An efficient double complementation algorithm for superposition-based finite-state morphology

Yli-Jyrä, Anssi Mikael

2009

Yli-Jyrä , A M 2009 , ' An efficient double complementation algorithm for superposition-based finite-state morphology ' , NEALT Proceedings Series , vol. 4 , pp. 206-213 . < http://hdl.handle.net/10062/9765 >

http://hdl.handle.net/10138/24223

publishedVersion

An Efficient Double Complementation Algorithm for Superposition-Based Finite-State Morphology

Anssi Yli-Jyrä
Department of General Linguistics
HFST Research Group
University of Helsinki
Finland
anssi.yli-jyra helsinki.fi

Abstract

This paper presents an *efficient compilation algorithm* that is several orders of magnitude faster than a standard method for context restriction rules. The new algorithm combines even hundreds of thousands of rules in parallel when the alphabet is large but the resulting automaton is sparse. The method opens new possibilities for representation of context-dependent lexical entries and the related processes. This is demonstrated by encoding complete HunSpell dictionaries as a single context restriction rule whose center placeholder in contexts is replaced with a new operation, called *underline operation*. The approach gives rise to new *superposition-based* context-dependent lexicon formalisms and new methods for on-demand compilation and composition of two-level morphology.

1 Introduction

The use of the context restriction rule of two-level morphology (Koskenniemi, 1983) has traditionally been limited to relatively simple phonological rules. The purpose of this paper is to make this operation more widely applicable in finite-state morphology, by improving its compilation methods and by developing new ways to encode morphological processes with this operation.

1.1 Prior Art

Compilation of compound context restrictions is a problem that has inspired a series related solutions:

- Kaplan and Kay's method (Karttunen et al., 1987; Kaplan and Kay, 1994) requires that the occurrences of the center in each string are bracketed. Then it applies double complementation (if-then idioms) to restrict the context of each bracket. The method needs separate brackets for each context.
- A related approach (Grimley-Evans et al., 1996; Kiraz, 1997; Kiraz, 2000) handles similar single-tuple rules directly with double complementation, being a special case a star-free compilation method of Yli-Jyrä (2003).
- Yli-Jyrä (2003) uses star-free regular operations to show that the context restriction rule with overlapping multi-tuple centers is definable in first-order predicate logic.
- Yli-Jyrä and Koskenniemi (2004; 2006) and Hulden (2009) use first-order quantification over substrings in order to express the same semantics more abstractly and efficiently.
- Additional bracketing can express disjoint but not necessarily contiguous centers (Yli-Jyrä, 2008a; Yli-Jyrä, 2008b). Kiraz (2000) uses it with contiguous centers.

The prior art suggests that complementation is an essential part of context restriction whereas bracketing has a supplementary role.

1.2 The Problem

Complementation can be really difficult to implement efficiently due to its transition complexity. This is illustrated in Fig. 1.

Automaton of Fig. 1(a) is a deterministic acceptor for the language $(a|b)^*\diamond^* a(a|c)^*$, and automaton of Fig. 1(b) accepts its complement

Kristiina Jokinen and Eckhard Bick (Eds.)
NODALIDA 2009 Conference Proceedings, pp. 206–213

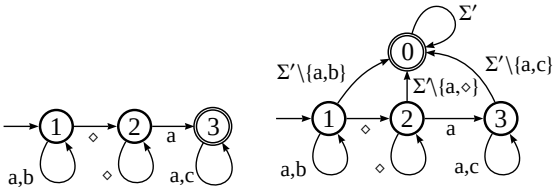

Figure 1: (a) A DFA and (b) its complement.

with respect to the universal language Σ'^* where $a, b, c, \diamond \in \Sigma'$. By complementation, only one state is added, but the number of transitions grows significantly. With an alphabet of *e.g.* 500 symbols and a deterministic automaton with 2 million states, we can easily end up with an automaton with 1 billion (10^9) transitions. Such a blow-up is not only a problem for complementation itself but also for further processing.

Finding an efficient compilation method for context restrictions is particularly important because it would pave the way for a similar efficient solution for an even more general-purpose operation, *generalized restriction (GR)* (Yli-Jyrä and Koskenniemi, 2004; Yli-Jyrä, 2008a). GR admits context restriction as one of its special cases. Other interesting uses of the GR operation include:

- conventional, partition-based and generalized two-level grammars (Silfverberg and Lindén, 2009; Barthélemy, 2007b; Yli-Jyrä and Koskenniemi, 2006; Yli-Jyrä, 2008a)
- replace rules (Yli-Jyrä, 2008b) and tree linearization (Barthélemy, 2007a).

The combinatorial properties of the GR operation are parallel to a first-order predicate logic (Hulden, 2009). Thanks to these combinatorial properties, the double-arrow rules of two-level morphology (Karttunen, 1991) and some other rules can be reduced into context-restriction rules. This underlines the importance of the efficient compilation method for context restriction rules.

