used. It consists of comput-

ing a transition hierarchy under TFS subsumption for all outgoing transitions (labels) of a given state. While traversing the grammar automaton, these transition hierarchies are utilized for inspecting outgoing transitions from a given state, starting with the least specific transition(s) first, and moving downwards in the hierarchy, if necessary. Although this technique proved to give a significant speed-up, the number of transitions which have to be inspected for computing 'subsequent' automaton configurations might be on an average relatively high due to the low degree of feature-value sharing.

5.2 Matching Flat Feature Structures

In order to efficiently perform the crucial matching step in the algorithm described in the previous section (lines 12-14) we apply in ExPRESS a technique which consists of flattening input FFSs into strings and converting all transitions labels of a given state into a single DFSA, so that computing 'new' target states (new automata configurations) is reduced to performing a simple deterministic automaton look-up.

Generally speaking, the process of finding a match at a given position in the input stream is split into three steps: (1) selection of the sequence(s) of input FFSs which is (are) covered by some rule(s) according to predefined selection strategy, (2) performing a fully-fledged match of the selected rule(s) against the selected input sequence of FFSs, which includes variable and label binding, and (3) producing and merging output structures. Postponing variable and label binding allows for efficiently implementing step (1). Further, once an input sequence and the rules (or more) that match this sequence have been selected, performing full matching in step (2) can be done quickly due to the limited number of applicable rules. Thus, step (1) can be seen as a prefiltering of applicable rules. Since there are potentially several paths in the automaton for the rule(s) selected in step (2), step (3) is necessary for merging and/or filtering out some output structures, but we do not describe it here any further.

We now turn to implementing step (1) and sketch the technique for quick computing matching transitions from a given state in a semi-formal way. Firstly, let us observe that only a finite number of feature-value pairs are used in the grammar rules. We can compute for all FFSs of a given type α, which appear in the rules, the respective value sets $\Sigma_1, \ldots \Sigma_k$, where Σ_i is the value-set for the i-th feature appropriate for the type α.[7] A given input FFS $s = [f_1 : v_1 \ldots f_k : v_k]_\alpha$ can be then encoded as a string $id(\alpha) \cdot \$ \cdot {v^*}_1 \cdot \$ \ldots \$ \cdot {v^*}_k$, where id maps types to unique symbols representing their identifiers, \$ is a unique symbol $\notin \Sigma_i \cup \{\top\}$ $(\forall 1 \leq i \leq k)$ which represents a separator and ${v^*}_i \in \Sigma_i \cup \{\top\}$ are defined as follows:

$$ {v^*}_i = \begin{cases} v_i : v_i \in \Sigma_i \\ \top : v_i \notin \Sigma_i \ \vee v_i = \top \end{cases} $$

For instance, an input FFS $[pos : noun, case : loc, gen : fem]_{morph}$ with pos, $case$, and gen being appropriate features for the type $morph$, where $\Sigma_{pos} =$

[7] Note that we order the features appropriate for a given type

$\{noun, adj\}$, $\Sigma_{gen} = \{masc, fem\}$, and $\Sigma_{case} = \{nom, acc, dat, gen\}$ (seen feature values), would be represented as the following string: $id(morph) \cdot \$ \cdot noun \cdot \$ \cdot \top \cdot \$ \cdot fem$.

Analogously, each FFS $s = [f_1 : v_1 \ldots f_k : v_k]_\alpha$ being a label of a transition t from a given state in the grammar automaton is represented as a regular expression of the form $id(\alpha) \cdot \$\ v^*{}_1 \cdot \$ \ldots \$ \cdot v^*{}_k \cdot \% \cdot trans(t)$, where id and \$ are defined as previously, % is another unique separator, $trans$ maps transitions to their unique symbolic identifiers, and $v^*{}_i \in (\Sigma_i \cup \{\top\})^*$ is a regular expression defined as follows:

$$v^*{}_i = \begin{cases} v_i & : v_i \in \Sigma_i \\ \{\top\} \cup \Sigma_i & : v_i = \top \end{cases}$$

The second part of the definition of $v^*{}_i$ has to be a disjunction of $\{\top\}$ and Σ_i since we intend to merge all regular expressions representing transitions from a given state into a single DFSA ('transition' automaton for a given state), i.e., in case of encoding a feature with unspecified value, all values (for that feature and type) seen in other patterns have to be considered (Σ_i). Now, let $T_1, \ldots, T_n$ be the regular expressions representing the labels of the transitions $t_1, \ldots, t_n$ from a given state q in M resp., which were obtained in the previously described manner. Let M_q be a DFSA which accepts the language $T_1 \cup \ldots \cup T_n$. Then, we can compute the set of possible target states for the state q in M and an input FFS a that is represented as a string w simply via computing a target state $p = \delta_{M_q}(q, w)$ in M_q and inspecting all outgoing paths from p, whose labels start with % in order to retrieve the target state identifiers in the grammar automaton M. In this way, the steps 12-14 in the algorithm in figure 7 are reduced to a simple string matching with the DFSA M_q.

We give an example to clarify the aforementioned technique. Let us assume that t_1 and t_2 are two outgoing transitions from state q, which are labeled with $[pos : noun, case : \top, gen : \top]_{morph}$ and $[pos : \top, case : acc, gen : \top]_{morph}$ and which lead to state q_1 and q_2 resp. Turning them into corresponding regular expressions yields $id(morph) \cdot \$ \cdot noun \cdot \$ \cdot \{nom, acc, gen, dat, \top\} \cdot \$ \cdot \{fem, masc, \top\} \cdot \%q_1$ for t_1 and analogously $id(morph) \cdot \$ \cdot \{noun, adj, \top\} \cdot \$ \cdot acc \cdot \$ \cdot \{fem, masc, \top\} \cdot \%q_2$ for t_2. The result of merging regular expressions representing the labels of t_1 and t_2 into one DFSA M_q is shown in figure 8 in a simplified form (\$ symbols were omitted).

Let us assume that an input FFS $s = [pos : noun, case : acc, gen : masc]_{morph}$ has to be matched against the grammar automaton M in state q. Matching the string representation of s, i.e., $id(morph) \cdot \$ \cdot noun \cdot \$ \cdot acc \cdot \$ \cdot masc$, in the transition automaton M_q results in state 8. Consequently, both states q_1 and q_2 are reachable via matching FFS s in M from state q.

Techniques similar to the one described in this section are also used in other finite-state based frameworks, e.g., in [15]. A further improvement could be achieved by turning all input FFSs at a given position into a union of their corresponding string representations and subsequently performing on-the-fly intersection thereof with the 'transition' automaton representing the outgoing transitions from a given state. Whether this results in an enhanced run-time performance is

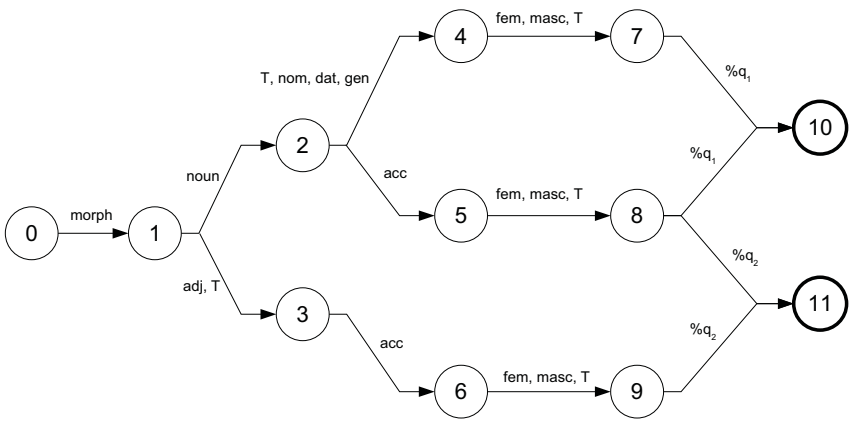


Fig. 8. Transitions labels merged into a single DFSA

unclear since intersection operation is more time-consuming than a single string acceptance check.

6 Technicalities

ExPRESS has been implemented fully in JAVA. The development is based on the Java Compiler Compiler [16] and the Java package `dk.brics.automaton` containing time efficient implementations of finite-state automata and a bag of standard operations for manipulating and optimizing them [17]. Currently, ExPRESS consists of two stand-alone programs (parser and interpreter) and a documented JAVA API for facilitating integration into other frameworks. Making ExPRESS publicly available for research purposes is envisaged at a later stage.

We have carried out some experiments to measure the run-time behavior of Express with a two-stage grammar for recognition of information on actors, kidnapped, dead and wounded in violent events. In the first stage standard named entities are recognized, e.g., persons, group of persons, numerical expressions, etc. In the second stage, single-slot and two-slot extraction rules are applied to retrieve the sought-after information on related events, in which the entities recognized in the first stage participate. The first-stage grammar consisting of circa 100 rules was developed by an expert, whereas the second-level grammar was obtained via semi-automatic conversion of ca. 3000 automatically learned IE patterns [18] into ExPRESS rules. Further, five linguistic processing resources (e.g., tokenizer, gazetteer and morphology look-up) were involved in the extraction process. Subsequently, the aforementioned two-level grammar has been converted in almost one-to-one manner into a XTDL grammar. It turned to be

a relatively simple task since the core linguistic components provided with ExPRESS have nearly identical functionality and I/O specification as those used in SProUT. However, some rules had to be expressed as two rules in XTDL since XTDL rules do not allow for specifying more than one output structure directly.

In an experiment, the grammars were applied to a 167 MB excerpt of the news in English on terrorism, consisting of 122 files on a PC Pentium 4 machine with 2,79 GHz. The table 1 gives figures of the average run-time (in seconds) for processing a single file (average size of 1,37 MB) at different stages. The average number of matches per document amounted to ca. 60 000. Clearly, ExPRESS

Time \ Grammar Interpreter	XTDL	ExPRESS
core linguistic components stage I	2.451	1.818
entity-pattern matching	38.212	1.923
entity-structure production	4.172	0.515
core linguistic components stage II	1.092	0.639
event-pattern matching	12.124	0.666
event-structure production	0.156	0.013
Total	58.207	5.574

Table 1. Run-time behavior (in seconds): XTDL vs. ExPRESS

performs significantly better than XTDL interpreter. The pattern matching itself constituted 46, 34% (ExPRESS) and 86, 48% (XTDL) of the total processing time respectively. In a second experiment, we have slightly 'compressed' the XTDL grammar through using coreferencing and other XTDL specific features, which resulted in deterioration of the run-time performance by the factor of two.

Finally, in the last experiment, we applied the same cascade of grammars to a collection of sentences (8 MB), where for each sentence in this collection there is at least one second-stage extraction rule that matches. ExPRESS run-time amounted to 36,7 seconds, whereas SProUT needed for processing the same collection ca 575 seconds.

Although converting ExPRESS grammars into JAPE format is a more laborious task, the above run-time figures for ExPRESS are better than one could potentially obtain when using JAPE according to the author's 'subjective' experience with the latter one and some basic experiments of converting the first-level grammar into JAPE.

7 Summary

In this paper, we presented ExPRESS, a new IE-oriented pattern specification and recognition engine, which borrows heavily from two previously introduced

pattern languages, namely JAPE and XTDL. In particular, ExPRESS grammars consist of extraction rules which are regular expressions over flat feature structures with string-valued features. ExPRESS was developed primarily in order to find a trade-off between 'compact descriptions' and efficient processing of huge text collections. It is already operational and it is being deployed in a real-time news event extraction system for detecting violent and natural disaster events [19]. In particular, ExPRESS is capable of applying modest-size event extraction grammars on MB-sized texts within seconds. Clearly, XTDL or some other IE-oriented pattern languages are more expressive and more powerful, but there is a wide range of extraction tasks for which ExPRESS will come in handy and might constitute a time-efficient alternative.

In future work, the pattern formalism will be extended by adding some new constructs and providing new native processing resources. Going beyond 'sequential' processing of grammars is planned. In general, ExPRESS will be kept as minimal as possible and any future developments will be strictly driven by the needs of specific applications. In particular, it will be deployed for named-entity and relation extraction. Finally, exploring additional performance enhancing techniques for processing grammars is envisaged, e.g., (a) intelligent reordering of feature-value pairs in the FFS in such a way that features which are most likely to eliminate a high number of potential target states precede other feature-value pairs, and (b) conversion of input FFSs starting at a given position into a union of their corresponding string representations and subsequently performing on-the-fly intersection thereof with the 'transition' automaton representing the outgoing transitions from a given state.

Acknowledgments

The work presented in this paper was supported by the Europe Media Monitoring Project (EMM) carried out by the Web Mining and Intelligence Action in the Joint Research Centre of the European Commission. I would like to thank Hristo Tanev, Vanni Zavarella, Bruno Pouliquen, Ralf Steinberger, Camelia Ignat and other EMM colleagues for fruitful discussions. Finally, I am indebted to Clive Best and Delilah Al-Khudhairy for valuable comments and proof-reading of this paper.

References

1. Cunningham, H., Maynard, D., Tablan, V.: JAPE: a Java Annotation Patterns Engine (Second Edition). Technical Report, CS–00–10, University of Sheffield, Department of Computer Science (2000)
2. Drożdżyński, W., Krieger, H.U., Piskorski, J., Schäfer, U., Xu, F.: Shallow Processing with Unification and Typed Feature Structures — Foundations and Applications. Künstliche Intelligenz **2004(1)** (2004) 17–23
3. Best, C., van der Goot, E., Blackler, K., Garcia, T., Horby, D.: Europe Media Monitor. Technical Report EUR 22173 EN, European Commission. (2005)

4. Copestake, A.: Appendix: definitions of typed feature structures. Natural Language Engineering **6**(1) (2000) 109–112
5. Neumann, G., Backofen, R., Baur, J., Becker, M., Braun, C.: An information extraction core system for real world German text processing. In: Proceedings of the 5th International Conference of Applied Natural Language. (1997) 208–215
6. van Noord, G., Gerdemann, D.: Finite State Transducers with Predicates and Identity. *Grammars* **4(3)** (2001) 263–286
7. Appelt, D., Onyshkevych, B.: The Common Pattern Specification Language. In: Proceedings of Tipster Text Program - Phase III. (1998) 23–30
8. Japec: Japec – A Jape-to-Java Optimizing Compiler. Web document, http://www.ontotext.com/gate/JapecPres.pdf (2006)
9. Drożdżyński, W., Krieger, H.U., Piskorski, J., Schäfer, U., Xu, F.: A Bag of Useful Techniques for Unification-Based Finite-State Transducers. In: Proceedings of of 7th KONVENS Conference, Vienna, Austria. (2004)
10. Muslea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. In: Proceedings of AAAI 1999. (1999)
11. Erjavec, T.: MULTEXT - East Morphosyntactic Specifications (2004)
12. Daciuk, J.: Incremental Construction of Finite-State Automata and Transducers. PhD Thesis. Technical University Gdańsk. (1998)
13. Piskorski, J.: On Compact Storage Models for Gazetteers. In: Proceedings of the 5th International Workshop on Finite-State Methods and Natural Language Processing, Helisnki, Finland, Springer, LNAI (2005)
14. Daciuk, J., Piskorski, J.: Gazetteer Compression Technique Based on Substructure Recognition. In: Proceedings of Intelligent Information Systems 2006 - New Trends in Intelligent Information Processing and Web Mining, Springer Verlag series "Advances in Soft Computing" (2006)
15. Skut, W., Ulrich, S., Hammervold, K.: A Flexible Rule Compiler for Speech Synthesis. In: Proceedings of the International IIS:IIP WM'2004 Conference. Zakopane, Poland. Springer, Advances in Soft Computing. (2004) 257–266
16. JavaCC: https://javacc.dev.java.net
17. Moller, A.: http://www.brics.dk/automaton (2007)
18. Piskorski, J., Tanev, H., Oezden-Wennerberg, P.: Extracting Violent Events from On-line News for Ontology Population. In: 10th International Conference on Business Information Systems. Poznan, Poland. Lecture Notes in Computer Science, LNCS 4439. (2007) 287–300
19. Tanev, H., Piskorski, J., Atkinson, M.: Real-Time News Event Extraction for Global Crisis Monitoring. In: Proceedings of the 13th International Conference on Applications of Natural Language to Information Systems (NLDB 2008), London, UK, 24–27 June, 2008. Lecture Notes in Computer Science Vol 5039, Springer Verlag Berlin Heidelberg. (2008) 207–218

On Resolving Long Distance Dependencies in Russian Verbs

Dirk Saléschus

University of Konstanz, Germany

Abstract. Morphological analyses based on word syntax approaches can encounter difficulties with long distance dependencies. The reason is that in some cases an affix has to have access to the inner structure of the form with which it combines. One solution is the percolation of features from ther inner morphemes to the outer morphemes with some process of feature unification. However, the obstacle of percolation constraints or stipulated features has lead some linguists to argue in favour of other frameworks such as, e.g., realizational morphology or parallel approaches like optimality theory. This paper proposes a linguistic analysis of two long distance dependencies in the morphology of Russian verbs, namely secondary imperfectivization and deverbal nominalization.We show how these processes can be reanalysed as local dependencies. Although finite-state frameworks are not bound by such linguistically motivated considerations, we present an implementation of our analysis as proposed in [1] that does not complicate the grammar or enlarge the network unproportionally.

