Conflict Resolution Using Weighted Rules in HFST-TWOLC

Miikka Silfverberg
Department of General Linguistics
University of Helsinki
Helsinki, Finland
miikka.silfverberg@helsinki.fi

Krister Lindén
Department of General Linguistics
University of Helsinki
Helsinki, Finland
krister.linden@helsinki.fi

Abstract

In this article we demonstrate a novel way to resolve conflicts in two-level grammars by weighting the rules. The rules are transformed into probabilistic constraints, which are allowed to compete with each other. We demonstrate a method to automatically assign weights to the rules. It acts in a similar way as traditional conflict resolution, except that traditionally unresolvable left-arrow rule conflicts do not cause lexical forms to be filtered out. The two-level lexicon and probabilistic two-level grammar are combined using the new transducer operation weighted intersecting composition. The result is a weighted lexical transducer. To the best of our knowledge, this is the first time probabilistic rules have been used to solve two-level rule conflicts. The possible applications of probabilistic lexical transducers range from debugging flawed two-level grammars to computer-assisted language learning. We test our method using a two-level lexicon and grammar compiled with the open source tools HFST-LEXC and HFST-TWOLC.

1 Introduction

In a two-level phonological grammar the rules are parallel constraints whose joint effect determines the surface realizations for lexical analyses. A valid correspondence between a lexical string and its surface realization has to be accepted by all of the rules, otherwise it is filtered out (Koskenniemi, 1983).

Situations where correspondences are filtered out because two rules require a lexical form to be realized in two different ways are called rule conflicts. In the worst case, all surface forms corresponding to a lexical analysis are lost and the analysis is filtered out by the grammar.

Conflict resolution is an automated mechanism in two-level rule compilers, which attempts to find conflicting rules and modify them so that no lexical analyses are lost. Traditional conflict resolution can resolve arbitrary conflicts between right-arrow rules, but it is limited to special cases in the case of left-arrow rule conflicts as we explain in section 2.1.

Instead of traditional conflict resolution we propose a method, which builds on making the whole two-level grammar probabilistic in section 2.2. The rules become weighted violable constraints, which are allowed to compete against each other.

Weighting rules is not a new idea. All statistical parser use the idea. However, casting conflict resolution in probabilistic terms is new. We are not aware, of other systems, which use probabilistic rules for conflict resolution. In finite-state syntax, weighted parallel constraints have been considered by Voutilainen (Voutilainen, 1994). Voutilainen considered the use of weighting to order sentence parses by typicality. The weights could be assigned by a linguist or corpus data could be used. The method he proposes is based on penalty weights like our method.

In section 3 we propose a general method for assigning weights to rules. In section 4 we use this method to weight rules in a way, that parallels traditional conflict resolution, when all rule conflicts are solvable. The difference between traditional conflict resolution and our method is, that our method preserves lexical analyses for surface forms even in conflict situations, although these might not have a preferred order, since their weight may be the same. The rule writer may choose to refine the weighting we propose either according to her intuition or by using corpus-data.

The applications of the new kind of conflict resolution might include identifying typical gram-

matical errors made by children and language learners or in information retrieval, since the grammar becomes violable and some erroneous forms are retained, although these are less probable, than their correct counterparts. A very significant use is for debugging two-level grammars. Since forms are not filtered out in a conflict situation, the linguist, who is writing the two-level grammar gets a clearer picture of the way the grammar is broken.

In section 5 we use a new transducer operation, weighted intersecting composition to combine a two-level lexicon and a two-level grammar. Weighted intersecting composition allows the weights in the rules to be passed to the resulting lexical transducer. The operation has been modelled on unweighted intersecting composition, which was introduced by Karttunen (Karttunen, 1994).

We test our method using an example lexicon and grammar in sections 6 and 7. The example concerns gradation of stops in Finnish. The example lexicon was compiled using the open source two-level lexicon compiler HFST-LEXC[1] and the example grammar was compiled using the open source two-level grammar compiler HFST-TWOLC[2]. Both compilers belong to the finite-state morphology toolkit HFST Morphology Tools[3].

2 Rule Conflicts

Rule conflicts occur when two-level rules require, that a lexical symbol is realized in two different ways in the same context. Conflict resolution is a process, which aims to modify the two-level grammar in such a way, that rule conflicts vanish. Traditionally it has been restricted to conflicts between several right-arrow rules or two left-arrow rules. We shall make the same restriction, although conflicts occur in other types of rulesets as well (Yli-Jyrä and Koskenniemi, 2006).