1.3 The Contributions of This Paper

The first significant contribution of this paper is an algorithm that compiles context restriction rules much more efficiently than the prior approaches to the underlying double complementation. The new algorithm has been inserted to a branch of SFST code base (Schmid, 2005) and it is in the process of migrating from there to the HFST API and the HFST family of tools (Koskenniemi and Yli-Jyrä, 2009; Lindén et al., 2009). The algorithm can be used to compile also other two-level operations such as prohibition and coercion rules. Furthermore, the algorithm can be embedded into a new operation, *superposing composition*. This operation compiles and composes the lexicon and the two-level grammar in parallel.

The second significant contribution of the current paper is to initially demonstrate that large context restrictions can be used to address a wide range of context-dependent morphological processes. In particular, we will show that it can be used for

- synthesis of cyclic lexicons
- context-dependent concatenation and truncation for prefixing and suffixing
- context-dependent circumfixing (with aid of a postprocessing step).

Moreover, we have reasons to believe that this list could be extended with a number of other morphological and phonological processes. In particular, we propose *underlined expressions and languages* as a convenient means for representing context-dependent processes through coincidences and superposition. The current algorithm and other efficient implementations for such underlined languages give rise to *superposition-based* lexicon formalisms that are more general than concatenation-based LEXC (Karttunen, 1993) and truncation/concatenation-based HunSpell (Németh et al., 2004).

1.4 The Structure of the Paper

The paper is structured as follows: The definitions and notations are given in Section 2. Section 3 motivates the aimed semantics, simplifies its representation and generalizes it to capture a variety of rules. Section 4 presents the new algorithm that implements this semantics procedurally. Section 5 presents applications of the new algorithm to morphological processes. Section 6 evaluates the efficiency using tiny examples and huge HunSpell dictionaries, and then discusses further optimizations and generalizations. The paper is concluded by Section 7.

2 Preliminaries

Denote the empty language with $\emptyset$ and the empty string with ϵ. If x is string, the set $\{x\}$ is denoted

alternatively with x. Let A and B be regular languages and let k be a positive integer. Concatenation AB, intersection $A \cap B$, union $A \cup B$, complement $\overline{A}$, asymmetric difference $A \backslash B$, Kleene's closure A^* and bounded iteration $A^{\leq k}$ are defined as usual.

A deterministic finite automaton (DFA) is a tuple (A, Q, i, F, δ) where A is the *finite input alphabet*, Q is the *finite set of states*, $i \in Q$ is the *initial state*, $F \subseteq Q$ is the *set of final states*, $\delta : Q \times A \to Q$ is the *transition relation*. Extended transition relation $\hat{\delta} : Q \times A^* \to Q$ is defined in such a way that $\hat{\delta}(q, \epsilon) = q$ and $\hat{\delta}(q, aw) = \hat{\delta}(\hat{\delta}(q, a), w)$ for all $q \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$. The automaton *accepts* a string $w \in \Sigma^*$ if and only if $\hat{\delta}(i, w) \in F$. The language recognized by the automaton is the set $\{w | \hat{\delta}(i, w) \in F\}$.

Two-level systems (Koskenniemi, 1983; Kiraz, 2000) and their rules describe binary or n-ary regular same-length relations between strings. However, these systems and their rules can be viewed also as descriptions of languages over a tuple alphabet. In this paper, two-level rules describe languages over a set of tuples, Σ. For related conventions and the definitions of Id, Range, Domain, and composition $\circ$, see (Kaplan and Kay, 1994).

2.1 Context Restriction

A *(compound) context restriction (CR)* rule (Koskenniemi, 1983; Kaplan and Kay, 1994) is conventionally written in the form

$$X \Rightarrow \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \#. \tag{1}$$

where the *center* X and the left and right parts $L_i, ..., L_n, R_1, ..., R_n$ of *contexts* are regular languages over a known alphabet Σ. It is reasonable to assume that $\epsilon \notin X$. For strings $v, y \in \Sigma^*$, denote condition $v \in L_i \wedge y \in R_i$ by expression $v \_ y \in L_i \_ R_i$. The semantics of the CR rule is a set of strings given by

$$\{w | w \in \Sigma^* \wedge (\forall v \in \Sigma^*)(\forall x \in X)(\forall y \in \Sigma^*) \\ w \neq vxy \vee (\exists L_i \_ R_i) v \_ y \in L_i \_ R_i\}.$$

2.2 Generalized Restriction

Generalized restriction (GR) (Yli-Jyrä, 2008a) is an operation whose operands consist of the universal language Σ^*, a set of markers M, a language $W \subseteq \Sigma^*(M\Sigma^*)^{\leq k}$ and a language $W' \subseteq (\Sigma \cup M)^*$. Set M is such that $M \cap \Sigma = \emptyset$ and it contains, conventionally, symbols $\diamond, \diamond_1, \diamond_2, ...$ that are called *diamonds*. Languages W and W' are called *generalized precondition* and *generalized postcondition*, respectively. The relationship between the syntax and the semantics of GR is defined by equation

$$[W \overset{\Sigma,k,M}{\Longrightarrow} W'] = [W \overset{M}{\Longrightarrow} W'] = \Sigma^* \backslash h_M(W \backslash W')$$

where $h_M : (\Sigma \cup M)^* \to \Sigma$ is a morphism that deletes the markers from strings.