1 Secondary Imperfectivization in Russian Verbs

Like tense and person, aspect is one of the grammatical categories of Russian verbs. The aspectual category consists in the opposition between perfective and imperfective aspect. The majority of Russian verbs expresses this grammatical distinction, although its morphological exponence is quite complex. Prefixation, suffixation, suppletion, stem allomorphy, and a combination thereof can be used. We will not consider the intricate semantics of the aspect [2] but only concentrate on the forms, especially on the joint interaction of prefixation and suffixation. An overview of the data can be found in any grammar of Russian, as e.g. in [3].

We start with simple verb stems. The overwhelming majority of them are imperfective, however, there also exist perfective simple stems. The aspect of simple stems is thus an idiosyncratic property and has to be marked for each single stem.

Before turning to the morphological expression of the aspectual opposition it is important to consider first the formation of complex verb stems by means of lexical prefixation. Lexical prefixation of verbs is quite productive in Russian. It is a derivational process and leads to the formation of both new lexemes and complex stems. From a semantic point of view this process of lexeme formation can either be rather opaque or can lead to quite transparent composed meanings

as, e.g., in the case of different semantic classes called *Aktionsarten*. See the following examples for semantically transparent and opaque lexical prefixation:[1]

1	*nosít'*	(carry-indet.ipf.)	*vnosít'*	(carry in-pf.)
	vynosít'	(carry out-pf.)	*iznosít'*	(to wear out-pf.)
2	*begát'*	(run-indet.ipf.)	*vbegát'*	(run inside-ipf.)
	vybegát'	(run out-ipf.)	*izbegát'*	(avoid-ipf.)

We also give an example for semantically transparent lexical prefixation with the ingressive *Aktionsart* from [4]:388 f.:

3	*govorít'*	(speak-ipf.)	*zagovorít'*	(start speaking-pf.)
	igrát'	(play-ipf.)	*zaigrát'*	(start playing-pf.)
	kričát'	(cry-ipf.)	*zakričát'*	(start crying-pf.)

From a morphological point of view, there are around 20 prefixes which can be used for lexical prefixation with both perfective and imperfective simple verb stems. Lexical prefixation can also be applied cyclically, leading to complex forms such as:

4	*polnít'*	(sloppy:fill-pf.)	*výpolnít'*	(fulfil-pf.)
	perevýpolnit'	(overfulfil-pf.)		

This phenomenon is also found in other languages like English or German which forms the verbs *füllen*, *erfüllen*, and *übererfüllen*, respectively.

Note, however, that not every complex stem is formed from an actual existing base stem. There are verbs like *dobávit'* (fill-pf.), *pribávit'* (add-pf.), *zabávit'* (amuse-pf.), without an existing verb *bávit'*. Even though they look like complex stems they have to be analyzed as simple stems. This is similar to English morphology with verbs like *perceive*, *receive* with no existing word *ceive* (see [5] for a discussion of such examples).

A last fact to note about lexical prefixation is that some stems have only one or a few actual prefixed variants whereas others combine with lots of prefixes [4]. See, e.g., the possible lexical derivations of the stem *xodít'* (go-indet.ipf.):

vxodít', vsxodít', vyxodít', doxodít', zaxodít', isxodít', naxodít', obxodít', otxodít', perexodít', poxodít', podxodít', prixodít', proxodít', rasxodít's'a, sxodít', uxodít'; only exception: **nadxodít'*

[1] The following writing conventions are adopted here: the *y* stands for the high back unrounded dorsal [ɨ]. A soft consonsant is a consonant that is palatal or has secondary palatalization. The latter feature is signalled by an apostrophe after the consonant (e.g. *t'*). The softness of consonants is predictable when they are followed by the front vowels *i* or *e* and is left out in these contexts. The symbols č, š and ž stand for a soft alveo-palatal affricate [tʃ] and the posterior voiced and unvoiced fricatives [ʃ] and [ʒ], respectively. At the surface, the č is always soft whereas š and ž are always hard. Finally, an accent signals stress.

Our lexicon shall be fully productive and contain all potential complex word forms, whether actually existing in Russian or not.

From a grammatical point of view there is one important biproduct associated with lexical prefixation. All newly formed lexemes are always perfective stems. In other words, lexical prefixation always leads to perfectivization (with just a few exceptions). This shows the intricate connection between lexical and aspectual meaning [6] but its description goes beyond the scope of this work.

Now we can look at the exponence of the aspectual opposition by means of prefixation and suffixation (ignoring other morphological exponences). Imperfective simple verb stems can express the perfective aspect by grammatical prefixation. From a semantic point of view this stem formation does not alter the lexical meaning. From a morphological point of view a new complex stem is formed.

The crucial fact is that the set of prefixes used for grammatical prefixation is identical to the set of prefixes used for lexical prefixation. However, there is one important difference between grammatical and lexical prefixation – for each simple imperfective stem there is exactly one prefix which is used exclusively for grammatical prefixation. All remaining prefixes can be used for lexical prefixation. The choice of the grammatical prefix that can combine with an imperfective simple stem is not predictable and has to be marked for every simple imperfective stem. The following sketch with some prefixes and stems illustrates this ("G" means grammatical prefixation, "L" means lexical prefixation):

		G: *postróit'*	(build-pf.)
stróit' (build-ipf.)	po	L: *popisát'*	(write a bit-pf.)
		L: *podélat'*	(carry on-pf.)
		L: *nastróit'*	(adjust-pf.)
pisát' (write-ipf.)	na	G: *napisát'*	(write-pf.)
		L: *nadélat'*	(cause-pf.)
		L: *sostróit'*	(look surly-pf.)
délat' (do-ipf.)	s	L: *spisát'*	(copy-pf.)
		G: *sdélat'*	(do-pf.)

If a simple imperfective stem has formed a grammatically prefixed partner stem both stems together make up an aspectually complete verbal lexeme. Since the aspectual opposition is not expressed via exponence on the same stem (as is the case for other grammatical categories like number or person) this morphological process is called grammatical derivation [7].

Now let us see how a (simple or complex) perfective stem expresses the imperfective form in order to create an aspectually complete lexeme. In traditional analyses simple perfective verb stems can change the stem vowel to express imperfective partner stems. In our analysis this is considered to be a case of suffixation. There are two allmorphs of the imperfective suffix: an empty V-slot

and the string /yv/. If not filled by an adjacent vowel from some suffix the empty slot is per default filled with the vowel [a]. The right allomorph of the ipf. suffix is determined by morphological class membership. Interestingly the allomorph for simple perfective stems is always the same, namely *a*:

5 throw: *brósit'* (pf.) *brosát'* (ipf.)
deprive: *lišít'* (pf.) *lišát'* (ipf.)

The *i* in *brosít'* could also be analyzed as an aspectual suffix. However, we analyze it as a thematic vowel for one morphological class of simple perfective stems. The root would then be *bros*. It is a fact that morphological verb classes assign thematic vowels to stems of a paradigm in different ways. Sometimes this vowel is kept in only some stems of the paradigm, sometimes in almost all stems. This separation of thematic vowels from roots facilitates the analyses of the imperfective suffix and of the deverbal nominalization.

Complex perfective verb stems also use suffixation. For these stems this process is called secondary imperfectivization. This is a grammatical process only and is never an option for simple imperfective verb stems. Complex pf. stems normally take the allomorph *yv* but some have the allomorph a and some even have both:

6 manufacture: *izgotóvit'* (pf.)
izgotovl'át' (ipf.) or *izgotávlivat'* (ipf.)

Finally, some (simple or complex) perfective stems show consonant alternations when imperfectivized while others do not. This also has to be marked lexically:

7 stem allmorph: render: *javít'* (pf.) *javl'át'* (ipf.)
manufacture: *izgotóvit'* (pf.) *izgotávlivat'* (ipf.)

8 no stem allmorph: throw: *brósit'* (pf.) *brosát'* (ipf.)
copy: *perepisát'* (pf.) *perepísyvat'* (ipf.)

The following table gives an overview of prefixation and suffixation in Russian verb stems with two example verbs. Empty cells signal that the morphological process is not possible.

	simple verb stem					
	perfective (*brós-it'*)			imperfective (*pis-át'*)		
prefixation	gramm	lex	none	gramm	lex	none
↓ input to ↓		*výbrosit'*	*brósit'*	*napisát'*	*spisát'*	*pisát*
ipf. suff		*vybrásyvat'*	*brosát'*		*spísyvat'*	

Before turning to the analysis we want to point out an interesting typological fact. In morphological approaches based on word syntax there is some discussion about the notion of a head (e.g. [8] or [9]). On the assumption that Russian complex stems are indeed derived and not stored in the lexicon, both prefixes and suffixes can determine the aspect of the complex stem and thus constitute the head. The last applied affix always has the last word.

2 Blocking Affixation with Derived Flag Diacritics

The crucial question is how secondary imperfectivization can be blocked for grammatically prefixed perfective complex stems. We will first sketch the solution informally and then consider possible implementations.

In fact, the imperfective suffix needs two kinds of information. First, is the stem perfective or imperfective? Suffixation is only possible for perfective verb stems.

The second question is whether the verb is lexically prefixed. Secondary imperfectivization applies only if the complex stem is created by lexical prefixation, not by grammatical prefixation. Thus, imperfectivization is accomplished in order not to just create imperfective verb stems but to create imperfective partner verbs, i.e. aspectually complete lexemes.

How can this long distance dependency be captured in a morphological framework? How can a morphological type of affixation be made sensible to the complex morphological as well as lexical structure of a verb stem?

One possibility would be the following. Assume that the imperfective suffix requires a stem which is not yet aspectually complete. Assume also that every stem can signal whether it is aspectually complete by some kind of feature.

Next let us assume that every simple imperfective stem is marked for its matching grammatical prefix (if there is one at all). Let us call this marking the stem-prefix-feature. Once this prefix is encountered the lexeme is saturated and signals that it is aspectually completed by setting the mentioned feature. The imperfective suffix is then blocked from application by reference to that feature. Thus, if the prefix and the stem-prefix-feature match, then a new feature is set and suffixation is blocked by that new feature.

There are several ways to implement these ideas in computational linguistics. [10] discusses different strategies, among them using concurrent rule transducers as in two-level morphology or composing in constraints at compile time. The first solution has the disadvantage of slower performane at runtime whereas the latter solution leads to an enormous increase in network size.

Beesley favours a solution with flag diacritics (also described in [1]). In *xfst*, flag diacritics are part of the normal alphabet insofar as they are interpreted like epsilons. However, the enhanced *xfst* lookup routines process them specially and enforce the dependencies between morphemes. The lookup routines do this by introducing a small amount of memory which suffices to capture the long distance dependencies [10]. The overall flag system in *xfst* is non-monotonic and thus exceed the expressive power of regular transducers [1]. With flags it is in

principle possible to create a blatantly overgenerating lexicon and let the lookup routines rule out impossible or undesired combinations. The only disadvantage of flags is a possible slower performance due to backtracking.

We try to solve the problem with the Russian data by reference to flag diacritics, a feature-like symbol in *xfst*. The solution seems quite obvious: the flag of the imperfective suffix interacts with the flags from stem and prefix according to well-defined conditions. This is broken down into two steps. First the flags of the prefix and the stem interact. Second, the result is handled over to the flag of the imperfective suffix. There are several kinds of flags triggering different processes and so again there are several possible strategies for an implementation of flags. We will not describe the whole inventory of flags in *xfst* but hope that the following discussion is easy to understand. For detailed information we refer the reader to [1].

Working exclusively with flag diacritics, one could assign every prefix an own flag signaling a positive value, e.g. `@P.NA.PLUS@` for the prefix *na* and similarly `@P.U.PLUS@` for the prefix *u*. The first `P` stands for an operation over the feature, in that case positive setting of the feature. `NA` is the name of the feature and `PLUS` is its value. The whole expression is surrounded by @-signs.

Stems are also assigned flags. A stem flag checks the value of the prefix flag and resets it only if the prefix combines with this stem for grammatical prefixation. E.g., the grammatical prefix of the stem *pis* is *na* and therefore changes its flag value. This is achieved by the flag `@N.NA.MINUS@`. In that case, `N` signals negative resetting of the value ot the `NA` feature such that the value is reset to `MINUS`. The value of another prefix like *u* is left intact by the stem *pis*.

The imperfective suffix, finally, is also assigned a flag requiring a flag with a value set to `PLUS`. It has to list all possibilities like `@R.NA.PLUS@`, `@R.U.PLUS@` etc.

A simplified example shall illustrate how this works. The following expressions would lead to possible concatenations in *lexc*[2] (concatenation is accomplished via continuation classes (linked sublexicons) and here marked with a plus for clarification):

```
na@P.NA.PLUS@+pis@N.NA.MINUS@+[yv @R.NA.PLUS@ |
@R.U.PLUS@...]

u@P.U.PLUS@+pis@N.NA.MINUS@+[yv@R.NA.PLUS@ |
@R.U.PLUS@...]
```

In the first case the value of the first flag is set to `PLUS`. This value is reset by the stem flag to `MINUS`. As a result, the flag of the imperfective suffix requiring a `PLUS` value can no longer match with this complex stem. In the second case the value of the first flag is again set to `PLUS`. This time it is left intact by the

[2] A language for specifying lexicons, also provided by the **XEROX** finite-state tools [1].

stem flag and the imperfective suffix can successfully check for a PLUS value. This gives us exactly the right results.

With the sketched solution the imperfective suffix would have attached to it flags which check for every possible prefix. This is expressed by the disjunctive listing of flags for the imperfective suffix, indicated by "|". One could also have multiple entries for the imperfective suffix, each one bearing only flag. However, we would like to keep the number of flags and morphemes minimal.

There is a second strategy which uses a combination of flags and continuation classes by doubling the entries for stems. To take the example from above, the prefix *na* takes again the flag `@P.NA.PLUS@`. The first entry for the stem *pis* has the flag `@D.NA.PLUS@` where the `D` indicates that the feature `NA` is not allowed to have the value `PLUS`. Thus the stem *pis* may combine with any prefix except the one it uses for grammatical prefixation, namely *na*. The continuation class of that stem is the imperfective suffix. The second entry for *pis* has the flag `@R.NA.PLUS@`. Here the `R` indicates the requirement for a preceding flag with the feature `NA` set to the value `PLUS`. The absence of a preceding flag or any other flag setting is forbidden. The continuation class of that stem entry can be anything except the imperfective suffix. The obvious disadvantage of that solution is the increase in network size by doubling information.

There is also a third solution with single entries for all morphemes and a minimal number of flags used that is already sketched in [1]. In that analysis all morphemes in the overgenerating *lexc* grammar have a special formal marking. Rewrite rules check the markings of the morphemes and change them into flags in special contexts. To take a concrete example, the prefix *na* has the notation `naPLEX`, the prefix *u* has the form `uPLEX`, and the prefix *s* has the form `sPLEX`. `PLEX` is a dummy meaning lexical prefix. Stems also have some special formal marking which indicates the prefix that is used for grammatical prefixation. E.g., the stem *pis* has the form `pisNA`, indicating that the prefix *na* is used for grammatical prefixation. Similarly, the stem *sluš* is notated as `slushU`. Finally, the imperfective suffix also has an additional formal marking, namely `IMPRLEX`. All these additional formal markings are just menmonic placeholders and could be transformed into a more sophisticated notation according to the linguist's needs. When the *lexc* grammar is compiled it contains all conceivable combinations of prefixes, stems and suffixes, among them unwanted combinations. In a next step simple rewrite rules delete the special formal marking of the prefix if the prefix happens to have the same form as the extra formal marking of the stem. Rewrite rules are given here in the *xfst* formalism. They denote regular relations which can later be composed with the lexicon.