2.1 Traditional Conflict Resolution

A number of right-arrow rules, with equal centers are conflicting if their contexts represent different languages. E.g. the right-arrow rules

$$a{:}b \Rightarrow c \_ \ ; \text{ and } a{:}b \Rightarrow d \ \_ \ ;$$

are in conflict. Since the first one limits the realization of lexical a as surface b into contexts where c precedes and the second one into contexts, where d precedes, the result is that a can't be realized as b anywhere. The reasonable way to interpret the two rules, is that $a{:}b$ should occur either in context $c \_$ or context $d \_$ because we may assume, that the grammar writer intends all rules to be true. Similar considerations apply to other right-arrow rule conflicts as well. Such conflicts may therefore be resolved by joining the conflicting rules into one rule whose context is the union of the contexts of the conflicting rules. Our example becomes

$$a{:}b \Rightarrow c \mid d \_ \ ;$$

Obviously right-arrow conflicts may always be resolved.

In contrast to right-arrow rule conflicts, left-arrow conflicts may not always be resolved. Two left-arrow rules concerning the same lexical symbol are in conflict if they require the symbol to be realized in two different ways in the same context. Consider the rules

$$a{:}b \Leftarrow X \_ \ ; \text{ and } a{:}c \Leftarrow x \_ \ x \ ;$$

where $X = \{x, y, z\}$. In the context $x \_ \ x$ the rules have conflicting requirements.

Here the context of the second rule is subsumed by the context of the first rule, so the second rule may be considered a special case of the first rule (Karttunen et al., 1987). The first rule applies everywhere in the context $X \_$ except in the context $x \_ \ x$. This means that the rule conflict is resolved by modifying the first rule by subtracting the more specific context $x \_ \ x$ from the more general $X \_$. This kind of left-arrow conflicts reflect the fact, that linguists tend to split rules into general tendencies and absolute laws, which are exceptions to those tendencies.

We use the *general restriction* (later GR) operation introduced by Yli-Jyrä (Yli-Jyrä and Koskenniemi, 2004) to compile two-level rules. Rule contexts are compiled into regular expressions, which allows us to operate on them with regular expression operations. Hence, it is possible to resolve a left-arrow conflict by subtracting the context of a sub case rule from the context of a more general rule. It is also possible to resolve right-arrow

[1] For documentation: https://kitwiki.csc.fi/twiki/bin/view/KitWiki/HfstLexC

[2] For documentation: https://kitwiki.csc.fi/twiki/bin/view/KitWiki/HfstTwolC

[3] For downloading HFST programs: http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/sources.shtml

conflicts simply by uniting them into one rule. Its context is the union of the contexts of the conflicting rules.

Left-arrow conflicts between rules neither of which is a sub case of the other were not resolved in the first two-level rule compiler (Karttunen et al., 1987). Nor are they resolved by the current Xerox two-level compiler (Karttunen, 1992). The result is that lexical forms are filtered out by the grammar.

2.2 A Probabilistic Interpretation of Rule Conflicts

Consider a grammar, which has two left-arrow rules R_1 and R_2 concerning the lexical symbol x. The rules are defined

$$x{:}\mathrm{y} \Leftarrow \mathrm{C}_1 \text{ and } \mathrm{x{:}z} \Leftarrow \mathrm{C}_2$$

respectively. If the set $C_1 \cap C_2$ is nonempty, the rules are conflicting.[4]

Clearly rule R_1 should hold vacuously in the set $C_1 \setminus C_2$ and so should R_2 in the set $C_2 \setminus C_1$. We may interpret the situation probabilistically. The rule R_1 should apply with probability 1 in the context $C_1 \setminus C_2$. Similarly rule R_2 should apply with probability 1 in context $C_2 \setminus C_1$. Since we have no information concerning the relative importance of R_1 and R_2, it is reasonable to assume, that the rules have equally high probability p of applying in the context $C_1 \cap C_2$. Traditional conflict resolution corresponds to assigning p the value 0, but one could equally well argue, that p should have the value 0.5.

In a situation where the rule R_1 is stronger than the rule R_2 it may be given higher probability than R_2, which means that surface realizations derived using rule R_2 are more likely than realizations, which are derived using rule R_1.