3 The Aimed Semantics

The GR operation (Yli-Jyrä and Koskenniemi, 2006) yields the semantics of a CR rule by

$$[X \Rightarrow \#L_1 \_ R_1 \#, \ldots, \#L_n \_ R_n \#]$$
$$= [\Sigma^* \diamond X \diamond \Sigma^* \overset{M}{\Rightarrow} \cup_{i=1}^{n} L_i \diamond X \diamond R_i] \tag{2}$$
$$= \Sigma^* \backslash h_M((\Sigma^* \diamond X \diamond \Sigma^*) \backslash (\cup_{i=1}^{n} L_i \diamond X \diamond R_i)). \tag{3}$$

While formula (2) looks elegant, it actually employs, in (3), complementation of rather complex languages. In the following, we will simplify the representation of the centers and contexts and arrive at formula (13) that captures the meaning of a CR rule set. After this, we will add to this rule set two special CR rules that account for a restricted universe and prohibition rules.

3.1 Simplifications

3.1.1 Decomposition of Generalized Centers

CR rule centers are, in general, subsets of Σ^+ rather than subsets of Σ. In the GR semantics, CR rules can be reduced to a GR of more limited kind with a decomposition technique.[1] The technique expresses that every symbol in the restricted strings must have separately a valid context. By decomposition, (2) gives

$$[\Sigma^* f(X) \Sigma^* \overset{M}{\Rightarrow} \cup_{i=1}^{n} L_i \; g(X) \; R_i]. \tag{4}$$

employs two functions $f, g : \Sigma^+ \to \Sigma^* \diamond^* \Sigma^+$, that are given by

$$f(x) = h_M^{-1}(x) \cap \Sigma^* \diamond \Sigma^+ \tag{5}$$
$$g(x) = h_M^{-1}(x) \cap \Sigma^* \diamond^* \Sigma^+. \tag{6}$$

3.1.2 Combining Sets of Rules

In a set of context restriction rules, separate rules may induce right arrow conflicts. The right-arrow conflicts of two (or more) context restrictions can

[1] For related techniques, see (Yli-Jyrä and Koskenniemi, 2006; Yli-Jyrä, 2008a).

be resolved using the *coherent intersection operation* ⩒ (Yli-Jyrä, 2008a) of generalized restrictions:

$$[W \overset{M}{\Longrightarrow} W'] \Cap [U \overset{M}{\Rightarrow} U'] \quad (7)$$

$$= [(W \cup U) \overset{M}{\Rightarrow} ((W \cap W') \cup (U \cap U'))]. \quad (8)$$

The same operation can be used to combine context restrictions that are not in conflict. Thus, coherent intersection operation combines arbitrary many context restrictions and resolves also any right-arrow conflicts. This gives equation

$$\Cap_{r=1}^{m} [X_r \ \texttt{=>} \ \#L_{r,1}\_R_{r,1}\#, \ldots, \#L_{r,n_r}\_R_{r,n_r}\#]$$

$$= [\cup_{r=1}^{m} \Sigma^* f(X_r) \Sigma^* \overset{M}{\Rightarrow} \cup_{r=1}^{m} \cup_{i=1}^{n_r} L_{r,i}\, g(X_r)\, R_{r,i}]. \quad (9)$$

3.1.3 Constrained Center Alphabet

Let $S \subseteq \Sigma$ be the alphabet of centers of rules in such a way that $\cup_{r=1}^{m} X_r \subseteq S^*$. In fact, traditional two-level grammars admit $S = \cup_{r=1}^{m} X_r$. Even when $\cup_{r=1}^{m} X_r \not\subset \Sigma$, it is reasonable to assume that $S \subseteq \cup_{r=1}^{m} X_r$. From this, it follows that $\Sigma^* \diamond S \Sigma^* = \cup_{r=1}^{m} \Sigma^* f(X_r) \Sigma^*$. This equation (9) to

$$[\Sigma^* \diamond S \Sigma^* \overset{M}{\Rightarrow} \cup_{r=1}^{m} \cup_{i=1}^{n_r} L_{r,i}\, g(X_r)\, R_{r,i}]$$

$$= [\Sigma^* g(S) \Sigma^* \overset{M}{\Rightarrow} \cup_{r=1}^{m} \cup_{i=1}^{n_r} L_{r,i}\, g(X_r)\, R_{r,i}]. \quad (10)$$

3.1.4 Rearranging the Contexts

The right hand side of (10) can be viewed in such a way that the contexts are arranged according to the symbol that follows the marker:

$$[\Sigma^* g(S) \Sigma^* \overset{M}{\Rightarrow} \cup_{a \in S} \cup_{i=1}^{n_a} L_{a,i}\, g(a)\, R_{a,i}\ ] \quad (11)$$