The form `naPLEX`, e.g., is replaced by simple `na` if somewhere after the prefix the form `NA` is found:

```
define rule1 [n a P L E X -> "na" || _ $[N A] ];
```

All that is left to do now is to transform all special formal markings which have not been deleted into flag diacritics. This is also accomplished by rewrite

rules which are ordered after the prefix rules.

```
define rule2 [P L E X -> "@P.LEX.LEX@"] ;
define rule3 [R L E X -> "@R.LEX.LEX@"] ;
```

In effect, only prefixes which combine with stems in lexical prefixation can now have a flag which is required by the imperfective suffix. The imperfective suffix cannot combine with grammatically prefixed stems because in the complex stems the prefixes do not have the required flag. The long distance dependency between prefix and suffix is thus resolved into a local binary interaction between only two flags. This is similar to the resolution of long distance dependencies in phonology. A phoneme can be conceived as a muli-tiered representation with each feature belonging to its own tier [11]. Two sounds, although separated from each other by intervening sounds, can excert an influence on each other because their features are neighbours on one of these tiers.

Our implementation has several advantages. From a linguistic point of view, it comes close to the theoretical analysis. There are two stages of morphological stem formation and each time only two features are checked. It is not the case that one affix has to have access to the inner structure of a complex stem. In order to achieve these theoretical goals, in our analysis flags are not part of the lexicon but instantiated in a later step by rewrite rules.

From an implementational point of view, we avoid the use of filters and the resulting increase in network size. We also use a minimal amount of information by severely restricting the number of flags and continuation classes. As is to be expected, each time a stem is added to the lexicon the amount of states and arcs is more increased in the hybrid solution than in the solution with only flags. The opposite is true with the number of paths. As far as the mini lexicon and the addition of a handful of stems is concerned, the overall increase in network size is the same for both solutions.

So far we have shown how to block a special case of suffixation. The long distance dependency was resolved into two local phenomena. First, the concatenation of prefixes and stems was checked for some kind of pattern matching. Second, the concatenation of the complex stem and the imperfective suffix was restrained by flag diacritics.

In the next paragraph a long distance dependency is again resolved into a purely local phenomenon.

3 Russian Deverbal Nominalization with /nie/

In [12] a special case of blocking in Russian is described. Almost all simple or complex Russian verb stems as described above can combine with the deverbal nominalization suffix *nie*. See the following examples:

9	*pisát'* (write-ipf.)	*pisánie* (writing-N.)	RES/CEN
	pét' (sing-ipf.)	*pénie* (singing-N.)	SE/CEN
	sobrát' (collect-pf.)	*sobránie* (meeting-N.)	RES/SE
	starát's'a (try-ipf.)	*staránie* (endeavour-N.)	SE
	raspisát' (write out-pf.)	*raspisánie* (timetable-N.)	RES
	spisát' (write off-pf.)	*spisánie* (writing off-N.)	CEN
	zatverdét' (harden-pf.)	*zatverdénie* (hardening-N.)	RES/CEN
	izgotóvit' (manufacture-pf.)	*izgotovlénie* (manufacture-N.)	SE/CEN

The type of the resulting nominalization (adopted from [12] and given at the end of the line) is not predictable and can be a complex event nominal (CEN, e.g. *spisánie*), a simple event nominal (SE, e.g. *staránie*), a result nominal (RES, e.g. *raspisánie*), and one nominalization can also have different types. We will not provide more details about the semantic characteristics and the tests for different types of nominalizations because they are not crucial for the following discussion.

In [12] two generalizations are mentioned that apply to that kind of word formation. First, if nominalization applies the secondary imperfectivization of a lexically prefixed verb then the type of the nominalization is always the same, namely a complex event nominal. Second, if the secondary imperfectivization of a lexically prefixed word does not use the allomorph *yv* but instead the allomorph *a* then the deverbal nominalization suffix *nie* cannot be attached to that verb. As examples for that blocking they cite the following data from lexically prefixed verbs:

10	proclaim: proclamation:	*provozglasít'* (pf.) *provozglašénie*	*provozglašát'* (ipf.) **provozglašánie*
	visit: visit (N):	*posetít'* (pf.) *poseščénie*	*poseščát'* (ipf.) **poseščánie*
	inform: communication:	*soobščít'* (pf.) *soobščénie*	*soobščát'* (ipf.) **soobščánie*

consolidate:	*ukrepít'* (pf.)	*ukrepl'át'* (ipf.)
consolidation:	*ukreplénie*	**ukrepl'ánie*
destroy:	*razrušít'* (pf.)	*razrušát'* (ipf.)
destruction:	*razrušénie*	**razrušánie*
destroy:	*razorít'* (pf.)	*razor'át'* (ipf.)
destruction:	*razorénie*	**razor'ánie*
resolve:	*postávit'* (pf.)	*postavl'át'* (ipf.)
resolution:	*postanovlénie*	**postanovl'ánie*

Again this is a case of long distance dependency. The suffix *nie* has to have access to the inner structure of the morphologically complex verb.

In [12] this problem is discussed from the point of view of word syntax. According to their analysis such an approach in combination with locality conditions on affixation has to abuse feature marking and percolation conventions to "permit a purely morphological feature to percolate from the root to the top of the tree". Even then it does not explain blocking effects of deverbal nominalizations in Russian verbal morphology. It is argued that the generalization can only be stated by a morphological rule of referral.

In [13] it has already been shown how morphological frameworks using rules of referral can be fomulated in a finite-state framework. However, we would like to suggest another and much more simple analysis of the facts.

4 Renalysis of the Long Distance Dependency

A simple solution could again use flag diacritics. However, there is a much simpler approach. The crucial assumption in [12] is: "There is no purely phonological restriction which will account for the lack of **razrušánie*, **ukrepl'ánie* [...]."

But let us have a closer look at the data. The first thing to note is that if a verb contains stems with palatalized allomorphs in its paradigm then the deverbal suffix nie is always attached to such a stem allomorphs:

11	*izgotóvit'*	(manufacture-pf.)
	izgotóvl'u	(manufacture-1.Sg.pres.pf.)
	izgotovlénie	(manufacture-N.)
12	*provozglasít'*	(proclaim-pf.)
	provozglašú	(proclaim-1.Sg.pres.pf.)
	provozglašénie	(proclamation-N.)

The next thing to note is that the suffix nie is always added to a stem ending either in a or e, even though the stem to which the deverbal nominalization is added might never realize that vowel elsewhere in the paradigm:

13 *izgotóvit'* *izgotovlénie,* **izgotovlínie*
ukrepít' *ukreplénie,* **ukreplínie*

Our assumption then is that the vowel preceding *nie* in fact belongs to the nominalizing suffix. It is attached to a stem that does not have an thematic vowel and is realized as *a* after hard consonants and as *e* after soft consonants. The only morphological requirement for the application of the nominalization suffix is to take a special palatalized stem form if there is one in the paradigm. Everything else is governed by phonology:

14 /Vnie/ → [enie] / C[soft] _
/Vnie/ → [anie] elsewhere

This analysis can be stated in a more sophisticated phonological theory. Using underspecified feature structures, one possibility would be to say that the underlying vowel needs only to be specified for [LOW]. A postlexical rule deletes this feature in the context of a preceding soft consonant which is always specified for [HIGH] (see [14]). A vowel with no feature specification at all will per default be as realized as coronal [e]. With only specified for [LOW] after hard consonants, a redundancy rule will realize this vowel as dorsal [a].

There is one slight complication with sibilants in Russian. In Russian all consonants can have soft (with secondary palatalization) and hard variants (without secondary palatalization). The sibilants *š*, *ž* and the dental affricate *ts*, however, do not have both variants but always surface as hard consonants. On the other hand, in certain phonological contexts which are sensitive to the softness of the consonant the hard sibilants behave like soft consonants. This is true for the above mentioned rules of vowel alternation and for similar rules of stress-sensitive vowel neutralization. There is thus evidence that underlyingly they are soft. With that assumption one could explain why the form *provozglašénie* is encountered instead of *provozglašánie*. Again only a detailed phonological analysis will lead to these generalizations. The exact details of the Russian phonological system are quite elegant and straightforward but need not to be copied one by one into the *xfst*-framework. It suffices to know that the generalization to be captured is phonological. With these observations at hand one could now explain the following blocking effect:

15 *razrušít'* (destroy-pf.) *razrušénie* (destruction)
razrušát' (destroy-ipf.) **razrušánie*

16 *razorít'* (destroy-pf.) *razorénie* (destruction)
razor'át' (destroy-ipf.) **razor'ánie*

The reason why a form like **razoránie* is never encountered as opposed to the form *razorénie* is that there is a simple case of phonological neutralization at work. The underlying stem used for the formation of razorenie is /razor'/ which ends in a soft consonant. The vowel of the derverbal suffix undergoes a simple

assimilation – after soft consonants it is fronted and surfaces as [e] whereas elsewhere it surfaces as [a].

This phonological generalization holds also for verbs from the consonantal class which do not have imperfective suffixes but signal the secondary imperfectivization via the whole stem form:

17	*sobrát'*	(collect-pf.)	*sobirát'*	(collect-ipf.)
	sobránie	(collection)	*sobiránie*	(collecting-N)

18	*výžat'*	(wring out-ipf.)	*výžimat'*	(wring out-ipf.)
	not attested		*vyžimánie*	(wringing out-N)

The phonological generalization also applies to unprefixed verbs:

19	*pisát'*	(write-ipf.)	*pisánie*	(writing-N)
	bít'	(beat-ipf.)	*bijénie*	(beat-N)

The last piece of evidence comes from semantics. Normally, nominalizations with nie formed from perfective verbs do not show a predictable pattern of nominalization type, as pointed out in [12]. It is therefore interesting to note that nominalizations of lexcially prefixed verbs where the secondary imperfectivization uses the allomorph a almost always have a complex event reading besides a simple event reading or result reading. The explanation is easy in our analysis – since the nominalization of these verbs can have two potential underlying stems (perfective and imperfective) and since the secondary imperfectives show a regular pattern of nominalization type this generalization is preserved independent of the phonological neutralization.

In sum, with another morphological segmentation the effect of blocking turns into a case of phonological neutralization.

5 Conclusion

We have demonstrated how linguistic analyses can be simplified when seen from different perspectives. It remains an interesting but open question whether similar phenomena can also be reanalyzed in this way and what repercussions a special linguistic theory has for engineering aspects.

6 Acknowledgements

I would like to thank Annette Hautli for her inspiring suggestions and her help with the formalization of the analysis. Miriam Butt kindly provided me the opportunity to engage more in finite-state research. Special thanks to Thomas Mayer for discussing the topics with me and for his patient help with the formatting. Finally, I would like to thank two anonymous reviewers for their helpful comments on an earlier version of this paper. O course, I take the full responsibility for all possible errors.

This research was partially supported by the SFB 471 "Variation und Entwicklung im Lexikon" at the University of Konstanz and university research funds for Aditi Lahiri.

References

1. Beesley, K.R., Karttunen, L.: Finite State Morphology. Stanford, California: CSLI Publications (2003)
2. Lehmann, V.: Aspekt. In Jachnow, H., ed.: Handbuch der sprachwissenschaftlichen Russistik und ihrer Grenzdisziplinen. Harrassowitz, Wiesbaden (1999) 214–242
3. Timberlake, A.: A Reference Grammar of Russian. Cambridge University Press (2004)
4. Isacenko, A.: Die russische Sprache der Gegenwart – Formenlehre. fourth edn. Max-Hueber-Verlag, München (1995)
5. Spencer, A.: Morphological Theory. Blackwell Publishers, Oxford (1998)
6. Breu, W.: Interactions between lexical, temporal and aspectual meanings. Studies in Language **18**(1) (1994) 23–44
7. Breu, W.: Zur Position des Slavischen in einer Typologie des Verbalaspekts (Form, Funktion, Ebenenhierarchie und lexikalische Interaktion). In Breu, W., ed.: Probleme der Interaktion von Lexik und Aspekt (ILA). Linguistische Arbeiten, Tübingen (2000) 21–54
8. Williams, E.: On the notions 'lexically related' and 'head of a word'. Linguistic Inquiry **12** (1981) 245–274
9. Zwicky, A.: Heads. Journal of Linguistics (21) (1985) 1–20
10. Beesley, K.R.: Constraining separated morphotactic dependencies in constraining separated morphotactic dependencies in finite-state grammars. In Karttunen, L., Oflazer, K., eds.: FSMNLP'98. Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, Ankara, Turkey, Bilkent University (1998)
11. Lahiri, A.: Phonology: Structure, representation, and process. In Wheeldon, L., ed.: Aspects of language production. Psychology Press, Hove (2000) 165–225
12. Sadler, L., Spencer, A., Zaretskaya, M.: A morphomic account of a syncretism in Russian deverbal nominalizations. In Booij, G., van Marle, J., eds.: Yearbook of Morphology. Kluwer Academic Publishers, Dordrecht (1996) 181–215
13. Karttunen, L.: Computing with realizational morphology. In Gelbukh, A., ed.: Computational linguistics and intelligent text processing. Volume 2588 of Lecture notes in computer science. Springer, Berlin/Heidelberg (2003) 205–216
14. Lahiri, A., Evers, V.: Palatalization and coronality. In Paradis, C., Prunet, J.F., eds.: Phonetics and Phonology 2. Academic Press, Inc. (1991) 79–100

Transducers from Parallel Replace Rules and Modes with Generalized Lenient Composition

Anssi Yli-Jyrä

Department of General Linguistics, University of Helsinki, Finland

Abstract. Generalized Two-Level Grammar (GTWOL) provides a new method for compilation of parallel replacement rules into transducers. The current paper identifies the role of *generalized lenient composition* (GLC) in this method. Thanks to the GLC operation, the compilation method becomes bipartite and easily extendible to capture various application modes. In the light of *three notions of obligatoriness*, a modification to the compilation method is proposed. We argue that the bipartite design makes implementation of parallel obligatoriness, directionality, length and rank based *application modes* extremely easy, which is the main result of the paper.

1 Introduction

It is extremely difficult to compile grammars into finite-state transducers without efficient and readily implemented compilation methods for high-level rules. In particular, replace rules (such as [1]) have a rich semantics that is difficult to capture. The goal of this paper is to analyze the author's recently proposed method [2] and the related approach in general.[1]

The new method [2] differs from the most similar alternative approach of Kempe and Karttunen [1] in some obvious ways:

- It reduces oriented replace rules to two-level rules
- It does not necessarily use composition
- It derives all modes from optional replacement
- Its left-and-right context conditions are closed under Boolean operations
- It uses brackets only to avoid overlapping rule applications.

In this paper, perhaps the most important contribution is the recognition of the relevance of the *bipartite architecture* of the new method. According to it, the rule-independent mode constraints are separated from rule-specific condition. Related to this, we present the necessary machinery including Jäger's composition operator [3] and new strict preference relations. The second important contribution is to present *Bracketed Generalized Two-Level Grammar* (BGTWOL) that is crucial to the new compilation method. The third contribution is to separate *three modes of obligatoriness*. A clear understanding of these

[1] For further resources http://www.ling.helsinki.fi/users/aylijyra/replace.

modes helps relate the existing compilation methods and improve the compatibility of the new method and the Xerox calculus. Finally, the paper sketches a *rich rule system* that covers the multi-character two-level rules of GTWOL [4, 5] and BGTWOL, parallel replace and marking rules [1, 6], directed modes [7] and three principles for ranking [8] or disjunctive ordering [5].

The paper is structured as follows: Preliminary definitions are in Section 2. In Section 3, we describe the essentials of Generalized Two-Level Grammar (GTWOL) [4]. Section 4 reduces replace operations into the GTWOL formalism. Section 5 studies applications of generalized lenient composition to obligatory replacement. The new design pattern for compilation methods is discussed in Section 6. The conclusion is in Section 7.

2 Preliminaries

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 $U^0 = \epsilon$ and let U^k, where $k > 0$, denote the languages $UU^{(k-1)}$.

The *local A_2-closure* of $U \subseteq {A_1}^*$ is the relation $f_{A_2}{:}{A_1}^* \rightarrow {A_1}^*$ defined as $f_{A_2}(U) = \{f(a_0)f(a_1)\dots f(a_{m-1}) \mid a_0a_1\dots a_{m-1} \in U \wedge a_0, a_1, \dots, a_{m-1} \in A_1\}$ 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 $d_{A_2}(U) = f_{A_2}(U) \backslash {A_1}^* A_2 {A_1}^*$. The inverse of relation d_{A_2} is denoted by ${d_{A_2}}^{-1}$.

Notation $A_1{:}A_2$ denotes alphabet $\{a_1{:}a_2 | a_1 \in A_1 \wedge a_2 \in A_2\}$. 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 $\texttt{\#:\#}, \_{:}\_, \diamond_0{:}\diamond_0, \diamond_1{:}\diamond_1, \diamond_2{:}\diamond_2, \dots, \diamond_s{:}\diamond_s$ and it is disjoint from Π. The indices of the diamonds will be used to indicate the disjunctive ordering level of GTWOL rules. Level 1 is the level of the least specific rules. An identity pair $a{:}a \in (\Pi \cup M)$ is often written simply as a.