Generalizing the idea of rules as tendencies and absolute laws, we get a range of rules with different probabilities of applying, from laws which always hold to less certain tendencies. Compiling a lexical transducer becomes the equivalent of letting the different rules compete with each other, which results in an ordering of possible analyses for lexical forms.

[4]We operate with rule contexts like they were regular languages. We do this, because the GR-operation allows us to transform them into regular expressions

3 Conflict Resolution Using Weights

We propose a new method of resolving conflicts, which is based on the idea of making the two-level rules probabilistic. Instead of giving a probability for each rule, we assign a penalty weight for breaking a rule. The principle is, that breaking a more likely rule should always result in a higher penalty, than breaking a less likely one.

The rule R

$$x{:}\mathrm{y} \Leftarrow \mathrm{C}$$

is combined with a penalty weight transducer using weighted union of transducers (Allauzen et al., 2007), so after conflict resolution it becomes the weighted rule R'

$$R \cup (\Sigma^*_{\langle \mathrm{WEIGHT} \rangle})$$

The expression $\Sigma^*_{\langle \mathrm{WEIGHT} \rangle}$ denotes the language of all strings of feasible pairs, where every string receives weight WEIGHT.

In the transducer R', correspondences which break the rule R receive weight WEIGHT and correspondences which do not break the rule R receive weight 0. This happens, because we use the *tropical semi-ring* to represent weights, as we will see in section 5. In the tropical semi-ring addition of weights, performed by the weighted union, corresponds to taking the least of the weights.

The two-level grammar is equivalent to the intersection of the weighted rule transducers. The best paths corresponding to a surface realization for a lexical string are those for which the sum of the weight given by each rule transducer is the lowest.

When two rules compete, the correspondences which break the rule whose penalty weight is lower get a lower penalty than the ones that break a rule with a higher penalty. Conflict resolution thus becomes the task of finding suitable weights for the rules in a grammar.

4 Compiling Two-Level Rules and Weighting them

We now demonstrate a way to weight the rules in a two-level grammar, which will give the same results as ordinary conflict resolution, when all rule conflicts are solvable. The process may be completely automated.

The right-arrow rules will be compiled in the same way as usual, but we need to find penalty

weights associated with the left-arrow rules in the grammar.

We precompile the contexts of all $x{:}y \Leftarrow L \_ R$; into context expressions $L \diamond \Sigma^* \diamond R$, where Σ is the alphabet of the two-level grammar.

We define a relation *context-inclusion* in the grammar. We say that context-inclusion holds between two rules R_1 and R_2,whose centers have the same lexical symbol,

$$x{:}y \Leftarrow Cl_1 \_ Cr_1 \ ; \text{ and } \ x{:}z \Leftarrow Cl_2 \_ Cr_2 \ ;$$

iff

$$(Cl_1 \diamond \Sigma^* \diamond Cr_1) \subset (Cl_2 \diamond \Sigma^* \diamond Cr_2).$$

Context-inclusion is a partial ordering, so we use the symbol $<$ for it and write $R_1 < R_2$.

Let $\{X_1, ..., X_n\}$ be a set of rules in the two-level grammar. The set is a *chain* beginning at X_1, iff

$$X_1 < \ ... \ < X_n.$$

Specifically any set of size 1 is a chain.

We now give each rule R a penalty weight according to the length of the longest chain beginning at R.

Suppose there is a conflict between the rules R and S. If the conflict is resolvable and R is a sub case of S, then $R < S$. Since every chain beginning at S may be extended to a chain beginning at R, breaking the rule R will result in a greater penalty weight than breaking the rule S.

Conflicts, which couldn't be solved using traditional conflict resolution will not result in filtered out lexical forms using our conflict resolution, even though there might not be a preferred order for realizations of a lexical form. This happens because all left-arrow rules are violable.

Other ways of assigning weights for rules might be conceivable, e.g. using corpus data to extract probabilistic two-level rules or to extract lexical contexts in the form of n-grams. This could also be used to fine-tune a grammar obtained using our method. In addition the linguist writing the two-level grammar could assign penalty weights for breaking the rule. The penalty weight v preassigned by the linguist and the weight w given by our conflict resolution may be combined. The actual penalty weight received for breaking the compiled rule becomes $v \otimes w$, where $\otimes$ is the multiplication of the weight semi-ring.