3.1.5 The Underline Operator

For notational convenience, introduce an *underline* operator $\underline{X} = g(X)$. With this operator, (11) can be rewritten as

$$[\Sigma^* \underline{S} \Sigma^* \overset{M}{\Rightarrow} \bigcup_{a \in S} \bigcup_{i=1}^{n_a} L_{a,i}\ \underline{a}\ R_{a,i}\ ]. \quad (12)$$

3.1.6 Underlined Expressions and Languages

Regular expressions with the underline operator admit Boolean combinations between the right-hand sides. *E.g.* $\text{K}\underline{\text{i}}\text{TaB} \cup \text{KiT}\underline{\text{a}}\text{B} = \text{K}\underline{\text{i}}\text{T}\underline{\text{a}}\text{B} \cap \underline{\text{KiTaB}}$. The underline operator gives a compact representation for gapped centers. *E.g.*

$$[\Sigma^* \underline{\{i, a\}} \Sigma^* \overset{M}{\Rightarrow} (\ \text{K}\underline{\text{i}}\text{TaB} \cup \text{KiT}\underline{\text{a}}\text{B}\ )]$$

$$= [\Sigma^* \underline{\{i, a\}} \Sigma^* \overset{M}{\Rightarrow} \ \text{K}\underline{\text{i}}\text{T}\underline{\text{a}}\text{B}\ ].$$

Due to the closure properties of languages with underline, we view (12) more generally as:

$$[\Sigma^* \underline{S} \Sigma^* \overset{M}{\Rightarrow} C\ ] \quad (13)$$

where $C \subseteq (\Sigma \cup \{\diamond\})^*$ and $h(C) \subseteq C$.

3.2 Additional Context Restrictions

3.2.1 Constraining the Universe

The prior two-level systems present two approaches to the treatment of non-center symbols $\Sigma \backslash S$:

- the alphabet Σ gathered from the centers (Kiraz, 2000); thus $\Sigma \backslash S = \emptyset$.
- the strings $(\Sigma \backslash S)^*$ are not restricted by the rules (Koskenniemi, 1983).

There is an approach that can emulate both the prior approaches: in our approach, all strings are restricted by non-underlined contexts $h(C) \subseteq C$ with an additional CR rule:

$$[\Sigma^* \overset{M}{\Rightarrow} h(C)]$$

$$= [\Sigma^* \overset{M}{\Rightarrow} C]. \quad (14)$$

By combining formulas (13) and (14),[2] we obtain:

$$[\Sigma^* \underline{S} \Sigma^* \cup \Sigma^* \overset{M}{\Rightarrow} C\ ]. \quad (15)$$

3.3 Center Prohibition Rules

The rule operator of a center prohibition rule is `/<=` (Karttunen, 1991). The semantics a set of intersected prohibition rules is given by

$$\cap_{r=1}^{p} [X_r \ \texttt{/<=} \ \#L_{r,1}\_R_{r,1}\#, \ldots, \#L_{r,n_r}\_R_{r,n_r}\#]$$

$$= [\cup_{r=1}^{p} i \cup_{i=1}^{n_r} (L_{r,i}\ X_r\ R_{r,i}) \overset{M}{\Rightarrow} \emptyset] \quad (16)$$

Let $P = \cup_{r=1}^{p} i \cup_{i=1}^{n_r} (L_{r,i}\ X_r\ R_{r,i})$. Because the universe is already restricted to $h(C)$, we can assume that $P \subseteq h(C) \subseteq C$. Combining (16) with (15) gives

$$[\Sigma^* \underline{S} \Sigma^* \overset{M}{\Rightarrow} C] \cap [\Sigma^* \overset{M}{\Rightarrow} C] \cap [P \overset{M}{\Rightarrow} \emptyset]$$

$$= [\Sigma^* \underline{S} \Sigma^* \overset{M}{\Rightarrow} C] \Cap [\Sigma^* \overset{M}{\Rightarrow} C] \Cap [P \diamond_2 \overset{M}{\Rightarrow} \emptyset]$$

$$= [(\Sigma^* \underline{S} \Sigma^* \cup \Sigma^* \cup P \diamond_2) \overset{M}{\Rightarrow} C]$$

$$= \Sigma^* \backslash h(\ (\Sigma^* \diamond S \Sigma^* \cup \Sigma^* \cup P \diamond_2) \backslash C). \quad (17)$$

Observe that coherent intersection admits intersection when both the preconditions and the postconditions are disjoint, or the postconditions are equivalent.

[2] Formula (14) reminds us from a prior formula where no markers are in use (Kiraz, 2000, 87).