We use marker $\_ \in M$ to represent the place for centers in an environment string. The *center extension* with $V \subseteq {A_1}^*$ is the relation $\sigma_V{:}(A_1 \cup \{\_\})^* \rightarrow {A_1}^*$ defined as $\sigma_V(U) = \{\sigma(a_0)\sigma(a_1)\dots\sigma(a_{m-1}) \mid a_0a_1\dots a_{m-1} \in U \wedge a_0, a_1, \dots, a_{m-1} \in A_1 \cup \{\_\}\}$ where $\sigma(a)=V$ when $a = \_$, and $f(a)=a$ otherwise.

The null string is denoted by ϵ. Let u be a string over an alphabet A_1. We often denote set $\{u\}$ by u. The length of u is denoted by $|u|$. A sequence $u = a_0{:}b_0a_1{:}b_1\dots a_{m-1}{:}b_{m-1} \subseteq (A_1{:}A_2)^*$ is called a *symbol-pair string* and analyzed alternatively as a string pair $(x_1, x_2) = (a_0a_1\dots a_{m-1}, b_0b_1\dots b_{m-1})$. Pair (x_1, x_2) can be denoted by $x_1{:}x_2$ when $|x_1| = |x_2|$. String 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 symbols $<_1, <_2, \dots, <_s$, and set B_R contains symbols $>_1, >_2, >, \dots, >_s$. Let $B =$

$B_L \cup B_R$ and $B_i = \{<_i, >_i\}$. The indices of the brackets will be used to denote the ranking level of a ranked rule.

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

Let $U_2 = \Pi^* M \Pi^* M \Pi^*$. Define relations $\nu_{\star,l}, \nu_{2,l} : \Pi^* \rightarrow (\Pi \cup M)^*$ by equations $\nu_{\star,l}(w) = {d_{\{\diamond_1, \diamond_2, \ldots, \diamond_l\}}}^{-1}(w)$ and $\nu_{2,l}(w) = \nu_{\star,l}(w) \cap U_2$, and relations $\mu, \mu_4 : (\Pi \cup M)^* \rightarrow (\Pi \cup M)^*$ by equations $\mu(w) = \{\#v\nu_{\star,l}(x)y\# \mid \diamond_j \in M \wedge v, x, y \in \Pi^* \wedge \#v\diamond_j x \diamond_j y\# \in W\}$ and $\mu_4(w) = \mu(w) \cap \#U_2\#$.

Let $W, W' \in (\Pi \cup M)^*$. The language $\Pi^* \backslash d_M(W \setminus W')$ is denoted by *generalized restriction* $W \overset{\Pi,4,M}{\Rightarrow} W'$, if $W \subseteq \#U_2\#$, and by *extended generalized restriction* $W \overset{\Pi,\mu,M}{\Rightarrow} W'$, if $W{=}\mu(Y)$ and $W'{=}\mu(Y')$ for some $Y, Y' \subseteq \#U_2\#$.

It holds that $[W \overset{\Pi,4,M}{\Rightarrow} \mu_4(W')] = [W \overset{\Pi,4,M}{\Rightarrow} \mu(W')]$ and $[\mu_4(W) \overset{\Pi,4,M}{\Rightarrow} \mu(W')] = [\mu(W) \overset{\Pi,\mu,M}{\Rightarrow} \mu(W')]$. Accordingly, $[\mu_4(W) \overset{\Pi,4,M}{\Rightarrow} \mu_4(W')] = [\mu(W) \overset{\Pi,\mu,M}{\Rightarrow} \mu(W')]$.

3 Generalized Two-Level Grammars

The formalism of Generalized Two-Level Grammars (GTWOL) [4, 5] presents several improvements over the classical Two-Level formalism [9, 10] in computational morphology. Its main improvement is to support multi-character changes while not turning the formalism into so-called partition-based two-level system which would behave quite differently. Since Yli-Jyrä [5] adds disjunctive ordering to the definition of GTWOL grammars, we will use the same notation here. However, we adopt in this paper an extended notion of the GR operation.

Simple and Complex Rules For any $i \in \mathbb{N}$, let X_i, L_i and R_i denote regular languages over Π, and let l_i be a positive integer. The GTWOL formalism [4, 5] includes *center prohibition rule* $[l_i :: X_i$/<=L_i_$R_i]$, *context restriction rule* $[l_i :: X_i$=>L_i_$R_i]$, *surface coercion rule* $[l_i :: X_i$<=L_i_$R_i]$, and *composite i.e. double-arrow rule* $[l_i :: X_i$<=>L_i_$R_i]$ that is a short-hand notation for a context restriction rule and a surface rule. The symbols _ and # belong to the diamond alphabet M. Each context condition $C_i = \#L_i$_$R_i\# \subseteq \#\Pi^*$_$\Pi^*\#$ can be represented by a weaker form $C'_i \subseteq (\epsilon \cup \#)\Pi^*$_$\Pi^*(\epsilon \cup \#)$ that is related to C_i by the following equivalence:

$$C_i = [(\epsilon \cup \#\Pi^*)C'_i(\epsilon \cup \Pi^*\#)] \cap (\#\Pi^*\_\Pi^*\#). \tag{1}$$

In other words, the following syntactic conventions are implemented: ..._... $\Leftrightarrow$ ...ϵ_ϵ...; #Π^*L_... $\Leftrightarrow$ L_...; ..._$R\Pi^*\# \Leftrightarrow$ ..._R. The GTWOL formalism supports rules that have multiple contexts or, more generally, even Boolean combinations of two-sided context conditions, because these context conditions are actually languages.

Let the set of rule operators O contain symbols `/<=`, `<=`, `=>`, `<=>`. The rule types have a general form $X_i\ op_i\ C_i$, where $X_i \in \Pi^*$, $op_i \in O$, and $C_i \subseteq$ `#`Π^*`_`Π^*`#`.

The Individual Rules of GTWOL The semantics of the individual rules of GTWOL grammar is defined as follows:

$$[l_i :: X_i \texttt{/<=} C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_{\star,l_i}(X_i)}(C_i) \overset{\Pi,\mu,M}{\Rightarrow} \emptyset] \tag{2}$$

$$[l_i :: X_i \texttt{=>} C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_{\star,l_i}(X_i)}(\texttt{\#}\Pi^*\texttt{\_}\Pi^*\texttt{\#}) \overset{\Pi,\mu,M}{\Rightarrow} \sigma_{\nu_{\star,l_i}(X_i)}(C_i)] \tag{3}$$

$$[l_i :: X_i \texttt{<=} C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_{\star,l_i}(\pi_1{}^{-1}(\pi_1(X_i)))}(C_i) \overset{\Pi,\mu,M}{\Rightarrow} \sigma_{\nu_{\star,l_i}(X_i)}(C_i)] \tag{4}$$

$$[l_i :: X_i \texttt{<=>} C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_{\star,l_i}(\pi_j{}^{-1}(\pi_j(X_i)))}(C_i) \cup \sigma_{\nu_{\star,l_i}(X_i)}(\texttt{\#}\Pi^*\texttt{\_}\Pi^*\texttt{\#}) \overset{\Pi,\mu,M}{\Rightarrow} \sigma_{\nu_{\star,l_i}(X_i)}(C_i)]. \tag{5}$$

In contrast to [4], the generalized postconditions specify now immediately the successful rule applications (like S_i later in [4]).

Since the original GTWOL [4], context restriction rules have had both licensing and restricting functions because of the longest application principle [4, 5]. While the problematic left-arrow rules with empethesis [14] were addressed [4], the double function of context restriction rule [(`a` ∪ `a:b`)`*=>c_d`] restricted the occurrences of such substrings as ϵ, `a` and `aa`. The currently updated GTWOL contains a *default core* $\textsc{Gen}_{\text{core}}$ of two rules: rule $[1 :: \Pi\texttt{=>}\emptyset]$ says that every symbol in strings needs to be licensed, and rule $[1 :: I^*\texttt{=>\_}]$ says that all substrings consisting of identity pairs are licensed. The latter default rule is now in an intended conflict with [1 :: `a*=>c_d`].

Coherent Intersection An important aspect of GTWOL is how it combines rules. In the classical Two-Level Grammar, rules are compiled in separation and then combined under intersection, whereas GTWOL can combine rules *before* they are compiled. The operation $\pitchfork$ under which the rules are combined is introduced in [5], and it is called *coherent intersection*.

$$[W_1 \overset{\Pi,\mu,M}{\Rightarrow} W'_1] \pitchfork [W_2 \overset{\Pi,\mu,M}{\Rightarrow} W'_2] \stackrel{\text{def}}{=} [(W_1 \cup W_2) \overset{\Pi,\mu,M}{\Rightarrow} ((W_1 \cap W'_1) \cup (W_2 \cap W'_2))] \tag{6}$$

Let G be a collection of GTWOL rules that use alphabet Π. When all rules are combined under the coherent intersection, the grammar reduces to a single generalized restriction $W \overset{\Pi,\mu,M}{\Rightarrow} W'$ that returns the language described by G. This language is denoted by $\textsc{Gen}_G$.

Coherent intersection implements conflict resolution for various kinds of arrow conflicts [11, 4, 5]. In addition, two further resolution strategies follow from the definition of $\nu_{\star,l}$: conflicts between embedded rule applications are resolved on the basis of the *longest application* principle [4] and *disjunctive ordering*

of the levels [5]. Disjunctive ordering uses alternative diamonds $\diamond_1, \ldots \diamond_s$. The disjunctive level denoted by $\diamond_1$ is the least general one. Rules at level strictly greater than 1 use several alternative diamonds. However, partially overlapping applications are not resolved automatically and rules with shorter applications cannot override rules with strictly longer applications. This is where GTWOL will continue to mature.

Most GTWOL rules are stored at the level 1. Therefore, we can abbreviate such rules by leaving out their level specifications.

Bimorphisms Defined by GTWOLs Bimorphisms [12] are a useful notion that can be combined with generalized restriction [5]. 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 $\beta(P) = \{(\psi_1(w), \psi_2(w)) \mid w \in P\}$.

Let $\textsc{Gen}_G \subseteq \Pi^*$ be a language described by a two-level grammar. According to bimorphism $(\pi_1, \textsc{Gen}_G, \pi_2)$, this language defines a regular relation $\beta_1(\textsc{Gen}_G)$ where $\beta_1(P) = \{(\pi_1(w), \pi_2(w)) \mid w \in P\}$ [9, 4].

4 Reduction of Replace Rules into Two-Level Grammars

In the literature, a diverse variety of algorithms have been proposed as solutions to compilation of oriented, inverted, directed, parallel, and ranked replacement and marking rules [6, 7, 1, 13]. In order to integrate different rule types and their compilation methods, we relate them to Generalized Two-Level Grammar that provides a good basis for representation of conditions of individual rules.

Centers The heart of a usual replacement rule is the *description of change*, or the *center*, that consists of *two regular languages*, $U \subseteq \Sigma^*$ and $Y \subseteq \Sigma^*$, meaning that a substring in U will be replaced disjunctively with replacements that are picked from set Y. The separate description of U and V is motivated by the usual rule formats in production systems and it may be easier to read. Some rules *e.g.* in Generative Phonology contain backreferences that are normally expressed with feature variables. According to Kaplan and Kay [14], such rules could be split into a number of subrules.

However, it is arguable [15] that if the centers are defined as *regular relations* we obtain a more expressive and useful definition that includes, for example, marking rules. Therefore, we will specify the center X_i directly as a same-length relation *i.e.* a language over Π. In fact, there are various ways to obtain an adequate X_i from languages U_i and Y_i, if needed. If X_i is obtained adequately, cross product $U_i \times Y_i$ equals to $\beta_1(X_i)$. Rules that contain a *list of centers* $X_1, X_2, \ldots X_n$ reduce now to union $\cup_{i=1}{}^n X_i$. Moreover, the center of the *marking rules* [7, 13] can be expressed easily as a subset of Π^*. For example, the description of change in the marking rule [`a+ -> b e g ... e n d`] in XFST [13] corresponds to two-level center `0:b 0:e 0:g` $\mathtt{a}^+$ `0:e 0:n 0:d`.

Oriented Contexts Both replace (and marking) rules can be conditional [6, 7, 13] *i.e.* restricted to apply only in certain contexts. The context conditions of these rules can be reduced into GTWOL context conditions easily. For consistent presentation, assume that boundary markers `.#.` [13] and `#` are synonymous.

The previous implementations of replace rules express each context condition C_i as a language $\#L_i \_ R_i \#$, where $L_i, R_i \subseteq \Sigma^*$. For convenience, each such context condition can be represented in a weaker form $C'_i \subseteq (\epsilon \cup \#)\Sigma^* \_ \Sigma^*(\epsilon \cup \#)$ that is related to C_i by the following equivalence:

$$C_i = [(\epsilon \cup \#\Sigma^*)C'_i(\epsilon \cup \Sigma^*\#)] \cap (\#\Sigma^* \_ \Sigma^*\#). \tag{7}$$

In contrast to two-level contexts that are subsets of $(\Pi \cup M)^*$, the replace rules restrict their context conditions to languages over $(\Sigma \cup M)^*$. This is due to the fact that these contexts have four possible orientations: *left-to-right, right-to-left, upward* and *downward*. If the context condition C_i of the replace rule is left-to-right (or right-to-left), it is interpreted as a combination of a look-a-head condition R_i (L_i) in the input string and a trailer condition L_i (R_i) in the output string. Conditions with either upward or downward orientation are simpler and they check either the input or the output string, respectively.

In Generative Grammar, the slash character / is conventionally used to separate the description of change and the context condition [16]. In the replace formalism [6], the specific form of this separator $s_i \in \{//, \backslash\backslash, ||, \backslash/\}$ indicates also the orientation of the context. The oriented context condition $s_i C_i$ corresponds to a two-level context condition

$$C_i = \begin{cases} \{x_1{:}x_2 \in \Pi^* \mid x \in d_0{}^{-1}(C_i) \wedge x_1{:}x \in \#\Pi^* \_ \Sigma^*\# \wedge x{:}x_2 \in \#\Sigma^* \_ \Pi^*\#\}, & \text{if } s_i = '//'; \\ \{x_1{:}x_2 \in \Pi^* \mid x \in d_0{}^{-1}(C_i) \wedge x_1{:}x \in \#\Sigma^* \_ \Pi^*\# \wedge x{:}x_2 \in \#\Pi^* \_ \Sigma^*\#\}, & \text{if } s_i = '\backslash\backslash'; \\ \{\ x{:}x_2 \in \Pi^* \mid x \in d_0{}^{-1}(C_i)\}, & \text{if } s_i = '||'; \\ \{x_1{:}x\ \in \Pi^* \mid x \in d_0{}^{-1}(C_i)\}, & \text{if } s_i = '\backslash/'. \end{cases} \tag{8}$$

The reduction lends itself for a simple finite-state implementation *e.g.* by using composition or a simpler *ad hoc* algorithm. Given the reductions (7) and (8), a typical weak replacement context condition such as `// c_d` is considered a two-level context $\#\Pi^* \pi_2{}^{-1}(\mathtt{c}) \_ \pi_1{}^{-1}(\mathtt{d})\Pi^*\#$.

The subsets of $\#\Pi^* \_ \Pi^*\#$ are obviously closed under the Boolean operations. As we have now reduced all context conditions into these sets, we can combine contexts with different orientations under intersection, asymmetric difference and symmetric difference. Accordingly, we capture more than the usual possibilities [13] with considerable ease.

Two-Level Operators for Replace Modes When the center X_i and the context condition C_i are both in the two-level format, it is natural to introduce a flexible rule syntax for parallel rules. On one hand, centers and context conditions can both be combined with Boolean operations. On the other, the extended generalized restrictions obtained from each parallel rule can be combined under

coherent intersection. Parallel rules can be indicated in the rule formalism in different ways. One possibility is the XFST notation:

$$X_1\ op_1\ C_1 \text{„}\ X_2\ op_2\ C_2 \text{„}\ \ldots \text{„}\ X_n\ op_n\ C_n. \tag{9}$$

In XFST, the rule operators are used to indicate the mode of application. Alternative operators include `(->)`, `->`, `<-`, `<->`, `@->`, `->@`, `@>`, and `>@`. These indicate respectively the optional, obligatory, inverse, bidirectional, longest left-to-right, longest right-to-left, shortest left-to-right, and shortest right-to-left replacement modes.

In order to account for different replacement modes compactly, it is useful to understand what aspects they have in common and which mode could be used as a primary notion for obtaining the others.

