Implementation of replace rules using preference operator

Senka Drobac, Miikka Silfverberg, and Anssi Yli-Jyrä
University of Helsinki
Department of Modern Languages
Unioninkatu 40 A
FI-00014 Helsingin yliopisto, Finland
{senka.drobac, miikka.silfverberg, anssi.yli-jyra}@helsinki.fi

Abstract

We explain the implementation of replace rules with the .r-glc. operator and preference relations. Our modular approach combines various preference constraints to form different replace rules. In addition to describing the method, we present illustrative examples.

1 Introduction

The idea of HFST - Helsinki Finite-State Technology (Lindén et al. 2009, 2011) is to provide open-source replicas of well-known tools for building morphologies, including XFST (Beesley and Karttunen 2003). HFST's lack of replace rules such as those supported by XFST, prompted us to implement them using the present method, which replicates XFST's behavior (with minor differences which will be detailed in later work), but will also allow easy expansion with new functionalities.

The semantics of replacement rules mixes contextual conditions with replacement strategies that are specified by replace rule operators. This paper describes the implementation of replace rules using a preference operator, *.r-glc.*, that disambiguates alternative replacement strategies according to a preference relation. The use of preference relations (Yli-Jyrä 2008b) is similar to the *worsener* relations used by Gerdemann (2009). The current approach was first described in Yli-Jyrä (2008b), and is closely related to the matching-based finite-state approaches to optimality in OT phonology (Noord and Gerdemann 1999; Eisner 2000). The preference operator, *.r-glc.*, is the reversal of generalized lenient composition (glc), a preference operator construct proposed by Jäger (2001). The implementation is developed using the HFST library, and is now a part of the same.

The purpose of this paper is to explain a general method of compiling replace rules with *.r-glc.* operator and to show how preference constraints described in Yli-Jyrä (2008b) can be combined to form different replace rules.

2 Notation

The notation used in this paper is the standard regular expression notation extended with replace rule operators introduced and described in Beesley and Karttunen (2003).

In a simple rule

$$a\ op\ b\ dir\ L_1 \_ R_1, \ldots, L_n \_ R_n$$

op is a replace rule operator such as: $\rightarrow, (\rightarrow), @\rightarrow, @>, \leftarrow, (\leftarrow), \ldots$; $a \subseteq \Sigma^*$ is the set of patterns in the input text that are overwritten in the output text by the alternative patterns, which are given as set $b \subseteq \Sigma^*$, where Σ^* is a universal language and Σ set of alphabetical symbols; L_n and R_n are left and right contexts and *dir* is context direction (||, //, \\ and \/).

Rules can also be parallel. Then they are divided with double comma (,,), or alternately with single comma if context is not specified.

Operation	Name
X Y	The concatenation of Y after X
X | Y	The disjunction of X and Y
X:Y	The cross product of X and Y, where X and Y denote languages
X .o. Y	The composition of X and Y, where X and Y denote relations
X^+	The Kleene plus
X^*	The Kleene star
$proj_1(X)$	The projection of the input language of the relation X
$proj_2(X)$	The projection of the output language of the relation X

Table 1 – List of operations

Operators used in the paper are listed in Table 1, where X and Y stand for regular expressions.

Additionally, parenthesis ( ) are used to mark optionality, squared brackets [ ] for precedence and question mark ? is used to denote set Σ in regular expressions.

3 Method

The general idea for compiling replace rules with the *.r-glc.* operator and preference constraints is shown in Figure 1.

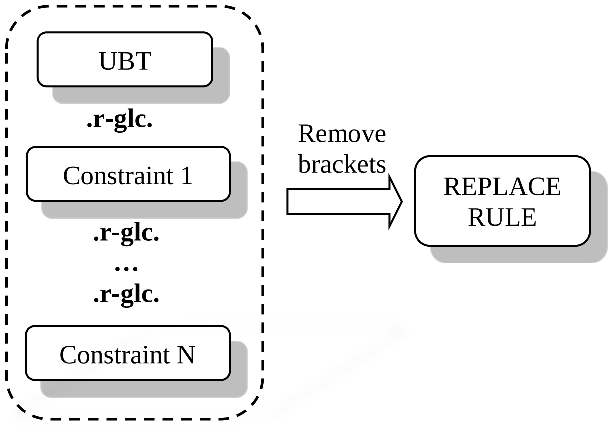

Figure 1: General method of building a replace rule

The method consists of the following steps:

1. Building an Unconstrained Bracketed Transducer (UBT) – a transducer which applies or skips contextually valid replacements freely in all possible portions of the inputs. Every application of the replacement rule is marked with special brackets. Similar replace rules that differ only with respect to their replacement strategies will use the same UBT. Thus, the compilation of UBT is independent of the replacement strategy, which increases the modularity of the compilation algorithm.
2. Implement the functionality of the replace rule operator by constraining the UBT with the respective preference relation.
3. Remove brackets from the transducer.

The major advantage of this method is its modularity. The algorithm is divided into small components which are combined in the desired way. This approach allows every part of the algorithm to be separately and clearly defined, tested and changed. Furthermore, modularity makes it possible to easily integrate new functionalities such as weighted replace rules or two level contexts.

3.1 Unconstrained Bracketed Transducer

As mentioned earlier, it is first necessary to build the UBT. This step can be seen as a variant of Yli-Jyrä and Koskenniemi's (2007) method for compiling contextually restricted changes in two-level grammars. The main difference now is that the rule applications cannot overlap because they will be marked with brackets.

Step 1: Bracketed center

The first step is to create a bracketed center, $center_B$ – the replace relation surrounded by brackets $\{\langle, \rangle\}$. For optional replacement, it is necessary that $center_B$ also contains the upper side of the relation bracketed with another pair of brackets $B_2 = \{[\![,]\!]\}$. This is necessary for filtering out all the results without any brackets (see later filter T_{MOST+}) and getting non optional replacement.

$$center_B = \bigcup_{i \le n} \langle a_i : b_i \rangle \cup [\![a_i]\!]$$

In case of parallel replace rules, bracketed center is the union of all individual bracketed centers. Like XFST, this implementation requires parallel replace rules to have the same replace operator (and optionality) in all replacements.

Step 2: The change centers in free context

The second step is to expand bracketed center to be valid in any context.

If $B = \{\langle, \rangle, [\![,]\!]\}$, we can define:

$$\mathcal{U} = [\Sigma - B \cup center_B]^*$$

Then, center in free context is:

$$center_{FREE} = \mathcal{U} \diamond center_B \, \mathcal{U}$$

where ⋄ is diamond, which is used to align centers and contexts during compilation.

Step 3: Expanded center in context

The next step is to compile contexts. The method used for constructing $center_{CONTXT}$ depends on whether the context must match on the upper or the lower side. Since it is possible to have multiple contexts, each replacement should be surrounded with all applicable contexts:

$$center_{CONTXT} = c_1 | \, c_2 \, | \, ... \, | \, c_n$$

Center surrounded with one context is:

$$c_1 = [L \mid L'] \diamond center_B \, [R \mid R'],$$

where L and R are left and right contexts from the replace rule, and L' and R' are expanded contexts, depending on which side the context matches. In the case when context must match on the upper side, L' and R' are:

$$L' = \big[[\mathcal{U}\, L] \ll \mathrm{B}\,\big]\,.\mathrm{o.}\ \mathcal{U}$$
$$R' = \big[[R\, \mathcal{U}] \ll \mathrm{B}\big]\,.\mathrm{o.}\ \mathcal{U}$$

If they must match on the lower side:

$$L' = \mathcal{U}.\mathrm{o.}\big[[\mathcal{U}\, L] \ll \mathrm{B}\big]$$
$$R' = \mathcal{U}.\mathrm{o.}\big[[R\, \mathcal{U}] \ll \mathrm{B}\big]$$

where brackets are freely inserted ($\ll$) in the contexts and then composed with $\mathcal{U}$.