4 The Algorithm

The algorithm to compute (17) is given in Fig. (2).

```
SUPERPOSE((Σ ⊎ {◇, ◇2}, Q, i, F, δ) ∈ DFA, S ⊆ Σ)
 1: assert{ δ : Q × (Σ ⊎ {◇, ◇2}) → Q}                          %1
 2: I′ ← {(i, 1)}∪{(q, 2)|(i, ◇, q)∈δ}; Q′ ← {I′}               %1%3
 3: M ← ∅; B ← ∅                                               %1
 4: for q ∈ {q|(q, ◇2, q′) ∈ δ} do                             %2
 5: |  if {a|(q, a, q) ∈ δ} = Σ then                           %2
 6: |  |  B ← B ∪ {q}                                         %2
 7: while ∃P′(P′ ∈ Q′\M) do                                    %1
 8: |  M ← M ∪ {P′}                                           %1
 9: |  P ← {p | (p, r)∈P′}                                    %1
10: |  c ← |{q | {(q, 3)∈P′}|                                 %4
11: |  if P′∩(F × {1}) ≠ ∅ then                               %1
12: |  |  if δ∩(P×{◇2}×Q) = ∅ then                           %2
13: |  |  |  if {q|(q, 3)∈P′}\F = ∅ then                     %4
14: |  |  |  |  F′ ← F′ ∪ {P′}                               %1
15: |  for all a ∈ Σ do                                       %1
16: |  |  C[a] ← 0;N[a] ← ∅;L[a] ← (a ∉ S)                   %1%3
17: |  for all (q, a, q′) ∈ δ ∩ (P×Σ×Q) do                   %1
18: |  |  for all (q, 1) ∈ P′ do                             %1
19: |  |  |  N[a] ← N[a] ∪ {(q′, 1)}                         %1
20: |  |  if a ∈ S then                                      %3
21: |  |  |  for all (q, 2) ∈ P′ do                          %3
22: |  |  |  |  N[a] ← N[a] ∪ {(q′, 3)}                      %3
23: |  |  |  |  L[a] ← true                                  %3
24: |  |  for all (q, 3) ∈ P′ do                             %4
25: |  |  |  N[a] ← N[a] ∪ {(q′, 3)}                         %4
26: |  |  |  C[a] ← C[a] + 1                                 %4
27: |  for all b∈Σ s.t. N[b]≠∅ do                            %1
28: |  |  for all (q, 1) ∈ N[b] ∧ (q, ◇, q′) ∈ δ do          %3
29: |  |  |  N[b] ← N[b] ∪ {(q′, 2)}                         %3
30: |  |  if N[b] ∩ (B × {1, 2, 3}) = ∅ then                %2
31: |  |  |  if L[b] then                                    %3
32: |  |  |  |  if C[b] = c then                            %4
33: |  |  |  |  |  Q′ ← Q′ ∪ {N[b]}                         %1
34: |  |  |  |  |  δ′ ← δ′ ∪ {(P′, a, N[b])}                %1
35: return (Σ, Q′, I′, F′, δ′)                                %1
```

Figure 2: An algorithm that computes $\Sigma^*\backslash h(\ (\Sigma^*\diamond S\,\Sigma^* \cup \Sigma^* \cup P\diamond_2)\backslash C)$ from $C \cup P\diamond_2$.

Theorem 1 *When G is a DFA that recognizes the language $C\cup P\diamond_2$, algorithm* SUPERPOSE(G, S) *in Fig. 2 returns a DFA that recognizes the language $\Sigma^*\backslash h((\Sigma^*\diamond S\,\Sigma^* \cup \Sigma^* \cup P\diamond_2)\backslash C)$.*

Proof. Basically, the algorithm performs a subset construction over the state space $Q \times \{1, 2, 3\}$, using a transition function $\delta_2 : (Q \times \{1, 2, 3\}) \times (\Sigma \cup \{\diamond, \diamond_2\}) \times (Q \times \{1, 2, 3\})$ defined in such a way that $\delta_2([q, 1], \diamond) = [\delta(q, a), 2]$, $\delta_2([q, 2], a) = [\delta(q, a), 3]$ and $\delta_2([q, r], a) = [\delta(q, a), r]$ for all $a \in \Sigma$ and $r \in \{1, 3\}$. The lines marked with comment %1 copy a subautomaton $Q\times\{1\}$ whose transitions are labeled with ordinary symbols Σ. This aspect of the algorithm computes $\Sigma^*\backslash h(\Sigma^*\backslash C) = \Sigma^* \cap C$. The lines marked with %2 subtract from this subautomaton another deterministic subautomaton whose final states are those that have a leaving transition on $\diamond_2$. Line 13 alone performs the subtraction, but the other lines with comment %2 implement an optimization that prevents insertion of states that correspond to language $\Sigma^* \cup \Sigma^*\diamond_2$. This aspect of the algorithm computes $\Sigma^*\backslash h(P\diamond_2)$. Lines marked with %3 compute correct prefixes $\Sigma^*\backslash h((\Sigma^*\diamond S)\backslash(L_{a,i}\diamond a))$ by testing that every symbol $a{\in}S$ is preceded by a diamond. Finally, the lines marked with %4 ensure that whenever the prefix $v\diamond$ of a context string $v\diamond ay{\in}C$ has provided a necessary and unique (since G is deterministic) diamond for prefix va, the path accepting $v\diamond ay \in C$ will continue (lines 10, 26, and 32) and finally reach a final state (line 13) when the $\diamond$-free string $vay \in h(C)$ ends. Together with %3-lines, this aspect ensures that if the resulting automaton accepts a string $x \in \Sigma^*$, then $x \in \Sigma^*\backslash h((\Sigma^*\diamond S\,\Sigma^*)\backslash C)$. The four aspects constrain one another. In sum, the algorithm computes the language $\Sigma^*\backslash h((\Sigma^*\diamond S\,\Sigma^* \cup \Sigma^* \cup P\diamond_2)\backslash C)$. □

5 Applications to Morphology

The presented algorithm has several applications.