4.1 Overlapping *vs.* Non-Overlapping Applications

In GTWOL, rules such as [1 :: `a:b=>c_d`] are actually very similar to optional replacement rules [4, 5]. Provided that the rule neither overlaps nor interacts with itself or any other rule than the default rules, the semantics of context restriction actually coincides with optional replacement. However, a self-overlapping context restriction [1 :: `a:b a:b=>_`] and optional replace `aa(->)bb` are not interchangeable (consider *e.g.* the input `aaa`). And due to the overlaps, context restriction rules [1 :: `a:b=>x_x`] and [1 :: `x a:b x=>_`] do not differ when considered in isolation [2]: both would accept the symbol-pair string `x a:b x a:b x`. However, the contributions of these rules differ under coherent intersection because the latter rule has a longer center.

Double-arrow rules are the classical way to express obligatoriness in two-level grammars. They involve a right-arrow rule and a left-arrow rule. However, the resulting notion of obligatoriness is quite strict (denoted by **M1**), because the combination of such left-arrow rules as [1 :: `A:a B:p<=_`] and [1 :: `B:b C:c<=_`] rejects the input `ABC`. The consequences `B:p` and `B:b` generate a conflict that is not solved automatically by the current GTWOL.

Kaplan and Kay [14] underlines that phonological rules do not normally rewrite their own output. This does not refer to overlapping simultaneous rule applications at the first place but such directed rewriting that does not advance monotonously in the original input string but resume, after an application, an earlier string position in the modified string. Anyway, overlapping applications does not belong to replace rules such as [1].

The interpretations of the optional replace rules [1] and right-arrow rules on one hand, and obligatory replace and double-arrow rules, on the other, will coincide if the center is free from self-overlaps and self-embeddings. Thus, replacement rules should somehow be reduced to overlap-free GTWOL grammars.

4.2 A Bracketed GTWOL

We use term *Bracketed Generalized Two-level Grammar* (BGTWOL) to refer to a loosely characterized family of such GTWOL grammars that assume a non-empty sub-alphabet $B \subseteq I$ and use it to indicate bracketing in the strings of

language $\textsc{Gen}_G$. The default core $\textsc{Gen}_{\text{core}}$ is now $[1 :: \Pi \texttt{=>} \emptyset] \pitchfork [1 :: \Sigma^* \texttt{=>} \_]$, because now alphabet $B \not\subseteq \Sigma$ is reserved for a special use and the user does not have a normal access to it.

Let G be a BGTWOL grammar. The language $\textsc{Gen}_G$ described by grammar G is used as the pivot in bimorphism $(\psi_1, \textsc{Gen}_G, \psi_2)$ where $\psi_1(w) = \pi_1(d_B(w))$ and $\psi_2(w) = \pi_2(d_B(w))$. In this bimorphism, the grammar describes the regular relation $\beta_2(\textsc{Gen}_G)$ where $\beta_2(P) = \{(\pi_1(d_B(w)), \pi_2(d_B(w)))|w \in P\}$.

Bracketed Grammar Rules In addition to the disjunctive ordering of rules, BGTWOL involves another ranking mechanism that is based on bracket labels. It is, however, not really used before Section 6. For all $i = 1, 2, \ldots$, let $X_i \subseteq (\Pi \backslash B)^*$ and $C_i \subseteq \texttt{\#}(\Pi \backslash B)^* \_ (\Pi \backslash B)^* \texttt{\#}$ be regular languages, and let $X_i{}' = \texttt{<}_{b_i} X_i \texttt{>}_{b_i}$, $C_i{}' = {d_B}^{-1}(C_i)$, $\Delta_0 = (\Pi \backslash B)^*$ and $\Delta_1 = \Delta_0 (B_L \Delta_0 B_R \Delta_0)^*$.

BGTWOL supports some new rule types that include *bracketed coercion* $[l_i :: \texttt{<}_{b_i} X_i \texttt{>}_{b_i} \texttt{(<=)}\ C_i]$, *bracketed inverse coercion* $[l_i :: \texttt{<}_{b_i} X_i \texttt{>}_{b_i} \texttt{(=<)}\ C_i]$, *bracketed context restriction* $[l_i :: \texttt{<}_{b_i} X_i \texttt{>}_{b_i} \texttt{(=>)}\ C_i]$, *bracketed double-arrow* $[l_i :: \texttt{<}_{b_i} X_i \texttt{>}_{b_i} \texttt{(<=>)}\ C_i]$, *bracketed inverse double-arrow* $[l_i :: \texttt{<}_{b_i} X_i \texttt{>}_{b_i} \texttt{(=<>)}\ C_i]$, and *bracketed bidirectional double-arrow* $[l_i :: \texttt{<}_{b_i} X_i \texttt{>}_{b_i} \texttt{(<=<>)}\ C_i]$. These operations are defined as follows:

$$[l_i :: X'_i\ \texttt{(<=)}\ C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_\star, l_i}({\pi_1}^{-1}(\pi_1(d_B(X'_i)))) (C'_i \cap \texttt{\#}\Delta_1 \_ \Delta_1 \texttt{\#})\ {}^{\Pi,\mu,M}\!\Rightarrow \emptyset] \quad (10)$$

$$[l_i :: X'_i\ \texttt{(=<)}\ C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_\star, l_i}({\pi_2}^{-1}(\pi_2(d_B(X'_i)))) (C'_i \cap \texttt{\#}\Delta_1 \_ \Delta_1 \texttt{\#})\ {}^{\Pi,\mu,M}\!\Rightarrow \emptyset] \quad (11)$$

$$[l_i :: X'_i\ \texttt{(=>)}\ C_i] \stackrel{\text{def}}{=} [l_i :: X'_i \texttt{ => } C'_i] \quad (12)$$

$$[l_i :: X'_i \texttt{(<=>)} C_i] \stackrel{\text{def}}{=} [l_i :: X'_i \texttt{(<=)} C_i] \pitchfork [l_i :: X'_i \texttt{(=>)} C_i] \quad (13)$$

$$[l_i :: X'_i \texttt{(=<>)} C_i] \stackrel{\text{def}}{=} [l_i :: X'_i \texttt{(=<)} C_i] \pitchfork [l_i :: X'_i \texttt{(=>)} C_i] \quad (14)$$

$$[l_i :: X'_i \texttt{(<=<>)} C_i] \stackrel{\text{def}}{=} [l_i :: X'_i \texttt{(<=>)} C_i] \pitchfork [l_i :: X'_i \texttt{(=<>)} C_i]. \quad (15)$$

Bracketed coercion bears functional similarity to surface coercion. It says intuitively that the center X_i that is non-embedded (*i.e.* $\texttt{\#}\Delta_1 \_ \Delta_1 \texttt{\#}$) must not be left unbracketed in the specified contexts.

Applications of bracketed surface coercion can overlap one another, but even the first application suffices to reject the pair-string and is normally not cancelled by other GTWOL rules. Meanwhile, applications of bracketed context restrictions cannot be embedded or overlapping because X_i does not contain brackets. Accordingly, we can give for the operator a simpler, purely licensing definition that looks like a tautology but still contributes against the default rule $[1 :: \Pi \texttt{ => } \emptyset]$ under coherent intersection.

$$[l_i :: X'_i\ \texttt{(=>)}\ C_i] \stackrel{\text{def}}{=} [\sigma_{\nu_\star, l_i}(X'_i)(C'_i)\ {}^{\Pi,\mu,M}\!\Rightarrow \sigma_{\nu_\star, l_i}(X'_i)(C'_i)]. \quad (16)$$

Optional Replace Rules Yli-Jyrä and Koskenniemi [2] observe that parallel conditional optional replace rules can be represented using context restriction in GTWOL. In the current terms, the representation uses bimorphism

$(\psi_1, \textsc{Gen}_G, \psi_2)$ where the pivot $\textsc{Gen}_G$ is obtained by changing the replace rules into *bracketed context restrictions*:

$$[X_1\,\texttt{(->)}\,C_1\texttt{„}\;\; X_2\,\texttt{(->)}\,C_2\texttt{„}\;\; \ldots\texttt{„}\;\; X_n\,\texttt{(->)}\,C_n] \stackrel{\text{def}}{=} \\ \beta_2\,(\textsc{Gen}_{\text{core}} \Cap \Cap_{i=1}{}^{n}[1 :: \texttt{<}_1 X_i \texttt{>}_1\,\texttt{(=>)}\,C_i]) \tag{17}$$

Note that the brackets indicate the regions where a rule has been correctly applied. This is a quite different approach than the multiplicity of brackets that are used in [1] to indicate partial satisfaction of conditions for rule application. For example, pivot language $\textsc{Gen}_G$ obtained from optional BGTWOL replace (that corresponds to rule [`ab (->) x // ab _b`] in Karttunen's [6] formalism)

$$[1 :: \texttt{a:x b:0}\;\; \texttt{(->)}\;\; \texttt{\#}\Pi^*\pi_2{}^{-1}(\texttt{a b})\_\;\pi_1{}^{-1}(\texttt{a})\Pi^*\texttt{\#}] \tag{18}$$

contains exactly the following mappings for input string `abababa`:

$$\begin{array}{lll} \text{(19a)} & \text{(19b)} & \text{(19c)} \\ \texttt{abababa} & \texttt{ab<}_1\texttt{ab>}_1\texttt{aba} & \texttt{abab<}_1\texttt{ab>}_1\texttt{a} \\ \texttt{abababa}, & \texttt{ab<}_1\texttt{x0>}_1\texttt{aba}, & \texttt{abab<}_1\texttt{x0>}_1\texttt{a}. \end{array} \tag{19}$$

Obligatory Replace Rules For the sake of compatibility to the Xerox calculus [13], it is desirable to pursue the semantics of obligatory conditional parallel replace such as in [6]. This can be done using *bracketed double-arrow* rules.

$$[X_1\,\texttt{->}\,C_1\texttt{„}\;\; X_2\,\texttt{->}\,C_2\texttt{„}\;\; \ldots\texttt{„}\;\; X_n\,\texttt{->}\,C_n] \stackrel{\text{def}}{=} \\ \beta_2\,(\textsc{Gen}_{\text{core}} \Cap \Cap_{i=1}{}^{n}[2 :: \texttt{<}_1 X_i \texttt{>}_1\,\texttt{(<=>)}\,C_i])\,. \tag{20}$$

Because the substrings undergoing a change are indicated by brackets, it is easy to enforce that a substring must be changed whenever the conditions for the replacement are met. This requirement is contributed by the bracketed coercion. Its disjunctive ordering level is now 2, because the default rule $[1 :: \Sigma^*\texttt{=>}\_]$ would cancel the effect of bracketed coercion $[1 :: \texttt{<}_1 X_i \texttt{>}_1\,\texttt{(<=)}\,C'_i]$ at level 1.

In inverse replacement (denoted by `<-`), the roles of the input and output strings are switched. The bidirectional obligatory replacement requires bracketed bidirectional double-arrow (*i.e.* `(<=<>)`).

5 Violable Mode Constraints

Although BGTWOL provides a solution to obligatory replacement through the bracketed double-arrow, we will now compare the solution to the new method of Yli-Jyrä and Koskenniemi [2]. For this purpose, we introduce some additional machinery.

5.1 Strict Preference Relations

A binary relation $T \subseteq I^* \times I^*$ is a *strict preference relation (SPR)* if it is irreflexive (thus not complete), transitive and antisymmetric. The following relations and their inverses are strict preference relations:

$$T_{\text{most}} = \{(\pi_1(w), \pi_2(w)) | w \in (B_L{:}0\ \Sigma^*\ B_R{:}0 \cup \Sigma \cup B{:}B)^*\} \tag{21}$$
$$T_{\text{most}+} = \{(\pi_1(w), \pi_2(w)) | w \in (B_L{:}0\ \Sigma^+ B_R{:}0 \cup \Sigma \cup B{:}B)^*\} \tag{22}$$
$$T_{\text{norep}} = \{(w, w') | w, w' \in I^* \wedge d_B(w) = d_B(w') \wedge w \notin (I^* B_L B_R B_L B_R I^*) \ni w'\} \tag{23}$$
$$T_{\text{lest}} = \{(w, w') | w, w' \in I^* \wedge d_B(w) = d_B(w') \wedge w \notin (I^* B I^*) \ni w'\} \tag{24}$$
$$T_{\text{lr}} = \{(vby, vau) | v, y, u \in I^* \wedge a \in \Sigma \wedge b \in B_L \wedge d_B(y) = d_B(au)\} \tag{25}$$
$$T_{\text{rl}} = \{(ybv, uav) | v, y, u \in I^* \wedge a \in \Sigma \wedge b \in B_R \wedge d_B(y) = d_B(ua)\} \tag{26}$$
$$T_{\text{lrlong}} = \{(vau, vby) | v, u, y \in I^* \wedge a \in \Sigma \wedge b \in B_R \wedge d_B(y) = d_B(au)\} \tag{27}$$
$$T_{\text{rllong}} = \{(uav, ybv) | v, u, y \in I^* \wedge a \in \Sigma \wedge b \in B_L \wedge d_B(y) = d_B(ua)\} \tag{28}$$
$$T_{\alpha,B'} = \{(w, w') | w, w' \in I^* \wedge (d_{B \setminus B'}(w), d_{B \setminus B'}(w')) \in T_\alpha\}. \tag{29}$$

Let T be an SPR. According to T, element $x_1 \in I^*$ is interpreted strictly more preferable than $x_2 \in I^*$, *i.e.* $x_1 \prec x_2$, if and only if $(x_1, x_2) \in T$. For example, $T_{\text{most}+}$ compares only compatible bracketings and prefers those that have more markup:

$$\texttt{aa<}_1\texttt{ab>}_1\texttt{<}_2\texttt{ab>}_2\texttt{a} \prec \{\ \texttt{aaab<}_1\texttt{ab>}_1\texttt{a},\ \texttt{aa<}_1\texttt{ab>}_1\texttt{aba}\ \} \prec\ \texttt{aababa};$$
$$\{\texttt{<}_1\texttt{aa>}_1\texttt{<}_1\texttt{aa>}_1,\ \texttt{a<}_1\texttt{aa>}_1\texttt{a}\} \prec\ \texttt{aaaa}.$$

Let us prove that $T_{\text{most}+}$ is an SPR. Relation $T_{\text{most}+}$ removes at least one pair of brackets, but it can also remove more, or even all brackets. Therefore, the expressed relation is transitive, since for all $v, x, y \in I^*$, $(v, y) \in T_{\text{most}+}$ if $(v, x) \in T_{\text{most}+}$ and $(x, y) \in T_{\text{most}+}$. It is irreflexive and antisymmetric, since for all $(v, w) \in T_{\text{most}+}$, $|v| > |w|$. Thus, the relation of $T_{\text{most}+}$ is a SPR.

The union of two strict preference relations is not generally a strict preference relation since the result is not necessarily antisymmetric. Still, some preference relations can be combined under union.

All the SPRs defined in (21–29) are regular and easily implementable with finite-state transducers or bimorphisms. Typically the corresponding transducer contains only a few states.

5.2 Applications of Strict Preference Relations

The Method of Yli-Jyrä and Koskenniemi Yli-Jyrä and Koskenniemi [2] were inspired by the "matching" approach [17] used in selecting candidates that have minimal compatible set of constraint violations in Finite-State Optimality Theory. A somewhat similar approach has been used in [18]. In order to implement the method for parallel obligatory replacement [2], the minimality

constraint is inverted to obtain strings with maximal bracketing. The five steps of the resulting method in [2] are the following:

1. Prepare C_i (and compute X_i);
2. Compute ${C_i}' = {d_B}^{-1}(C_i)$ and ${X_i}' = \texttt{<}_1 X_i \texttt{>}_1$;
3. Compute $\textsc{Gen}_G = [1 :: (\Pi \backslash \Sigma)\texttt{=>}\emptyset] \uplus \uplus_{i=1}{}^{n}[1 :: {X_i}'\texttt{=>}{C_i}']$;
4. Compute $D = \pi_1(\textsc{Gen}_G)$ and $D' = \{w_2 | w_1 \in D \wedge (w_1, w_2) \in T_{\text{most}+}\}$;
5. Compute $\textsc{Gen}'_G = \{w_1{:}w_2 \in \textsc{Gen}_G | w_1 \notin D'\}$ and return $\beta_2(\textsc{Gen}'_G)$.