A limitation of our system is, that a correspondence violating a rule receives the same penalty weight regardless of how many positions in the correspondence violate the rule, since equal penalty weights are assigned for all correspondences violating a rule. This might not be a problem in practice, since correspondences exhibiting the same rule conflict multiple times may be rather rare.

weight	surface form
1	ay
1	az
1	bz
2	by

Table 1: surface forms corresponding to ax.

Other possible limitations stem from complex rule-interferences. E.g. let the alphabet Σ be

$$\Sigma = \{a, a{:}b, x{:}y, x{:}z\}$$

and consider the somewhat strange rules

$$x{:}y \Leftarrow a{:} \_ \text{; and } x{:}z \Leftarrow a{:}b \_ \ ;$$

of a two-level grammar with alphabet Σ.

The second rule is a sub case of the first one, so our method of weighting the rules will give it a higher penalty weight. If these are the only rules for the lexical symbol x, then the first rule gets penalty weight 1 and the second gets penalty weight 2. Now, consider the rule

$$a{:}b \Leftarrow \ \_ x{:}y \ ;$$

If it is the only rule concerning the lexical symbol a, it will get the penalty weight 1.

The conceivable realizations for the lexical string ax ordered by weight are given in Table 1.

Among the three best correspondences, there is one, which breaks the third rule, which was not in a left-arrow or a right-arrow conflict with either of the other rules. This example is rather contrived and we are not sure, whether actual phonologies contain these kinds of phenomena.

A further limitation of our system is that, it is possible that surface forms which would not have received any analyses, if they would have been compiled in the normal way, receive analyses. Since our system deals with relative weights, it is not possible to see, if a form is likely to be a good surface form for a lexical form, without generating the best surface forms corresponding to the lexical analysis. We will see an example of this in section 7

5 Combining a Two-Level lexicon and a Probabilistic Two-Level Grammar

As usual, we combine the two-level lexicon and rules to form a lexical transducer. Traditionally this can be done using the operation *intersecting composition* introduced by Karttunen (Karttunen, 1994). Since our rules are weighted, we need to modify the operation slightly. We call the modified operation *weighted intersecting composition*.

The result of unweighted intersecting composition is equivalent to composing the two-level lexicon with the intersection of the two-level rules. The operation was developed, since sequential intersection followed by composition could be rather slow, as computing the intersection of the rule transducers is a resource demanding operation. In intersecting composition the composition and intersections are carried out simultaneously. This allows both the lexicon and the rules to limit the size of the result of the operation without large intermediate results.

Weighted intersecting composition is a modification of intersecting composition for weighted lexicons and rules. The result of the operation is equivalent to weighted composition of the lexicon and the weighted intersection of the rules as defined in (Allauzen et al., 2007).

A probability p can be interpreted as a penalty weight w, using the standard conversion $w = -\log(p)$. This means that high probabilities corresponds to low penalty weights.

We use the tropical semi-ring $\mathcal{T} = (\mathbb{R}_+, \bar{0}, \bar{1}, \oplus, \otimes)$ to represent penalty weights. Here $\mathbb{R}_+$ are the reals, $\bar{0} = \infty$, $\bar{1} = 0$ and the binary operations

$$\oplus : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+ \text{ and } \otimes : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$$

are defined

$$x \oplus y = \min(\{x, y\})$$

and

$$x \otimes y = x + y$$

respectively. The operation $+$ is the regular addition of the reals.

A weighted transducer has one initial state s_0 and may have several final states. A state s is final, if its final weight $s[w] \neq \bar{0} = \infty$

Let e be a transition from state s to state t with the pair $x{:}y$ and weight w. We represent it as a four tuple $e = (s, t, w, p)$ where $s = e[s]$ is the source state, $t = e[t]$ is the target state, $w = e[w]$ is the weight and $p = e[p]$ is the pair $x{:}y$ of the transition.

A *final path* P in the transducer is a sequence of transitions

$$P = e_0, ..., e_n$$

where the source state of e_0 is s_0, i.e. $e_0[s] = s_0$, $e_{i+1}[s] = e_i[t]$ for all $0 \leq i < n$ and the target of the last transition is a final state, i.e.

$$(e_n[t])[w] \neq \bar{0} = \infty.$$

We define the weight of P

$$w(P) = (e_n[t])[w] \otimes \bigotimes_i e_i[w]$$

For a transducer T and pair-string

$$x{:}y = x_1{:}y_1 \ ... \ x_n{:}y_n$$

we define the set of final paths

$$P_{x:y,T} = \{P = (e_0, ..., e_n) \mid e_i[p] = x_i{:}y_i, \ w(P) \neq \bar{0}\}$$

and the weight of $P_{x:y,T}$

$$w(P_{x:y,T}) = \bigoplus_{P \in P_{x:y,T}} w(P).$$

The weight of the string $x{:}y$ is $w(P_{x:y,T})$.