Two-Level Grammars The algorithm can be used to compile traditional two-level grammars (Koskenniemi, 1983) where left-arrow conflicts have already been resolved. For this purpose, (1) the center alphabet S is collected from the CR rules, (2) the centers of CR rules are moved to the contexts with the underline operator, (3a) the union of resulting languages and the universal language Σ^* is assigned to C, (4) all surface (and lexical) coercion rules are converted into prohibition rules, (5) the union of the prohibition rules becomes P, (6) the union $C \cup P\diamond_2$ is converted to a minimal deterministic automaton G. The compiled grammar automaton T_G is returned by SUPERPOSE(G, S).

It is well known that the size of the compiled two-level grammar may be prohibitively large in practice. The size blow-up can be avoided by restricting the compiled grammar T_G with the lexicon transducer T_L (Karttunen, 1994) in composition $T_L \circ T_G$. Following this general approach, we could define *superposing composition* where the current algorithm is embedded into a composition algorithm and compiles both the lexicon and the grammar at the same time.

Continuation Classes The superpose algorithm can be used to compile continuation classes that are an essential part of the widely used LEXC formalism (Karttunen, 1993). Consider the following entry in a LEXC sublexicon:

```
LEXICON Verbs
talk V ;                    (18)
```

In order to compile the lexicon, (1) put the sublexicon names into the center alphabet: S = {<Root>, <Verbs>, <V>, ..., <#>}, (2) compute the alphabet Σ as a union of S and the normal symbols occurring in the lexical entries, (3) convert each entry into a regular expressions with underline, *e.g.* <#>Σ*<u><Verbs></u>talk<V>Σ*, (4) compute the union of these expressions, (5) add to the union the expression (Σ*<u><#></u>) ∪ (<u><#></u><Root>Σ*), (6) this union, C, is converted to a minimal deterministic automaton G. The compiled lexicon automaton T_L is returned by SUPERPOSE(G, S).

This approach extends to HunSpell dictionary (.dic) format. For example, the entry

```
glossy/TSP                  (19)
```

corresponds to underlined regular expression <#>Σ*<u><Root></u>glossy{<T>, <S>, <P>, <#>}Σ*. In the terminology of LEXC, we could say that the .dic-file contains only the Root sublexicon, and its each entry has an implicit continuation to the word boundary class #.

Context-Dependent Affixation and Truncation The prefix and suffix rules of HunSpell work largely in a symmetrical way. A HunSpell (Németh et al., 2004) suffix rule,

```
SFX T y iest [^aeiou]y,     (20)
```

specifies continuation class T, affix iest, truncation y, condition [^aeiou]y and the implicit continuation class #. The combination of (19) and (20) is encoded as string <#><Root>gloss<D>y<T><-D>iest<#> where <D> and <-D> are additional center symbols that bracket the truncated part. The affix rule corresponds to an underlined expression: <#>Σ*[^aeiou]<u><D></u>y<u><T><-D></u>iest<#>Σ*.[3] To make the pieces to work together, the entry for glossy is extended with optional <D>-brackets.

[3]This reminds of alternation rules that are triggered by lexical features (Kiraz, 1997)

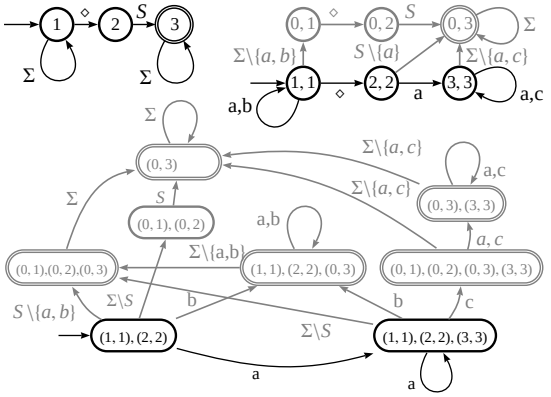

Figure 3: Minimal DFAs for languages $W' = \{a,b\}^*\underline{a}\{a,c\}^*$ and $\overline{W'} = \Sigma\backslash(\{a,b\}^*\underline{a}\{a,c\}^*)$ are displayed in Fig. 1. Let $S = \{a,b,c\}$. The three DFAs in this figure (in clockwise order) recognize languages (a) $W = \Sigma^*\diamond S\Sigma^*$, (b) $W\backslash W' = W \cap \overline{W'}$, (c) $h(W\backslash W')$. The shadowed states and transitions do not contribute to the final result that is $a^* = \Sigma^*\backslash h(W\backslash W')$.