(30)

Generalized Lenient Composition Jäger [3] defines a left-associative binary operator (`glc`) and controversially calls it *generalized lenient composition operator* although it rather addresses a problem with lenient composition than generalizes it. We add two variants: *inverse* one (denoted by `r-glc`) and *bidirectional* one (denoted by `b-glc`). The operators assume two operands: a candidate set $S \subseteq \Pi^*$ and a strict preference relation $T \subseteq I^* \times I^*$, and they are defined as follows:

$$S \texttt{ glc } T \stackrel{\text{def}}{=} \{w \in S \mid \neg\exists w' (w' \in S \wedge (\pi_2(w), \pi_2(w')) \in T)\}; \tag{31}$$

$$S \texttt{ r-glc } T \stackrel{\text{def}}{=} \{w \in S \mid \neg\exists w' (w' \in S \wedge (\pi_1(w), \pi_1(w')) \in T)\}; \tag{32}$$

$$S \texttt{ b-glc } T \stackrel{\text{def}}{=} (S \texttt{ glc } T) \cap (S \texttt{ r-glc } T). \tag{33}$$

Now, as we have slightly elaborated our formal machinery, we can express the compilation method of [2] as a two-step algorithm:

1. Compute $\textsc{Gen}_G = \textsc{Gen}_{\text{core}} \uplus \uplus_{i=1}{}^{n}[1 :: \texttt{<}_1 X_i \texttt{>}_1 \texttt{(=>)} C_i]$;
2. Compute $\beta_2(\textsc{Gen}_G \texttt{ r-glc } T_{\text{most}+})$.

(34)

5.3 The Alternative Modes of Obligatoriness

Together with Kaplan and Kay [14], Yli-Jyrä and Koskenniemi [2] maintain that all other replacement modes are *subsets* of the relation described by the corresponding optional replacement (denoted by **Opt**), which contrasts to the approach of [6].

It is not trivial to describe how obligatory replacement restricts **Opt**. Three different approaches have already been presented in this paper (Sections 4.1, 4.2 and 5.2). These do not produce the results in general, and it is therefore at least fair to say a word about their differences. The replace relations corresponding to these modes form an inclusion order **M1** $\subseteq$ **M2** $\subseteq$ **M3** $\subseteq$ **Opt**. The modes themselves can be described as follows:

M1 – Overlapping Synchronized Coercion Kaplan and Kay (page 357 of [14]) mention but do not elaborate a possibility of overlapping applications of

obligatory rules. However, Section 4.1 and [2] point out that a GTWOL rule can have overlapping applications. Due to this, a double arrow rule such as [1 :: A:a B:p ∪ B:b C:c <=> _] is in a self-conflict, which results into an over-constrained input-output mapping that fails to relate any output to input string ABC.

M2 – Non-Overlapping Coercion The bracketed double arrow of BGTWOL (Section 4.2) differs from the double arrow of GTWOL by using a bracketing that serves to avoid overlapping applications in every candidate mapping. Its left-arrow part is, however, surprisingly constrained since, for example, rule [2 :: a:xb:0 (<=) #$\Pi^*\pi_2{}^{-1}$(ab)_ $\pi_1{}^{-1}$(a)Π^*#] does not allow such mapping as (abab<$_1$ab>$_1$a): (abab<$_1$x0>$_1$a).

As far as I can judge, Karttunen *et al.* [6, 1, 13] seem to implement this notion of obligatoriness into XFST when compiling the right-oriented rule ab (->) x // ab _ b.[2] In particular, the definition 27 (the Replace component relation) in [6] cannot skip a center in a proper context without replacing it. Candidate (ab<u>ab</u><$_1$ab>$_1$a):(ab<u>ab</u><$_1$x0>$_1$a) is not included to the result, because there is a non-overlapping substring (underlined) that should have been replaced with x0. The method ignores the fact that this additional change cannot be done (it would result into incorrect symbol-pair string *(ab<$_1$ab>$_1$<$_1$ab>$_1$a):(ab<$_1$x0>$_1$<$_1$x0>$_1$a)) without altering the lower left context that was assumed by one of the changes.

M3 – Maximal Set of Non-Overlapping Changes This third notion of obligatoriness is represented by Section 5.2 and [2]. The subtle difference between the new method [2] and [6] was not recognized in [2] although it is a crucial part of backward compatibility. The semantics of the new method is illustrated considering the mappings of optional replace rule (18). Mappings (19b and 19c) are maximal candidates under the preference relation $T_{\text{most}+}$.

5.4 The GLC Approach to M2

Besides the bracketed double arrow, the new method of [2] can be modified to capture mode **M2**. The solution is based on the idea of a *bracketed identity rule* where brackets B_2 are used to mark the valid replacement locations that are held back *i.e.* the applications of the rule [1 :: <$_2\pi_1(X_i)$>$_2$ (=>) C_i]. The contribution of this is to make the candidate set more dense under $T_{\text{most}+}$. SPR $T_{\text{lest},B2}$ prefers candidates that do not contain B_2. Pivot GEN_G has always at least one candidate without brackets B_2.

$$[X_1\text{->}C_1, , \ldots, X_i\text{->}C_n] \stackrel{\text{def}}{=} \beta_2(\text{GEN}_G\ \texttt{r-glc}\ (T_{\text{most}+} \cup T_{\text{lest},B_2})) \qquad (35)$$
$$\text{where } \text{GEN}_G = \text{GEN}_{\text{core}} \Cap \Cap_{i=1}{}^{n}[1 :: <_1 X_i >_1 \cup <_2 \pi_1(X_i) >_2 (\texttt{=>}) C_i].$$

[2] It may be helpful to remark that [19] and [14] compile *directed* rewriting rules, see the discussion in [6]. Therefore they do not belong here.

Example The set $\textsc{Gen}_G$ contains the following candidates for the unbracketed input `abababa`:

$$\left\{\begin{array}{llll} \texttt{abababa} & \texttt{ab<}_2\texttt{ab>}_2\texttt{aba} & \texttt{abab<}_2\texttt{ab>}_2\texttt{a} & \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_2\texttt{ab>}_2\texttt{a} \\ \texttt{abababa}, & \texttt{ab<}_2\texttt{ab>}_2\texttt{aba}, & \texttt{abab<}_2\texttt{ab>}_2\texttt{a}, & \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_2\texttt{ab>}_2\texttt{a}, \\ & \texttt{ab<}_1\texttt{ab>}_1\texttt{aba} & \texttt{abab<}_1\texttt{ab>}_1\texttt{a} & \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_1\texttt{ab>}_1\texttt{a} \\ & \texttt{ab<}_1\texttt{x0>}_1\texttt{aba}, & \texttt{abab<}_1\texttt{x0>}_1\texttt{a}, & \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_1\texttt{x0>}_1\texttt{a} \end{array}\right\}. \quad (36)$$

The set of all *maximal* bracketings in $\textsc{Gen}_G$ is obtained using strict preference relation $T_{\text{most}+}$ that ignores the bracket labels when comparing bracketed strings:

$$\begin{array}{c} \textsc{Gen}_G\,\texttt{r-glc}\,T_{\text{most}+} \\ \cap \\ d_B{}^{-1}(\pi_1{}^{-1}(\texttt{abababa})) \end{array} = \left\{\begin{array}{l} \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_2\texttt{ab>}_2\texttt{a}\ \ \texttt{ab<}_1\texttt{ab>}_1\texttt{aba}\ \ \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_1\texttt{ab>}_1\texttt{a} \\ \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_2\texttt{ab>}_2\texttt{a},\ \texttt{ab<}_1\texttt{x0>}_1\texttt{aba},\ \texttt{ab<}_2\texttt{ab>}_2\texttt{<}_1\texttt{x0>}_1\texttt{a} \end{array}\right\}. \quad (37)$$

The set of candidates without identity rule applications is obtained with an additional preference transducer T_{lest,B_2}:

$$\begin{array}{c} \textsc{Gen}_G\,\texttt{r-glc}\,T_{\text{lest},B_2} \\ \cap \\ d_B{}^{-1}(\pi_1{}^{-1}(\texttt{abababa})) \end{array} = \left\{\begin{array}{ll} \texttt{ab<}_1\texttt{ab>}_1\texttt{aba} & \texttt{abab<}_1\texttt{ab>}_1\texttt{a} \\ \texttt{ab<}_1\texttt{x0>}_1\texttt{aba}, & \texttt{abab<}_1\texttt{x0>}_1\texttt{a} \end{array}\right\}. \quad (38)$$

There is only one candidate that remains in the intersection of these sets.

$$\begin{array}{c} \textsc{Gen}_G\ \texttt{r-glc}\ (T_{\text{most}+} \cup T_{\text{lest},B_2}) \\ \cap \\ d_B{}^{-1}(\pi_1{}^{-1}(\texttt{abababa})) \end{array} = \left\{\begin{array}{l} \texttt{ab<}_1\texttt{ab>}_1\texttt{aba} \\ \texttt{ab<}_1\texttt{x0>}_1\texttt{aba} \end{array}\right\}. \quad (39)$$

Insertion Replaces It does not make sense to apply the obligatoriness constraint to insertion rules or, more generally, to rules where $\epsilon \in \pi_1(X_i)$. Because there could always be more insertions, no candidate would qualify as maximal according to T_{most}. Using SPR $T_{\text{most}+}$ instead has avoided this problem.

Sometimes it is desirable to make insertions only once at any position matching the context conditions. *E.g.* we may not want to limit insertions by consuming the material that might be rewritten by other parallel rules. Kempe and Karttunen [1] address the problem by providing a special strategy for one-time insertion. A similar strategy can be captured easily with SPR T_{norep}. After this constraint has been applied, it is natural to apply SPR T_{most} in order to get candidates with maximal sets of non-repeated insertions and other replacements.

$$\texttt{[.}\ X_i\ \texttt{.]}\ \texttt{->}\ C_i \stackrel{\text{def}}{=} \beta_2(\textsc{Gen}_G\ \texttt{r-glc}\ T_{\text{norep}}\ \texttt{r-glc}\ (T_{\text{most}} \cup T_{\text{lest},B_2})). \quad (40)$$

Inverse and Bidirectional Replacement Inverse and bidirectional replacement [1] are extremely easy to implement. All what is needed is to use an adequate generalized lenient composition operator *i.e.* `glc` or `b-glc`.

Directed Replace It is possible to implement various directed replace operators [7] (and similar methods of [14, 19]) using suitable strict preference relations.

$$X_i \texttt{@->} C_i \stackrel{\text{def}}{=} \beta_2(\textsc{Gen}_G \texttt{ r-glc } (T_{\text{lr}} \cup T_{\text{lrlong}})) \tag{41}$$

$$X_i \texttt{->@} C_i \stackrel{\text{def}}{=} \beta_2(\textsc{Gen}_G \texttt{ r-glc } (T_{\text{rl}} \cup T_{\text{rllong}})) \tag{42}$$

$$X_i \texttt{@>} C_i \stackrel{\text{def}}{=} \beta_2(\textsc{Gen}_G \texttt{ r-glc } (T_{\text{lr}} \cup {T_{\text{lrlong}}}^{-1})) \tag{43}$$

$$X_i \texttt{>@} C_i \stackrel{\text{def}}{=} \beta_2(\textsc{Gen}_G \texttt{ r-glc } (T_{\text{rl}} \cup {T_{\text{rllong}}}^{-1})). \tag{44}$$

Accordingly, we observe that using generalized restriction with BGTWOL gives an elegant and uniform approach for computing a large family of different replace (and marking) rules.

6 The Bipartite Approach

A New Design Pattern The method of [2] has a bipartite design that contains two main components: $\textsc{Gen}_G$ and Con.

$$\beta_2(\textsc{Gen}_G \circ \textsc{Con}) = \beta_2(\textsc{Gen}_G \circ {}_1T_1 \circ {}_2T_2 \cdots \circ {}_mT_m). \tag{45}$$

The components are responsible for different kinds of tasks. $\textsc{Gen}_G$ is the *candidate generator*, and Con is the lenient constraint component. The latter consists of *lenient constraints* $T_1, T_2, \ldots, T_n$ and left-associative *generalized lenient composition operators* $\circ_i, \circ_2, \ldots, \circ_m \in \{\texttt{glc}, \texttt{r-glc}, \texttt{b-glc}\}$. Jäger [3] observes that lenient composition [20] can be expressed with generalized lenient composition.

The bipartite approach is very useful because it lends itself to many applications such as compilation of ranked rewriting rules and directed replacement rules. By encapsulating the context conditions of the replacements into $\textsc{Gen}_G$, the conditions are always observed in the generated candidates regardless of any strategic preferences. It is the task of Con to choose among alternative candidates, but it does not have to know about the internal structure of the candidate generator. By using strict preference relations, we obtain a uniform representation for various rule modalities.

Ranked Rules In Optimality Theory [21], the constraints are ranked. Similar ranking is possible also among parallel replacement rules. Various kinds of ranked rules have numberless applications beyond phonology and morphology.

Skut *et al.* [8] present a compiler for ranked left-to-right fixed-length replacement rules with upward-oriented contexts. A similar system of rules can be implemented easily in the current approach. First, we construct the bracketed GTWOL grammar corresponding to optional rules in such a way that brackets $<_i$ and $>_i$ occur in rules of rank i. The highest rank is now 1, and lowest n. The resulting bracketed relation, $\textsc{Gen}_G$, is constrained as follows:

$$\textsc{Gen}_G \texttt{ r-glc } T_{\text{lr},B_1} \texttt{ r-glc } T_{\text{lr},B_1 \cup B_2} \texttt{ r-glc } T_{\text{lr},B_1 \cup B_2 \cup B_3} \ldots \texttt{ r-glc } T_{\text{lr},B}. \tag{46}$$

In order to give preference to longest applications, strict preference relation $T_{\mathrm{lr},B'} \cup T_{\mathrm{lrlong},B'}$ could be used instead of $T_{\mathrm{lr},B'}$.

In [8], each rule rewrites a fixed-length substring. Our solution is more general since (i) the contexts in rules can be oriented and combined under Boolean operators, (ii) centers are not restricted to fixed-length substrings, and (iii) each rank can be shared by several parallel rules.

7 Conclusion

In the paper, we reviewed and extended the previously published 2-page description of the Yli-Jyrä and Koskenniemi method [2] for compiling parallel replace rules into transducers. Its relationship to the method of Kempe and Karttunen [1] is elaborated and discussed critically.

The background sections of this paper included an updated description of *Generalized Two-Level Grammars* (GTWOL). The semantics of GTWOL was defined, for the first time, using an *extended notion of generalized restriction*. In comparison to [14, 19, 1], the solution reduces considerably the number of different brackets needed to compile parallel replacement rules.

The main result in this work is to elaborate the *bipartite design pattern* that was employed implicitly in [2].

- Candidates are generated with a GTWOL grammar.
- Three forms of Jäger's composition operator [3] (GLC) were employed.
- Strict preference relations account for obligatoriness, directionality and length-based preferences.

The design makes it easy to capture a *variety of rule application modes* without bothering about conditions of individual rules. Parallel replace rules can have even heterogeneous modes and the rules can be ranked.

In addition, the paper presented *three important notions of obligatoriness* and defined new compilation methods for each of them. The notion corresponding to the method of Kempe and Karttunen [1] was covered by two alternative solutions.

Acknowledgements

I am grateful for Måns Hulden for a better example rule (18) that is simpler than my original example. In addition, he suggested in September 2007 that the mode **M2** could be captured without generalized lenient composition. This inspired me to add the bracketed double arrow operator. Further comments by Dale Gerdemann helped me improve the presentation.