In this example:

$$a \ \rightarrow\ b\ \parallel c\ \_\ d$$

both contexts should match on the upper side of the replacement, so $center_{CONTXT}$ is:

$$L' = \big[[\mathcal{U}\ \mathrm{c}] \ll \mathrm{B}\,\big]\,.\mathrm{o.}\ \mathcal{U}$$
$$R' = \big[[\mathrm{d}\ \mathcal{U}] \ll \mathrm{B}\big]\,.\mathrm{o.}\ \mathcal{U}$$
$$c_1 = (\, c \mid L') \ \ \diamond\ \langle a : b\rangle \cup [\![a]\!]\ (\, d \mid R')$$
$$center_{CONTXT} = c_1$$

This way of compiling contexts allows every rule in a parallel replace rule to have its own context direction (||, //, \\, \/). Therefore, rules like the following one are valid in this implementation:

$$a\ \rightarrow\ b \backslash\backslash\, c\ \_\ d\ ,,\, b\ \rightarrow\ c\ //\ c\ \_\ d$$

Steps 4: Final operations

Finally, to get the unconstrained replace transducer it is necessary to subtract $center_{CONTXT}$ from $center_{FREE}$, remove diamond and do a negation of that relation.

Let $V = \big[[\Sigma - B - \diamond]\ \cup center_B\big]^*$, then:

$$V\ - d_{\diamond}[\, center_{FREE} - center_{CONTXT}\,]$$

where $d_{\diamond}$ denotes removal of diamond.

3.2 Constraints

All the preference constraints were defined in Yli-Jyrä (2008), but since they were mostly difficult to interpret and implement, here is the list of the constraints written with regular expressions over the set of finite binary relations.

First, let us define RP – a regular expression often used in the restraints:

$$RP = \ \ [\, B\!:0 \mid 0\!:\!B \mid ? - B\,]^*$$

The left most preference is achieved by:

$$T_{LM} = \ ?^* <\!:0\ [B\!:0]^*[?\ -B]\ RP$$

Right most preference:

$$T_{RM} = \ RP\ [? - B]^+\ \ \rangle : 0\ ?^*$$

Longest match left to right:

$$L_L = \ [? - B] \mid [0\!:\langle \mid 0\!:\langle \mid \langle :0 \mid B\,][? - B]^+$$
$$T_{LRLONG} = ?^*\ \langle\ \ [?\ - B\,]^+\ \ 0 : \rangle\ L_L\ RP$$

Longest match right to left:

$$L_R = \ [? - B] \mid [? - B]^+\ [0\!:\langle \mid 0 :\ \ \rangle \mid\ \rangle : 0 \mid B\ ]$$
$$T_{RLLONG} = \ RP\ L_R\ 0 : \langle\ \ [?\ - B\,]^+ \rangle\, ?^*$$

Shortest match left to right:

$$S_L = \ [? - B] \mid [\ 0 : \langle \mid\ \rangle : 0 \mid \langle\!:\!0 \mid B\,][? - B]^+$$
$$T_{LRSHORT} = ?^*\ \langle\ \ [?\ - B\,]^+\ \ \ \rangle : 0\ \ S_L\ RP$$

Shortest match right to left:

$$S_L = \ [? - B] \mid [? - B]^+\ [\ 0 : \langle \mid\ \rangle : 0 \mid \langle\!:\!0 \mid B\ ]$$
$$T_{RLSHORT} = \ RP\ S_R\ \langle : 0\ [?\ - B\,]^+\langle\ \ \ ?^*$$

For compiling epenthesis rules, to avoid more than one epsilon in the row:

$$B_L = \{\langle\, , [\![\ \}$$
$$B_R = \{\ \rangle,\ ]\!]\}$$
$$T_{NO\ REP}{}' = ?^*\ \ B_L B_R B_L B_R\ ?^*$$

For non-optional replacements:

$$T_{MOST+} = ?^*\ \ [B_L\!: 0\ \ [? - B]^+\ B_R\!: 0\ ?^*\ ]^+$$

To remove paths containing B_2, where $B_2 = \{\ [\![,]\!]\}$:

$$T_{LEST}{}' = ?^*\ B_2\ ?^*$$

Since $T_{NO\ REP}{}'$ and $T_{LEST}{}'$ are reflexive, they are not preference relation. Instead, they are filters applied after preference relations.

3.3 Applying constraints with *.r-glc.* operator

To apply a preference constraint in order to restrict transducer *t*, we use *.r-glc.* operator. The *.r-glc.* operation between transducer *t* and a *constraint* is shown in Figure 2. Input language of a transducer is noted as *proj₁* and output language as *proj₂*.