Circumfixing The prefix-suffix pair un+iable can be viewed, however, as a circumfix because the prefix cannot be attached alone to some stems. Our encoding for word unidentifiable is <#>un<U-><Root>identif<D>y<U><D>iable<#>. In principle, each circumfix could be described as a gapped underlined expression, but because the superposition algorithm works from left to right, it does not see in the beginning of the word if suffix iable is encountered in the end of the word. This uncertainty generates alternative paths and slows down the algorithm. Often this effect is seen already during the determinization of the input DFA G. A practical solution is to compile prefixes and suffixes as separate entries and then check afterwards that for each bracket <U-> in a prefix there is a matching bracket <U> in a suffix.

6 Discussion

6.1 Efficiency

When the input DFA G is known, the size complexity of the result (17) can be analyzed: The complement of C is in $O(|\Sigma| \cdot |Q|)$. When $|H|$ is the number of states of a recognizer for language $\Sigma^*\diamond S\Sigma^*$, automaton for $H\backslash C$ is in $O(|\Sigma| \cdot |Q| \cdot |H|)$. A $\diamond$-removal of $H\backslash C$ results in size $O(|\Sigma| \cdot (|Q| \cdot |H|)^2)$. Finally, determinizing and complementing automaton $h(H\backslash C)$ results in size $O(|\Sigma| \cdot 2^{(|Q|\cdot|H|)^2)})$. In addition, computing $\Sigma^*\backslash h((\Sigma^* \cup P\diamond_2)\backslash C)$ results

in $O(|Q| + |E|)$. In sum, computing of (17) results in a DFA of size $O(c^{|Q|})$.

In comparison to a standard step-wise approach (=the baseline method) in Fig. 3, the superpose algorithm avoids creating many useless transitions and states because it does not construct state subsets that contain states $(0,1),(0,2),(0,3) \in (Q \cup \{0\}) \times \{1,2,3\}$ of Fig. 3b. These are reached only with strings $w \in h(W \backslash W')$ none of which belongs to the aimed result.

The algorithm was tested with some HunSpell lexicons.[4] The execution time of the superpose algorithm was roughly proportional to the sizes of the input and the output (Table 1) and several orders of magnitude faster than the baseline. With the superpose algorithm, the number of arcs did not typically grow, but the number of states would grow by a small factor before minimization.

6.2 Optimization

We can optimize the superpose algorithm in several ways.

When the automaton G recognizing $C \cup P\diamond_2$ is minimized, information about the prohibitive role of strings $w \in P\diamond_2$ could be used. In particular, final states that have a transition on $\diamond_2$ could be turned non-final, which may cause more pruning to take place during minimization. Optionally, one could substitute $C \cup (h^{-1}(P)\diamond_2)$ for $C \cup P\diamond_2$, which would extend pruning even to strings in $C \backslash h(C)$, but this can actually make G bigger.

The data structures could be improved as well. For example, using failure transitions (Mohri, 1997) reduces the memory footprint of G and optimizes the computation of accessible subsets during the subset construction.

During the two-level grammar compilation procedure (Sect. 5), we could restrict the shared tape of T_L and T_G in the result T_G as follows: (3b) Add markers to the tape that is extracted from T_L *i.e.* $U = \mathrm{Id}(h^{-1}(\mathrm{Range}(T_L)))$. (3c) Restrict the lexical side of C with U: $C \leftarrow U \circ C$. Continue the compilation procedure from (4).

6.3 Possible Extensions

Some generalizations to the semantics of the algorithm would be desirable.

[4]The Hungarian lexicon (hu) used 26 GB in the original format and it had the total alphabet of 301 characters (letters and continuation class symbols). The induced underlined expression took 83 GB and the minimized result automaton 3.2 MB in the SFST file format.

	dic		aff	arcs i/o/m			nodes i/o/m			time b/s	
sw	48		0	98	69	69	28	28	28	756	2
sv	55		.4	291	299	119	58	182	51	19005	17
en	46		1	483	549	116	50	333	48	n/a	95
hu	841		21	7876	2366	359	418	1265	113	n/a	503

Table 1: The total number of root dictionary and affix entries (in *thousands*), DFA transition and state set sizes (input/superpose output/minimized) (in *thousands*), and execution time of baseline/superpose algorithm (in seconds).

We would like to deal with infixes and inserted characters like <D> more abstractly without a need to add them to the root dictionary as optional characters. Such characters should emerge "out of the blue" when needed. The idea could be based on a simplified notion of multi-tape automata where multiple tapes are projected to a single tape.

The current algorithm and its deterministic input are not the most efficient ways to handle nested (Barthélemy, 2007a) or crossing bracketing. A more efficient approach would be based on layered, iterative construction of the result. A layerization method for bracketing constraints is given (Yli-Jyrä and Koskenniemi, 2004), but it assumes that all rules are compiled separately.

The current algorithm cannot compile two-level grammars that have left-arrow conflicts. There are ways to resolve the conflicts when the prohibition rules are combined into one automaton.