Let $\mathcal{L}$ be a weighted two-level lexicon and $R_1, ..., R_m$ weighted two-level rules. We mark their intersecting composition by

$$\mathcal{L} \circ \cap (R_1, ..., R_m).$$

It contains paths corresponding to a pair-string

$$x{:}z = x_1{:}z_1, ..., x_n{:}z_n,$$

if there are pair-strings

$$x{:}y = x_1{:}y_1, ..., x_n{:}y_n,$$

and

$$y{:}z = y_1{:}z_1, ..., y_n{:}z_n,$$

s.t. $P_{x:y,L} \neq \emptyset$ and $P_{y:z,R_i} \neq \emptyset$ for each i. Following from the definitions of weighted composition and intersection, the weight for $x{:}z$ is

$$w(P_{x:y,L}) \otimes \bigotimes_i w(P_{y:z,R_i}).$$

The weighted rule transducers are more complex, than unweighted ones, so the role of intersecting composition is even bigger in the weighted situation. It might not even be feasible to compute the intersection of weighted rule transducers.

6 A Test Grammar: Gradation of k in Finnish

We tested weighted conflict resolution using a small two-level lexicon and grammar. The grammar consists of three rules governing the gradation of k in Finnish. The grammar we used is a part of a two-level grammar for Finnish gradation of k, p and t, which appeared in (Karttunen et al., 1987).

Gradation is an alternation in the stems of a number of Finnish words. The quality of the final stop k, p or t depends on, whether it is in an open (CV) or closed syllable (CVC). The lenited form of the stop may be a fricative, or the stop may vanish. This is determined by the phonological context.

We use K to mark the morphophoneme participating in k-gradation. In surface forms, it may be realized as 0, j, k, or v. The correspondence K:k is the default correspondence. Our rules govern the realization of K as 0, j and v.

The compiler we use is HFST-TWOLC, an open source two-level rule.compiler, whose syntax is very similar to Xerox TwolC. The sets we use in the rules are We also use the named regular

```
Cons = h j k l n r s t v ;
Vowel = a e i o u ;
Liquid = l r ;
HighLabial = u y ;
```

expressions

```
ClosedOffset =
    Cons: [ Cons: | #:0 ] ;

ClosedCoda =
   Vowel: ClosedOffset ;
```

where ClosedCoda is the coda of a closed syllable.

The grammar has three rules

```
"Gradation of K to 0"
 K:0 <=>
 [h | Liquid | Vowel:] _ ClosedCoda;

"Gradation of k to j"
 K:j <=>
 [Liquid | h] _ [:i | e:] ClosedOffset;

"Gradation of k to v"
 K:v <=>
 Cons :HighLabial _ :HighLabial
 ClosedOffset;
```

All the rules are double-arrow rules and will be split down into a left-arrow rule and a right-arrow rule (this is usually done when two-level rules are compiled).

E.g. the rule `Gradation of K to 0` will be broken down into two sub-rules

```
K:0 <=
[h | Liquid | Vowel:] _ ClosedCoda;

K:0 =>
[h | Liquid | Vowel:] _ ClosedCoda;
```

We call the left-arrow rules, which are formed, L_0, L_j and L_v and the right-arrow rules R_0, R_j and R_v according to the surface symbol in their center. Our grammar now has six rules.

The two-level lexicon we use is defined by the HFST-LEXC file.

```
Multichar_Symbols K +NOUN +NOM +GEN +SG

LEXICON Root

arki+NOUN:arK CASE1 ;
arka+NOUN:arKa CASE2 ;
luku+NOUN:luKu CASE2 ;

LEXICON CASE1

+SG+NOM:0i# # ;
+SG+GEN:en# # ;

LEXICON CASE2

+SG+NOM:0# # ;
+SG+GEN:n# # ;
```

It covers the singular nominative and singular genitive cases of three words *arki* (workday), *arka* (timid) and *luku* (number) exhibiting different kinds of gradation of k. The surface forms, which correspond to the cases, are given by Table 2.