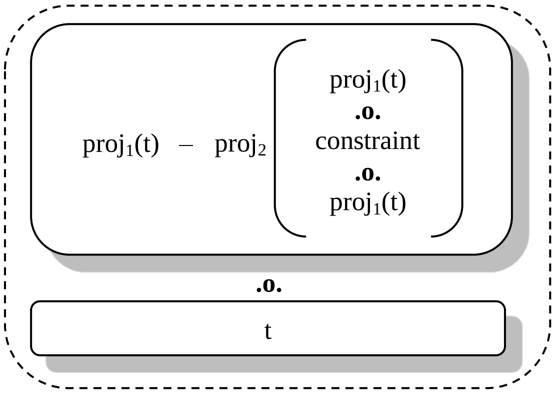

Figure 2: Breakdown of the operation: *t **.r-glc.** constraint*

Contraints combinations

As shown in Figure 1, in order to achieve desired replace rules, it is often necessary to use several constraints. For example, to achieve left to right longest match, it is necessary to combine T_{LM} and

T_{LRLONG}. If the same longest match contains epenthesis, $T_{NO\ REP}$ constraint should also be used.

3.4 Removing brackets

Removing brackets is simply achieved by applying $T_{LEST} = ?^{*}\ B\ ?^{*}$ constraint, where B is set of brackets we want to remove. Additionally, in HFST implementation, it is also required to remove the brackets from the transducers alphabets.

4 Examples

Let us show how the replace rule is compiled on different examples.

Since it would take too much space to show whole transducers, we will show only output of the intermediate results applied to an input string.

The first example shows how to achieve a non-optional replacement. Intermediate results of the replace rule $a \rightarrow\ x\ ||\ a\ \_\ a$ is shown in the Table 2. Since the arrow demands non-optional replacement, the unconstrained bracketed replace, if applied to the input string $a\ a\ a$, contains three possible results. The first result is the input string itself, which would be part of the non-optional replacement. The second result is necessary to filter out the first one. In this example, because of the restricting context, replacement is possible only in the middle, and therefore, it is bracketed with special brackets. Finally, the third result contains the bracketed replace relation.

$a \rightarrow\ x\ ||\ a\ \_\ a$

UBT	T_{MOST+}	T_{LEST}'
$a\ a\ \ a$ $a\ [\![a]\!]\ a$ $a\langle a\!:x\rangle\ a$	$a\langle a\!:x\rangle\ a$ $a\ [\![a]\!]\ a$	$a\langle a\!:x\rangle\ a$

Table 2: Steps of the non optional replacement

Once when we have the unconstrained bracketed replace transducer, we are ready to apply filters. First filter, T_{MOST+} will filter out all results that contain smaller number of brackets in every position, without making difference to the type of brackets. In this example, it will filter out the first result, the one that does not have any brackets at all.

The second filter, T_{LEST}' will filter out all the results containing B_2 brackets because they don't contain the replace relation. Finally, to get the final result, it is necessary to remove brackets from the relation.

Following examples will be shown on the input string $a\ a\ a\ a$. Table 3 shows steps of building left to right longest match and Table 4 left to right shortest match.

Both longest match and shortest match have the same first two steps. After building Unconstrained Bracketed Replace, we apply T_{LM} filter which finds all the results with left most brackets in every position and filters out all the rest. This contraints characteristic filters out the results without the brackets as well, so the result will be non-optional. In order to get the longest match, we apply another filter (T_{LRLONG}) to the result of the left most filter. This filter finds the longest of the bracketed matches with the same starting position. In the final step, if we apply filter $T_{LRSHORT}$ instead of T_{LRLONG}, we will get the shortest match (Table 4).

$a+\ @\!\!\rightarrow x\ ||\ a\ \_\ a$

UBT	T_{LM}	T_{LRLONG}
$a\ a\ a\ a$ $a\langle a\!:x\rangle\ a\ a$ $a\langle a\!:x\rangle\langle a\!:x\rangle\ a$ $a\langle a\!:x\ a\!:\varepsilon\rangle\ a$ $a\ a\ \langle a\!:x\rangle\ a$	$a\langle a\!:x\rangle\ a\ a$ $a\langle a\!:x\rangle\langle a\!:x\rangle\ a$ $a\langle a\!:x\ a\!:\varepsilon\rangle\ a$	$a\langle a\!:x\ a\!:\varepsilon\rangle\ a$