There exist already three other algorithms that generalize the idea of superposition to weighted automata (Yli-Jyrä, 2009).

7 Conclusions

The paper presents a new direct algorithm to compilation and combination of context restriction rules. It does not need to know the total alphabet and it is several orders of maginitude faster than a standard, stepwise approach. The presented solution has applications in computational morphology and phonology, predicate logic and in general-purpose finite-state calculus.

In addition, regular expressions with the underline operator are introduced. Underlined expressions and languages are a natural way to describe context restriction rules, context-dependent lexical entries and non-concatenative phenomena.

Acknowledgements

The HunSpell compilation problem was introduced to the author by András Kornai, Péter Halácsy, György Gyepesi, Dániel Varga and Viktor Trón during his visit to BUTE in Hungary.

References

François Barthélemy. 2007a. Multi-grain relations. In *CIAA*, volume 4783 of *LNCS*, pages 243–252. Springer.

François Barthélemy. 2007b. Using Mazurkiewicz trace languages for partition-based morphology. In *Proceedings of ACL 2007*.

Edmund Grimley-Evans, George Anton Kiraz, and Stephen G. Pulman. 1996. Compiling a partition-based two-level formalism. In *16th COLING 1996, Proc. Conference*, volume 1, pages 454–459.

Måns Hulden. 2009. Regular expressions and predicate logic in finite-state language processing. In *Proceedings of FSMNLP 2008*, pages 82–97. IOS Press.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Lauri Karttunen, Kimmo Koskenniemi, and Ronald M. Kaplan. 1987. A compiler for two-level phonological rules. Report CSLI-87-108, Center for Study of Language and Information, Stanford University, CA.

Lauri Karttunen. 1991. Finite-state constraints. In *Proceedings of the International Conference on Current Issues in Computational Linguistics*, Universiti Sains Malaysia, Penang, Malaysia.

Lauri Karttunen. 1993. Finite-state lexicon compiler. Technical Report ISTL-NLTT-1993-04-02, Xerox Palo Alto Research Center.

Lauri Karttunen. 1994. Constructing lexical transducers. In *15th COLING 1994, Proceedings of the Conference*, volume 1, pages 406–411, Kyoto, Japan.

George Anton Kiraz. 1997. Compiling regular formalisms with rule features into finite-state automata. In *35th ACL 1997, 8th EACL 1997, Proceedings of the Conference*, pages 329–336.

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

Kimmo Koskenniemi and Anssi Yli-Jyrä. 2009. Clarin and free open source finite-state tools. In Jakub Piskorski *et al.*, editor, *Finite-State Methods and Natural Language Processing - Post-proceedings of the 7th International Workshop FSMNLP 2008*, volume 191 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Kimmo Koskenniemi. 1983. *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.

Krister Lindén, Miikka Silfverberg, and Tommi Pirinen. 2009. HFST tools for morphology – an efficient open-source package for construction of morphological analyzers. In *Proceedings of SFCM 2009 (to appear)*, Zurich, Switzerland.

Mehryar Mohri. 1997. String-matching with automata. *Nordic Journal of Computing*, 2(2):217–231.

László Németh, Viktor Trón, Péter Halácsy, András Kornai, András Rung, and István Szakadát. 2004. Leveraging the open source ispell codebase for minority language analysis. In *Proceedings of the SALTMIL Workshop at LREC 2004*, pages 56–59.

Helmut Schmid. 2005. A programming language for finite state transducers. In *Pre-proceedings of FSMNLP 2005*, pages 280–281.

Miikka Silfverberg and Krister Lindén. 2009. Conflict resolution using weighted rules in HFST-TWOLC. In *Proceedings of NODALIDA 2009 (to appear)*, Odense, Denmark.

Anssi Yli-Jyrä and Kimmo Koskenniemi. 2004. Compiling contextual restrictions on strings into finite-state automata. In *The Eindhoven FASTAR Days, Proceedings*, Eindhoven, The Netherlands.

Anssi Yli-Jyrä and Kimmo Koskenniemi. 2006. Compiling generalized two-level rules and grammars. In *Proceedings of FinTAL 2006*, LNAI.

Anssi Yli-Jyrä. 2003. Describing syntax with star-free regular expressions. In *11th EACL 2003, Proc. Conference*, pages 379–386, Budapest, Hungary.

Anssi Yli-Jyrä. 2008a. Applications of diamonded double negation. In *Finite-State Methods and Natural Language Processing, 6th International Workshop, FSMNL-2007, Potsdam, Germany, September 14–16, Revised Papers*, pages 6–30. Universitätsverlag, Potsdam.

Anssi Yli-Jyrä. 2008b. Transducers from parallel replacement rules and modes with generalized lenient composition. In *Proceedings of FSMNLP 2007*, Potsdam, Germany, forthcoming.

Anssi Yli-Jyrä. 2009. Context-dependent alignments in ambiguous weighted automata. Manuscript.

ISSN 1736-6305 Vol. 4
http://hdl.handle.net/10062/9206