	K:0	K:j	K:v
sg. nom.	arka	arki	luku
sg. gen.	ar0an	arjen	luvun

Table 2: Surface Forms

7 Weighting the rules and Compiling the Test Grammar

We proceed to identifying conflicts. There are no right-arrow conflicts, since all of the rules have different centers. There are two left-arrow conflicts, however. One occurs between the rules L_0 and L_j and the other between L_0 and L_v. There

is no conflict between the rules L_j and L_v, since their contexts are disjoint.

To begin resolving the conflicts in the grammar, we first order the left-arrow rules according to context-inclusion

$$L_j < L_0 \text{ and } \mathrm{L_v} < \mathrm{L_0}$$

We can see that the longest chain starting at L_0 has length 1, since the context of rule L_0 is not included in any other rules. The longest chains beginning at L_j and L_v have length two, since both of the rules are sub cases of the rule L_0.

We now compile the rules using the GR operation and and weight the rule transducer. Weighting the three left-rule transducers, we obtain the weighted transducers

$$\begin{aligned} L'_0 &= L_0 \cup \Sigma^*_{\langle 1\rangle}, \\ L'_j &= L_j \cup \Sigma^*_{\langle 2\rangle} \\ L'_v &= L_v \cup \Sigma^*_{\langle 2\rangle} \end{aligned}$$

Let the transducer obtained by compiling the HFST-LexC file in the example be $\mathcal{L}$. The lexicon $\mathcal{L}$ and the unweighted rules R_0, R_j and R_v may be converted into weighted transducers, following the principle that all transitions get weight $\bar{1}$, all final states get final weight $\bar{1}$ and all other states get final weight $\bar{0}$. Since all states in a weighted transducer get a final weight and only those states, whose final weight is infinite are non-final, we must give the non-final states weight $\bar{0}$.

We now compile the lexical transducer using weighted intersecting composition. It is given by the expression

$$\mathcal{L} \circ \cap\big(R_0, \mathrm{R_j}, \mathrm{R_v}, \mathrm{L}'_0, \mathrm{L}'_\mathrm{j}, \mathrm{L}'_\mathrm{v}\big)$$

The result is a weighted acyclic transducer. The pair-strings it accepts, together with their weights are shown below as pair-strings[5].

As might be expected, the correspondences with weight 0.0 are those, that adhere to of all the rules.

The pair-strings with weight 1.0 are those, that violate the general rule L_0, but don't violate either of the more specific rules L_j and L_v. One of these is `arka+NOUN:0+SG:n+GEN:0` which is erroneous, but there is a better correspondence `ark:0a+NOUN:0+SG:n+GEN:0` with weight 0.0.

[5] A pair-string `a:bcd` corresponds to the string-pair `acd:bcd`

```
WEIGHT  PATH

0.0 ark:0a+NOUN:0+SG:n+GEN:0
0.0 arka+NOUN:0+SG:0+NOM:0
0.0 luku+NOUN:0+SG:0+NOM:0
0.0 arki:0+NOUN:0+SG:i+NOM:0

1.0 arka+NOUN:0+SG:n+GEN:0
1.0 luk:vu+NOUN:0+SG:n+GEN:0
1.0 ark:ji:0+NOUN:0+SG:e+GEN:n

2.0 luk:0u+NOUN:0+SG:n+GEN:0
2.0 ark:0i:0+NOUN:0+SG:e+GEN:n

3.0 luku+NOUN:0+SG:n+GEN:0
3.0 arki:0+NOUN:0+SG:e+GEN:n
```

The forms `ark:ji:0+NOUN:0+SG:e+GEN:n` and `luk:vu+NOUN:0+SG:n+GEN:0` are the correct sg. gen. forms. They are the best correspondences given the lexical strings `arki+NOUN+SG+GEN` and `luku+NOUN+SG+GEN`.

Lexical forms, which occur in a rule conflict, have no unweighted surface realizations. This is because rule conflicts are situations, where it isn't possible to avoid breaking some rule.

The form `arka+NOUN:0+SG:n+GEN:0` demonstrates, that we can get slightly erroneous forms with larger weight, than the best forms. This might be useful e.g. for finding and identifying common errors in the writing of a child or a language learner.