Table 3: Left to right longest match

$a+\ @\!> x\ ||\ a\ \_\ a$

UBT	T_{LM}	$T_{LRSHORT}$
$a\ a\ a\ a$ $a\langle a\!:x\rangle\ a\ a$ $a\langle a\!:x\rangle\langle a\!:x\rangle\ a$ $a\langle a\!:x\ a\!:\varepsilon\rangle\ a$ $a\ a\ \langle a\!:x\rangle\ a$	$a\langle a\!:x\rangle\ a\ a$ $a\langle a\!:x\rangle\langle a\!:x\rangle\ a$ $a\langle a\!:x\ a\!:\varepsilon\rangle\ a$	$a\langle a\!:x\rangle\langle a\!:x\rangle\ a$

Table 4: Left to right shortest match

5 Conclusion

The large number of different replace operators makes it quite complicated and error-prone to build a supporting framework for them. However, the *.r-glc.* operator and preference relations allow splitting the algorithm into small reusable units which are easy to maintain and upgrade with new functionalities.

The replace rules are now part of the HFST library and can be used through *hfst-regexp2fst* command line tool, but there is still some work to be done to build an interactive interface. Additionally, we are planning to add support for two level contexts and parallel weighted rules.

Acknowledgments

The research leading to these results has received funding from the European Commission's 7th Framework Program under grant agreement n° 238405 (CLARA).

References

Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI publications (2003)

Eisner, J.: Directional constraint evaluation in optimality theory. In: 20th COLING 2000, Proceedings of the Conference, Saarbrücken, Germany (2000) 257–263

Gerdemann, D. (2009). Mix and Match Replacement Rules. Proceedings of the Workshop on RANLP 2009 Workshop on Adaptation of Language Resources and Technology to New Domains, Borovets, Bulgaria, 2011, pages 39-47.

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)

Gerdemann, D., van Noord, G.: Transducers from rewrite rules with backreferences. In: 9th EACL 1999, Proceedings of the Conference. (1999) 126–133

Jäger, G.: Gradient constraints in Finite State OT: The unidirectional and the bidirectional case. In: Proceedings of FSMNLP 2001, an ESSLLI Workshop, Helsinki (2001) (35–40)

Karttunen, L.: The replace operator. In: 33th ACL 1995, Proceedings of the Conference, Cambridge, MA, USA (1995) 16–23

Karttunen, L.: Directed replace operator. In Roche, E., Schabes, Y., eds.: Finitestate language processing, Cambridge, Massachusetts, A Bradford Book. The MIT Press (1996) 117–147

Kempe, A., Karttunen, L.: Parallel replacement in finite state calculus. In: 16th COLING 1996, Proc. Conf. Volume 2., Copenhagen, Denmark (1996) 622–627

Lindén, K., Axelson, E., Hardwick, S., Silfverberg, M., Pirinen, T.: HFST - Framework for Compiling and Applying Morphologies, Communications in Computer and Information Science, vol. 100, pp. 67-85. Springer Berlin Heidelberg (2011)

Lindén, K., Silfverberg, M., Pirinen, T.: Hfst tools for morphology - an efficient open-source package for construction of morphological analyzers. In: Mahlow, C., Pietrowski, M. (eds.) State of the Art in Computational Morphology. Communications in Computer and Information Science, vol. 41, pp. 28-47. Springer Berlin Heidelberg (2009)

Yli-Jyrä, A., Koskenniemi, K.: A new method for compiling parallel replacement rules. In Holub, J., Ždárek, J., eds.: Implementation and Application of Automata, 12th International Conference, CIAA 2007, Revised Selected Papers. Volume 4783 of LNCS., Springer (2007) 320–321

Yli-Jyrä, A.: Applications of Diamonded Double Negation. In Finite-state methods and natural language processing. Thomas Hanneforth and Kay-Michael Würtzner. 6th International Workshop, FSMNLP 2007. Potsdam, Germany, September 14-16. Revised Papers. Universitätsverlag Potsdam (2008a) 6-30

Yli-Jyrä, A., Transducers from Parallel Replace Rules and Modes with Generalized Lenient Composition. In Finite-state methods and natural language processing. Thomas Hanneforth and Kay-Michael Würtzner. 6th International Workshop, FSMNLP 2007. Potsdam, Germany, September 14-16. Revised Papers. Universitätsverlag Potsdam (2008b) 197-212