References

1. Kempe, A., Karttunen, L.: Parallel replacement in finite state calculus. In: 16th COLING 1996, Proc. Conf. Volume 2., Copenhagen, Denmark (1996) 622–627

2. Yli-Jyrä, A., Koskenniemi, K.: A new method for compiling parallel replacement rules. In Holub, J., Žďárek, J., eds.: Implementation and Application of Automata, 12th International Conference, CIAA 2007, Revised Selected Papers. Volume 4783 of LNCS., Springer (2007) 320–321
3. Jäger, G.: Gradient constraints in Finite State OT: The unidirectional and the bidirectional case. In: Proceedings of FSMNLP 2001, an ESSLLI Workshop, Helsinki (August 20–24 2001) (35–40)
4. Yli-Jyrä, A., Koskenniemi, K.: Compiling generalized two-level rules and grammars. In: Proceedings of FinTAL 2006. LNAI (2006)
5. Yli-Jyrä, A.: Applications of diamonded double negation. In: Proceedings of FSMNLP 2007, Potsdam, Germany (forthcoming 2008)
6. Karttunen, L.: The replace operator. In: 33th ACL 1995, Proceedings of the Conference, Cambridge, MA, USA (1995) 16–23
7. Karttunen, L.: Directed replace operator. In Roche, E., Schabes, Y., eds.: Finite-state language processing, Cambridge, Massachusetts, A Bradford Book. The MIT Press (1996) 117–147
8. Skut, W., Ulrich, S., Hammervold, K.: A flexible rule compiler for speech synthesis. In: Proceedings of Intelligent Information Systems 2004, Zakopane, Poland (2004)
9. 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)
10. Karttunen, L., Beesley, K.R.: Two-level rule compiler. An additional documentation file on the CD-ROM supplement of Beesley and Karttunen (2003) (March 5 2003)
11. Karttunen, L.: Finite-state constraints. In: Proceedings of the International Conference on Current Issues in Computational Linguistics, Universiti Sains Malaysia, Penang, Malaysia (June 10-14 1991)
12. Arnold, A., Dauchet, M.: Morphismes et bimorphismes d'arbres. Theoretical Computer Science **20** (1982) 33–93
13. Beesley, K.R., Karttunen, L.: Finite state morphology. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, CA, USA (2003)
14. Kaplan, R.M., Kay, M.: Regular models of phonological rule systems. Computational Linguistics **20**(3) (September 1994) 331–378
15. Gerdemann, D., van Noord, G.: Transducers from rewrite rules with backreferences. In: 9th EACL 1999, Proceedings of the Conference. (1999) 126–133
16. Chomsky, N., Halle, M.: The Sound Pattern of English. Harper and Row, New York (1968)
17. Gerdemann, D., van Noord, G.: Approximation and exactness in Finite-State Optimality Theory. In Eisner, J., Karttunen, L., Thériault, A., eds.: SIGPHON 2000, Finite State Phonology. (2000)
18. Eisner, J.: Directional constraint evaluation in optimality theory. In: 20th COLING 2000, Proceedings of the Conference, Saarbrücken, Germany (2000) 257–263
19. Mohri, M., Sproat, R.: An efficient compiler for weighted rewrite rules. In: 34th ACL 1996, Proceedings of the Conference, Santa Cruz, CA, USA (June 24-27 1996) 231–238
20. Karttunen, L.: The proper treatment of optimality in computational phonology. In: Finite State Methods in Natural Language Processing. (1998) 1–12
21. 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)

Finite-State Rule Deduction for Parsing Non-Constituent Coordination

Sina Zarrieß and Wolfgang Seeker

Potsdam University, Germany

Abstract. In this paper, we present a finite-state approach to constituency and therewith an analysis of coordination phenomena involving so-called non-constituents. We show that non-constituents can be seen as parts of fully-fledged constituents and therefore be coordinated in the same way. We have implemented an algorithm based on finite state automata that generates an LFG grammar assigning valid analyses to non-constituent coordination structures in the German language.

1 Introduction

In standard syntactic theories, coordination is usually seen as a structure that conjuncts syntactic entities which are both of the same phrasal category and maximal projections (see [1] for an example). However, in various languages one finds examples for coordinated structures where conjoins don't fit into any standard concept of syntactic constituent, or don't even share their phrasal properties. The main insight the following examples should give is that, in general, syntactic entities can be coordinated if the material they share completes them in a syntactically well-formed way.

(1) Asterix darf und Obelix darf nicht vom Zaubertrank trinken.
Asterix may and Obelix may not of magic potion drink

(2) Asterix gibt Obelix ein Wildschwein und Idefix einen Knochen.
Asterix gives Obelix a boar and Idefix a bone

(3) Obelix verschlingt ein kleines und nascht von einem großen
Obelix devours a small and snacks of a big
Wildschwein.
boar

Even on the level of constituent structure, it is quite difficult to assign a valid analysis to this kind of constructions, see figure 1.

In [2], the author gives an exhaustive overview of strategies that have been implemented for covering these phenomena. In his paper, he mainly shows that despite their nearly identical behaviour, constituent and non-constituent coordination are often treated as completely seperated structures. Sometimes even the underlying parsing algorithm is modified for being able to parse this particular

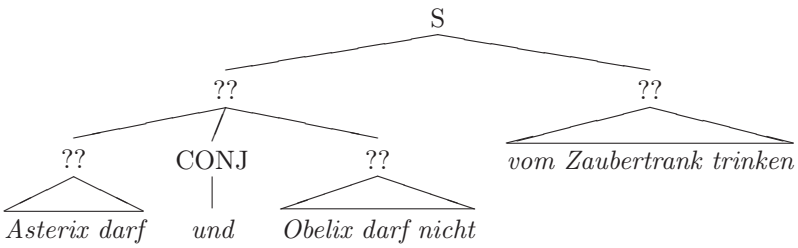


Fig. 1. Which category has a non-constituent?

structure. What seems to make it so difficult to analyze non-constituent phenomena is that they undermine the fundamental concept of syntactic category.

We have implemented an algorithm that generates a grammar allowing partial constituents in the context of coordination from a standard context-free LFG grammar in the well-known XLE format. This automatically generated grammar then covers the main types of non-constituent coordination as *Right Node Raising* and *Conjunction Reduction*. Our approach has been inspired by [3] who make use of the fact that the right rule sides of an LFG grammar are regular languages and can therefore be represented by finite state automata. But still, their strategy operates on the level of the parsing algorithm and suffers from the drawback of being barely formalized.

In the remaining of this paper, we will describe the way we apply the theory of automata to the problem of partial rule generation. Our basic goal is to assign the same category to partial constituents that expect identical completing material. When appropriate, we will go into the details of the XLE grammar implementation that realizes the assignment of a well-formed syntactic analysis to non-constituent coordination structures.

2 Preliminaries

2.1 Formal Language Devices

Initially, since we want to conceptualize constituency in a finite-state frame, we give some notations for formal devices that specify regular as well as context-free languages. For instance, regular expressions are instances of regular languages over an alphabet Σ. They are usually defined recursively as follows:

Definition 1. *The empty set $\emptyset$, the empty word ϵ and all symbols $a \in \Sigma$ denote regular expressions. If r and s are regular expressions, their disjunction $(r + s)$, concatenation (rs) and closure r^* also denote regular expressions.*

An alternative device for manipulating regular languages are the so-called finite-state automata.

Definition 2. *A finite-state automaton (or FSA) A is a 5-tuple $A = (\Sigma, Q, I, F, E)$ where:*

1. *Σ is the finite input alphabet of the automaton;*
2. *Q is the finite set of states;*
3. *$I \subseteq Q$ is the set of initial states;*
4. *$F \subseteq Q$ is the set of final states;*
5. *$E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ the set of transitions.*

Given a transition $e \in E$, we denote by $s[e]$ its input label, by $p[e]$ its origin (previous) and by $n[e]$ its destination (next) state. A path $\pi = e_1 \dots e_n$ in the automaton A is defined as an element of E^* where forall $n[e_{i-1}] = p[e_i]$. A *successful* path in A is a path $\pi = e_1 \dots e_n$ where $p[e_1] \in I$ and $n[e_n] \in F$, thus a path from an initial to a final state. The concept of an origin and destination state can be extended to paths so that $p[\pi]$ and $n[\pi]$ are meant to be the origin and destination state of the path π. Accordingly, $s[\pi]$ denotes the concatenation of the input symbols of a path π.

We define the right language of a state $q \in Q$ as follows:
$\overrightarrow{L}(q) = \{w \mid \pi(q, w, q'), q' \in F\}$. The language accepted by an automaton A is the union of the right languages of all initial states $L(A) = \bigcup_{q \in I} \overrightarrow{L}(q)$.

The definition of a finite-state automaton can be easily extended to a finite-state transducer (FST) $T = (\Sigma, \Delta, Q, I, F, E)$ where Δ is the finite output alphabet and $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q$. The composition of two finite state transducers $T = T_1 \circ T_2$ is defined as $T = (\Sigma_1, \Delta_2, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, E)$ where $(< q_1, q_1{}' >, a, b, < q_2, q_2{}' >) \in E$ if $(q_1, a, c, q_2) \in E_1$ and $(q_1{}', c, b, q_2{}') \in E_2$.

Whereas the language $L(A)$ accepted by an FSA A is a set $L \subseteq \Sigma^*$, the language recognized by an FST can be seen as a relation $L(T) \subseteq (\Sigma \times \Delta)^*$. We will sometimes use the common set theoretic operations like union and product to denote operations on FSTs. Furthermore, we define an operation of *projection*, where $P_1(T) \subseteq \Sigma^*$ corresponds to the input and $P_2(T) \subseteq \Delta^*$ to the output language of T.

The LFG grammar that will provide input for our rule generator is an instance of a context-free language specified as a context-free grammar.

Definition 3. *By the classical definition, CFGs are tuples $G_c = \langle V_T, V_N, S, R \rangle$ including the requirements:*

1. *V_T is the finite alphabet of terminals;*
2. *V_N is the finite alphabet of nonterminals;*
3. *S is the start symbol;*
4. *R is the set of rules $r : \sigma \rightarrow \omega$, so that $\sigma \in V_N$ and $\omega \in (V_T \cup V_N)^*$.*

In the following section, we will mainly show how these computationally quite distinct devices can be interrelated to give rise to a formal concept of (partial) constituency.

2.2 LFG Grammars in XLE

The actual implementation of our coordination grammar, as described in the following section, has been done within the XLE development environment for

LFG grammars (see [4] or [5] for a general introduction). To enable the reader to follow the presented examples, we will give a short description of some common XLE notations.

In general, XLE grammars have to be specified as *left-canonical* (see definition 4). An example for a typical rule in XLE is shown in figure 2. Disjunction is expressed by curly brackets, optionality by round brackets. The category symbol stands to the left of a colon, its f-annotation to the right. An expression $(\ SUBJ) =!$ in the f-annotation means that whatever is derived by that symbol is in the $SUBJ$ feature of the current f-structure. The $-symbol is the $\in$-symbol and is used for sets.

```
NP -->
    { (D) A*: !$(^ADJUNCT); N:(^SUBJ)=!;
    | PRON:(^SUBJ)=!;
    }
    (PP:!$(^ADJUNCT);).
```

Fig. 2. A simple NP rule

To formulate category independent structures in a general way one can define so-called XLE *macros*. In figure 3, our macro for coordination is shown. A category symbol placed in **_cat** is copied to all occurences in the rule body and associated with the specified f-annotation. Note that the $-operator causes the coordinated elements to be unified in a set and their f-structures to be treated as a single set.

```
COORD(_cat) =
    _cat: !$^;
    ( COMMA _cat: !$^; )*
    {CONJ[konj]: (^NUM)=pl | CONJ[disj]: (^NUM)=sg }
    _cat: !$^.
```

Fig. 3. The coordination macro

Another useful XLE mechanism called *complex category symbols* allows enriching category symbols with different parameters to generalize information about different instances of the same constituent. A parameterized symbol then stands for all possible instanciations of its parameters. An example is shown in figure 4.

However, one should not think of category parameters in terms of Prolog-style variables. Parameters are only interpreted when they occur at the left side of a rule. Thus, if parametrized categories appear on the right side of a rule,

```
NP[_param $ { pl sg }] -->
    {
        D N
    |
        N: _param = pl;
    }.
```

Fig. 4. Parameterized NP rule

their mother category is parametrized by at least the sum of all its daughter parameters. In consequence, underspecified parameters aren't licensed and we are forced to enumerate all possible instanciations of a parameter at least at the root symbol.

3 A Finite-State Concept of Constituency

3.1 A First Approach to Partial Constituents

One can easily see that the right side ω of a standard context-free rule as defined in definition 3 is equivalent to a regular expression by the finite concatenation of symbols $x_i \in (V_T \cup V_N)$, where $x_1 \ldots x_n = \omega$. Very often, context-free grammars (LFG grammars for instance) especially make use of the regular syntax in that they allow operations like disjunction or closure to appear on the right rule side. This fact enables us to define the concept of a *left-canonical* grammar.

Definition 4. *A context-free grammar $G_c = \langle V_T, V_N, S, R\rangle$ is called left-canonical if and only if for every $\sigma \in V_N$ there exists at most one $r \in R$ such that $r : \sigma \rightarrow \omega$.*

For every context-free grammar G that doesn't satisfy this property, it is possible to define an equivalent *left-canonical* grammar G'. Given a grammar G_1 that contains one pair of rules r_1, r_2 such that $\sigma_1 = \sigma_2$, you simply have to unify the right sides ω_1, ω_2 such that $r_{1+2} : \sigma_{1+2} \rightarrow \omega_1 + \omega_2$.

This type of grammar is a useful construction when you want to define the regular language $L_1(\sigma)$ that can be derived from a nonterminal $\sigma \in G$ by a single rule application. If G is left-canonical, for every $\omega_i \in R$, ω_i denotes exactly the regular expression that corresponds to the language $L_1(\sigma_i)$. Thus, it is possible to represent the set of rules R of a context-free grammar by a set of automata $A_R = \{A_{\omega_i} \mid L(A_{\omega_i}) = L_1(\sigma_i)\}$.

Given these connections between regular and context-free languages, one can formalize the syntactic concept of constituency in the following way:

Definition 5. *A linguistic entity ω_i is called a constituent with respect to a context-free grammar G, if there exists a path $\pi \in A_\omega$ such that A_ω is equivalent to some right rule side $\sigma \rightarrow \omega \in G$ and $s[\pi] = \omega_i$.*

In case $p[\pi] \notin I$ or $n[\pi] \notin F$, thus if π isn't *successful*, the related constituent $s[\pi]$ is called *partial* and labelled by $\omega^{i,j}$ where $i = p[\pi]$ and $j = n[\pi]$. Otherwise, if π is a *successful* path in A_ω we call the constituent $\omega^{0,n}$ that corresponds to $s[\pi]$ *complete*. We define an isomorphism $\mu : L_1(\sigma^{i,j}) \rightarrow \Pi_{e_i \ldots e_j}$ that maps the regular language representing a (possibly partial) constituent to its corresponding set of paths in the rule automaton.

Now, we are able to state the hypothesis in [3] concerning coordination of non-constituents in a formal way:

Lemma 1. *Two constituents* $\omega^{i,j}, \omega^{u,v}$ *with* $\omega^{i,j} = s[\pi_{i,j}]$ *and* $\omega^{u,v} = s[\pi_{u,v}]$, $\pi_{i,j}, \pi_{u,v} \in A_\omega$ *can be coordinated iff* $i = u$ *and* $j = v$.

3.2 A Precise Formalization of Coordination

The intuition underlying this generalized coordination rule is that (partial) constituents can be conjoined if the constituents that make them complete are identical. However, this lemma doesn't completely capture this intuition. Consider the automaton in figure 5 that represents a simplified NP-rule.

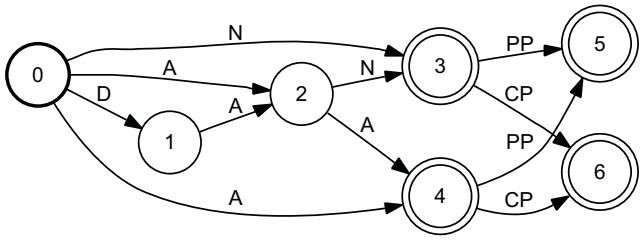


Fig. 5. An automaton representing the right side of a simple NP rule

Although the constituents labelled by the input sequences N and A can be completed by identical constituents PP or CP, their corresponding paths don't have a single destination state in common. Therefore, we have to refine our concept of a valid rule automaton A_ω.

It can be shown that for every regular set there exists a canonical minimum state automaton. The well-known Myhill-Nerode theorem (see [6]) states that a regular language is the union of equivalence classes on a right invariant equivalence relation R_L of finite index. R_L is defined by: xR_Ly if and only if for all $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$. In terms of an automaton this means that its set of states can be partitioned into equivalence classes with respect to a right invariant relation $R_L{}'$ where $q_i R_L{}' q_j$ if $\overrightarrow{L}(q_i) = \overrightarrow{L}(q_j)$.

The concept of a state representing an equivalence class with respect to its right language is exactly what we need for the partial constituents being

assigned a meaningful category. In consequence of the Myhill-Nerode theorem, one can now say that all partial constituents $\omega^{i,j}$ that can be completed by an identical constituent $\omega^{j,k}$ lead the canonical rule automaton into an unique state k, since they have the same right language $\overrightarrow{L}(k)$. This leads us to a more precise generalized coordination rule:

Lemma 2. *Two constituents* $\omega^{i,j}, \omega^{u,v}$ *with* $\omega^{i,j} = s[\pi_{i,j}]$ *and* $\omega^{u,v} = s[\pi_{u,v}]$, $\pi_{i,j}, \pi_{u,v} \in A_\omega$ *can be coordinated iff* $i, u \in ||i||$ *and* $j, v \in ||j||$.