All paths with weight 2.0 or 3.0 are erroneous. The paths with weight 2.0 violate the specific rules, but keep the more general rule. The paths with weight 3.0 violate both a specific rule and the general one. The weight 3.0 is a maximum. There are no paths with weight 5.0, although such paths might be conceivable. This is a consequence of the fact that the rules L_j and L_v never apply at the same time, so a correspondence can't violate both of them.

Note, that the surface form `lujun` of `luku+NOUN+SG+GEN` is not possible, since the right-arrow rule R_j limits the distribution of the pair `K:j`. The right-arrow rules R_0, R_j and R_v hold vacuously.

8 Discussion and Further Work

The test, which we conducted in sections 6 and 7 shows that our method of conflict resolution works. However, a more extensive test should be conducted to ascertain, that the method is practical even when used on complete two-level lexicons and grammars. There is some worry, that the

lexical transducer may become rather large, because it contains both grammatical and ungrammatical forms with different weights. Complex grammars with more intricate interplay between the rules should also be tested.

Since the rules are weighted, standard look up will have to be replaced by an n-best algorithm, which will slow down the parsing process.

Previously improvements to conflict-resolution have been considered by Yli-Jyrä, who proposes an unweighted method for resolving general rule conflicts (Yli-Jyrä and Koskenniemi, 2006). In addition to left-arrow conflicts and right-arrow conflicts, the method also resolves other kinds of rule conflicts, but it diverges from traditional conflict resolution, since it does not prefer rules, which are sub cases of more general rules. Still, it would be interesting to compare our method to the one Yli-Jyrä proposes especially in regard to conflicts, which represent other types of conflicts than right-arrow or left-arrow conflicts.

The uses of our method, without modifications, are probably limited by the fact, that ungrammatical surface forms may receive analyses (as was seen in section 7). It is possible to check the best surface forms matching an analysis, but this means that every surface string for which the best analysis receives a penalty greater than 0.0 requires an extra look up in the lexical transducer. This is feasible for diagnostics purposes e.g. in applications related to computer assisted language learning.

Applications, which require error tolerance could still benefit from our method. Our method might be used both in applications, which test two-level grammars and in computer assisted language learning applications. Increasing recall in information retrieval systems, through error tolerant analysis of queries, is also a possible area of applications. Related to information retrieval is the task of finding corrections for forms, which have been tagged as ungrammatical by a speller.

9 Acknowledgements

We would like to thank Kimmo Koskenniemi, Anssi Yli-Jyrä and the anonymous referees for their comments on the paper. We would also like to thank our colleagues Erik Axelson and Tommi Pirinen in the HFST team for making development of open source finite-state tools an interesting and rewarding experience.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: a general and efficient weighted finite-state transducer library. In Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA 2007). volume 4783 of Lecture Notes in Computer Science, pages 11–23, Prague, Czech Republic, July 2007. Springer-Verlag, Heidelberg, Germany.

Kimmo Koskenniemi. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. University of Helsinki, Department of General Linguistics, Helsinki.

Lauri Karttunen. 1994. *Constructing Lexical Transducers*. The Proceedings of the 15th International Conference on Computational Linguistics COLING 94, I, pages 406–411. Association of Computational Linguistics, Morristown, NJ.

Lauri Karttunen, Kimmo Koskenniemi and Ronald M. Kaplan. 1987. A Compiler for Two-level Phonological Rules. In Dalrymple, M. et al. Tools for Morphological Analysis. Report CSLI-87-108. Center for the Study of Language and Information. Stanford University.

Lauri Karttunen. 1992. *CA:Two-Level Rule Compiler - Xerox XRCE*. `http://www.xrce.xerox.com/competencies/content-analysis/fssoft/docs/twolc-92/twol92.html`

Atro Voutilainen. 1994 *Designing a Parsing Grammar*. University of Helsinki, Department of General Linguistics, Helsinki.

Anssi Yli-Jyrä and Kimmo Koskenniemi. 2004. *Compiling Contextual Restriction on Strings into Finite-State Automata*. In L. Cleophas and B. W. Watson, eds., Proceedings of the Eindhoven FASTAR Days 2004. Computer Science Reports 04/40. The Netherlands: Technische Universiteit, Eindhoven.

Anssi Yli-Jyrä and Kimmo Koskenniemi. 2006. *Compiling Generalized Two-Level Rules and Grammars* . In Advances in Natural Language Processing, Springer Berlin/Heidelberg