Hence, our goal is to generate all possible partial right rule sides $\omega^{i,j} = s[\pi_{i,j}]$ from the canonical rule automaton and assign them a category which is parametrized by the pair of states (i, j) in the canonical rule automaton. Formally, this can be expressed by a mapping that relates a *left-canonical* context-free grammar G to a grammar G' that is equivalent with respect to the maximal projections but explicits its partial constituents:

Definition 6. *Given a canonical CFG* $G_c = \langle V_T, V_N, S, R\rangle$ *and its equivalent canonical rule automaton* A_ω *we define a mapping onto a grammar* $G_p = \langle V_T\text{`}, V_N\text{`}, S\text{`}, R\text{`}\rangle$ *that fulfils the following conditions:*

1. $S = S\text{`}$ *and* $V_T = V_T\text{`}$.
2. $V_N\text{`} = V_N \cup \bigcup_{\sigma \in V_N, i,j \in Q_{A_\omega}} \sigma^{i,j}$.
3. *Every rule* $r : \sigma \rightarrow \omega$, $r \in R$ *is mapped by a function* f *to a set of rules* $f(r) = \{\sigma^{i,j} \rightarrow \omega^{i,j} \mid s[\pi_{i,j}] = \omega^{i,j}, \pi_{i,j} \in {E^*}_{A_\omega}, \pi_{i,j} \in \mu(L(\sigma^{i,j}))\}$.
4. $R\text{`} = \bigcup_{r \in R} f(r) \cup R$.

However, in our XLE implementation we avoid blowing up the set of rules as is described above. The mechanism of *complex category symbols* allows us to treat all partial constituents $\omega^{i,j}$ of a mother category σ as derivations of the same rule that parametrize σ in a different way. Figure 6 shows an example of a parametrized NP-rule, where the origin and destination state of the constituent are realized as the parameters **_from** and **_to**. To indicate the possibly partial status of this category it is called *XPsub*. Every *XPsub* is parameterized by all parameters $\phi_1 ... \phi_n$ of its original category, by a parameter **_koord**, which marks the coordination status of the constituent, and two parameters **_from** and **_to** to mark the start and end index of a particular substring (we use the values **sa** and **se** for start and end state). Thus, *XPsub* is equal to the set of rules yielded by the mapping $f(XP)$.

Because of the constituent status of the non-constituents we can now coordinate them with the same macro we use for constituent coordination, see figure 3. The restriction, that only constituents with congruent origin and destination state can be coordinated, is inherent to the macro since only categories with identical parameters form the same category symbol.

3.3 Assembling Complete Constituents

Up to now, we have completely ignored the fact that we don't want our grammar to derive partial constituents in contexts other than coordination. Finally, we

```
NPsub[_ntype $ {std rel int},_koord,_from,_to] -->
{
AP: _ntype = std; e: _from=s19 _to=s21 _koord=no;
|
...
|
@(COORD_PART NPsub[_ntype $ {std rel int},no,_from,_to])
e: _koord=yes;
|
@(COORD NPsub[_ntype $ {std rel int},_koord,_from,_to]
e: _from=sa _to=se)
}.
```

Fig. 6. *NPsub* generated from an NP rule

will have to add rules to the grammar that reassemble partial constituents to complete ones. The formalization is straightforward:

Lemma 3. *A sequence of constituents* $\omega^{i_0,j_0}, \omega^{i_1,j_1}, \ldots, \omega^{i_n,j_n}$, $\omega^{i_i,j_i} \in A_\omega$ *forms a complete constituent, iff for every* $j_n = i_{n+1}$ *and* i_0 *is the initial state and* i_n *some final state in* A_ω.

Thus, we would have to add to our coordination grammar G_c' all possible instances of the rule $r : \sigma^{i,k} \rightarrow \omega^{i_i,j_i}, \ldots, \omega^{i_k,j_k}$. But this would lead to an extreme ambiguity in the resulting grammar where every complete constituent could, in addition to its original derivations, be derived by numerous ways of putting together its partial right rule sides $\omega^{i,j}$. To cope with this serious overgeneration, we restrict the completion to require at least one partial constituent that has been formed by a conjunction of partial constituents. Every partial constituent is further parametrized with a boolean feature that marks its coordination status to check for this condition.

Lemma 4. *A sequence* ${\omega_{coord}}^{i_0,j_0}, {\omega_{coord}}^{i_1,j_1}, \ldots, {\omega_{coord}}^{i_n,j_n}$, $\omega^{i_i,j_i} \in A_\omega$ *forms a complete constituent, iff there exists some* ${\omega_{coord}}^{i,j}$ *with* $coord = true$ *and for every* $j_n = i_{n+1}$ *and* i_0 *is the initial state and* i_n *some final state in* A_ω.

In our implementation we have restricted the respective completion rules to binary branching.

We are now able to give an elegant analysis for e.g. the phenomenon of Right Node Raising as shown in figure 8.

4 Automated Deduction of Partial Constituents

We could now extract all partial constituents from the rule automaton by performing a standard breadth-first search on its transitions. However, we prefer this operation to be defined on the algebra of finite state automata.

```
NPkompl[_ntype $ {std rel int},_from,_inter,_to] -->
{
NPsub[_ntype $ {std rel int},yes,_from,_inter]
NPsub[_ntype $ {std rel int},no,_inter,_to]
|
NPsub[_ntype $ {std rel int},no,_from,_inter]
NPsub[_ntype $ {std rel int},yes,_inter,_to]
|
NPmiss[_ntype $ {std rel int},yes,_inter]
PPsub[std,no,_inter,se]: !$(^ADJUNCT); e:_from=sa _to=se;
}.
```

Fig. 7. *NPkompl* generated from an NP rule

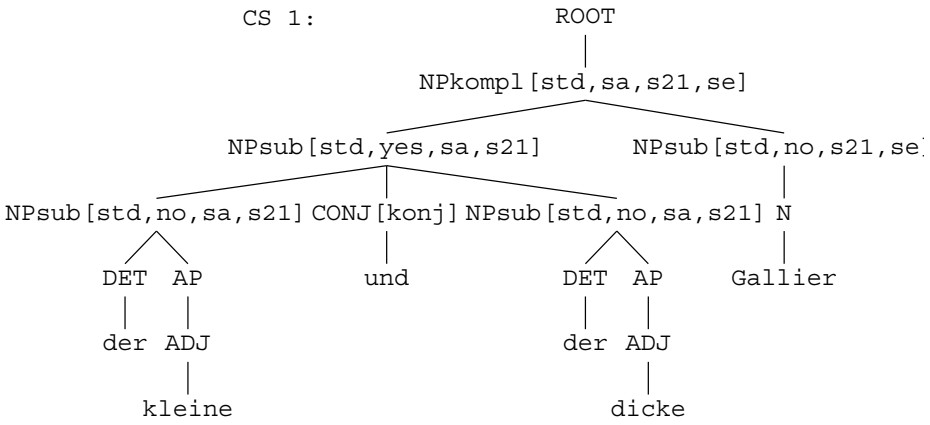


Fig. 8. Analysis for the German NP *the small and the big gaul*

In a first step, we define a transducer T_{sub} that can be notated as a regular expression $T_{sub} = (ID(\Sigma)^* \cdot (\Sigma \times \{\epsilon\})^* \cdot ID(\Sigma)^*)$. By definition of the composition of FSTs, the transducer $ID(A_\omega) \circ T_{sub}$ yields a relation that maps arbitrary long prefixes and suffixes in A_ω to ϵ, so that the second projection of this relation $P_2(ID(A_\omega) \circ T_{sub})$ yields an automaton $A_{\omega^{i,j}}$ that accepts all partial and complete constituents of A_ω. The problem with this first approach is that the states in $A_{\omega^{i,j}}$ of course don't correspond to those in A_ω anymore. However, we have shown in the previous section that the state information inherent to the canonical rule automaton A_ω is exactly what we need to parametrize partial categories.

Thus, before generating partial constituents, we have to explicitely encode the states in the FSA A_ω. Broadly speaking, we do this by indexing the regular expression that constitutes the right rules sides ω to label every distinct position of the expression. Then, the automaton ${A_\omega}^{ind}$ equivalent to the indexed right rule side contains position labels such that for every given state $n[e] \in A$ with the right language $\overrightarrow{L} \setminus I$ there exists a unique input label $i[e] \in I_\omega$. By a series of compositions with deletion transducers we obtain an automaton ${A_\omega}^{i,j}$ whose input sequences $s[\pi]$, where π is *successful*, are strings of the form $s = i_i \cdot \omega^{i,j} \cdot i_j$.

First, we have to define the operation of indexing positions in regular expressions.

Definition 7. *Let r, s and t denote regular expressions and $I = \{1, 2, 3 \ldots n\}$ an index alphabet such that n is the number of subexpressions of r, t or s respectively. If $r = a, a \in \Sigma$, $r^{ind} = i_0 \cdot a \cdot i_n$. If $r = (s+t)$, $r^{ind} = (s^{ind} + t^{ind}) \cdot i_n$. If $r = st$, $r^{ind} = (s^{ind} t^{ind}) \cdot i_n$. If $r = s^*$, $r^{ind} = (s^{ind})^* \cdot i_n$.*

The automaton ${A_\omega}^{ind} = (\Sigma \cup I_\omega, Q, I, F, E)$ corresponding to an indexed right rule side ω^{ind} accepts input sequences $s[\pi_{0,n}] = i_0 \cdot {\omega^{0,n}}_{ind} \cdot i_n$. To remove redundant position labels in ${A_\omega}^{ind}$, it is composed with the transducer T_{delpos}:

$$T_{delpos} = (ID(i_0) \cdot (I \times \{\epsilon\})^* \cdot ID(\Sigma)) \cdot ((I \times \{\epsilon\})^* \cdot (ID(I) \cdot ID(\Sigma))^*) \quad (1)$$

The second projection ${A_\omega}^{delpos} = P_2({A_\omega}^{ind} \circ T_{delpos})$ yields input sequences of the form $s[\pi_{0,n}] = i_0 x_0 i_1 x_1 \ldots x_n i_n$ where $i_i \in I, x_i \in \Sigma$ so that $x_0 \ldots x_n = \omega$.

Lemma 5. *If there are two constituents ${\omega^{i,j}}_{ind} = s[\pi_{i,j}], {\omega^{u,v}}_{ind} = s[\pi_{u,v}]$, with $\pi_{i,j}, \pi_{u,v} \in {A_\omega}^{delpos}$, there will be a pair of sequences $s[\pi_{i-1,j+1}] = i_i \cdot x_i \ldots x_j \cdot i_j, s[\pi_{u-1,v+1}] = i_i \cdot x_u \ldots x_v \cdot i_j$ if and only if $i = u, j = v$ and $i - 1 = u - 1, j + 1 = v + 1$.*

This lemma shows that 1) if there are some indexed constituents (some paths whose input sequences start and end with a symbol $a \in \Sigma$) that have an identical origin and destination state, there are some paths in the same automaton that accept these constituents labeled with an identical start and end position and 2) if there are some input sequences that have an identical start and end label, their corresponding constituents have an identical origin and destination state.

Now, the deletion transducer has to be changed to preserve the right labels for each partial constituent. Given the following transductions:

$$T_{sub} = (ID(\Sigma) \cup (\Sigma \times \{\epsilon\}) \cup ID(I))^*$$
$$T_{normpos} = ((I \times \{\epsilon\})^* \cdot (ID(I) \cdot ID(\Sigma))^+ \cdot ID(I) \cdot (I \times \{\epsilon\})^*)$$
$$T_{uni} = (ID(I) \cdot (ID(\Sigma) \cup (I \times \{\epsilon\}))^+ \cdot ID(I))$$

and the composition:

$$T_{\omega^{i,j}} = A_\omega \circ T_{delpos} \circ T_{sub} \circ T_{normpos} \circ T_{uni}$$

The second projection $P_2(T_{\omega^{i,j}})$ finally produces an automaton ${A_\omega}^{i,j}$ where each input sequence $s[\pi_{i-1,j+1}] = i_i \cdot \omega^{i,j} \cdot i_j$.

To conclude, we have shown that the mapping defined in 6 can be performed on the algebra of FSTs.

5 Extensions

Consider the following coordination:

(4) Asterix spielt mit und Obelix schimpft auf Idefix.
Asterix plays with and Obelix rails against Idefix

Currently, our grammar doesn't cover this type of constructions because there aren't some partial constituents being coordinated, but two complete phrases that contain a partial constituent of the same type. To be able to cover these phenomena, we define a third constituent type called *missing*.

Definition 8. *A constituent $\omega^{i,j}$ has the missing-property if it fulfils the following conditions:*

1. *i and j correspond to some initial and final state in A_ω.*
2. *For the rightmost symbol $\sigma_r = \omega^{j-1,j}$ there is a rule in G such that $r : \sigma_r \rightarrow \omega$.*
3. *The rightmost symbol σ_r is a partial constituent ${\sigma_r}^{i,j}$ where i corresponds to the inital state in $A_{\omega r}$ and $j \notin F_{A_{\omega r}}$*

In figure 9, we give an example for a $NPmiss$-category. The $XPmiss$ symbol also contains all parameters $\phi_1 ... \phi_n$ of its original category and a parameter **_koord** to mark its coordination status. In figure 10 we then present an analysis for a category with the missing property.

Of course, there are still some other specific constructions that aren't covered by the presented model. For instance, one could imagine the coordination of two partial constituents that don't have the same *right language*, but whose intersection of right languages isn't empty, and the material that completes them is contained in this intersection. Or else, coordinated constituents may have an identical right language, but not in the same rule automaton. In this case, their origin and destination state won't help identifying them. Thus, an alternative parametrization for partial constituents would be their right and left language in contrast to their origin and destination state.

```
NPmiss[_ntype $ {std rel int},_koord,_missing] -->
{
DET: ^=!; AP: _ntype = std; N: _ntype = std;
PPsub[std,no,sa,_missing]: ! $ (^ADJUNCT); e: _koord=no;
|
...
|
@(COORD_PART NPmiss[_ntype $ {std rel int},no,_missing])
e: _koord=yes;
}.
```

Fig. 9. *NPmiss* generated from an NP rule

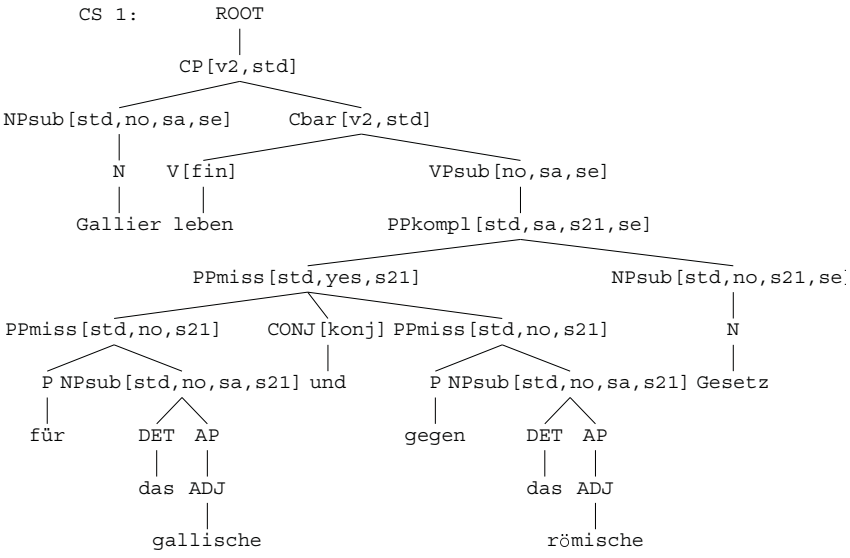


Fig. 10. Analysis for the German sentence *Gauls live for the gaulic and against the roman law*

6 Conclusion

These considerations outlined some formal properties as well as the limitations of the coordination approach proposed by [3]. We have shown that the problem of non-constituent coordination can be solved by expliciting partial constituents on the surface of the grammar. An adequate way of describing these partial categories is to see them as paths in a canonical right rule side automaton. This formalization gives rise to an efficient mechanism of subrule generation. We have also extended the approach of [3] to coordination of complete constituents with embedded partial constituents at the right periphery.

References

1. Kaplan, R.M., Maxwell, J.T.: Constituent coordination in lexical-functional grammar. In: Proc. of the 12th COLING, Budapest, Hungary (1988) 303–305
2. Milward, D.: Non-constituent coordination: Theory and practice. In: Proceedings of the 15th International Conference on Computational Linguistics (COLING94), Kyoto. (1994)
3. Maxwell, J., Manning, C.: A theory of non-constituent coordination based on finite-state rules (1996)
4. Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The parallel grammar project. In: Proc. of COLING-2002 Workshop on Grammar Engineering and Evaluation. (2002) 1–7
5. Kaplan, R.M., Maxwell, J.T.: Lfg grammar writer's cookbook. Technical report, Xerox PARC (1996)
6. Hopcroft, J., Ullmann, J.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)