A compiler from XFST to FSA

Project documentation

Yael Cohen-Sygal

Laboratory in natural languages processing – winter semester 2002-2003

Contents

1 Introduction

1.1 Finite state technology

Finite-state technology is widely considered to be the appropriate means for describing the phonological and morphological phenomena of natural languages.

Finite state technology has some important advantages, making it most appealing for implementing natural language morphology. One can find it very hard, almost impossible, to build the full automaton or transducer describing some morphological phenomena. This difficulty arises from the fact that there are a great number of morpho-phonological processes combining together to create the full language. However, it is usually very easy to build a finite state machine to describe a specific morphological phenomenon. The class of regular languages is closed under union, concatenation, intersection and complementation. The class of regular relations is closed under union, concatenation and composition (but not under intersection and therefore complementation). These properties make it most convenient to implement each phenomenon independently and combine them together using the closure properties. Moreover, finite state techniques have the advantage of being efficient in their time and space complexity, as the membership problem is solvable in time linear in the length of the input. Moreover, there are known algorithms for minimizing and determinizing automata and some restricted kinds of transducers.

Several FS "toolboxes" exist which facilitate the stipulation of phonological and morphological rules by extending the language of regular expressions with additional operators. Such toolboxes typically include a language for extended regular expressions and a compiler from regular expressions to finite-state devices (automata and transducers). Unfortunately, there are no standards for the syntax of extended regular expression languages.

1.2 The goal of this project

The goal of this project is to design and implement a compiler which will translate grammars, expressed in the XFST finite-state toolbox of Xerox, to grammars in the language of the FSA Utils package. For the most part, there is a strong parallelism between the languages, but certain constructs are harder to translate and require more innovation.

The contribution of this project lies in the fact that the Xerox utilities are proprietary; compilation to FSA enables us to use grammars developed with the Xerox tools on publicly available systems. Furthermore, this project offers parallel investigation of two similar, yet different, systems. Finally, such a compiler enables us to compare the performance of the two systems on very similar benchmarks.

1.3 The basic design of this project

The XFST grammar is proprietary and therefore can not be obtained, thus, the first step towards creating a compiler was to extract and reconstruct the grammar of XFST and represent it (as possibly using the available documentation and existing software) in some formal grammar. Next, each XFST construct was translated into FSA. Part of the XFST constructs could be translated easily into FSA since the FSA consists of equivalent constructs (although with a different syntax), but some which does not exist in the FSA Utils had to be implemented from scratch (mainly all the replacement,restriction and markup rules).

2 XFST operators and their translation into the FSA Utils

In this section all XFST operators are presented. For each operator we explain its meaning and show how to translate it into the FSA Utils syntax. For the part of XFST operators that have an equivalent operator in FSA, we simply describe the syntax of the equivalent FSA operator; For the part that has no equivalent operator in FSA we describe for each operator how it can be constructed from the operators that have already been translated. Notice that when translating using previously translated operators, we write the previously translated operators using the XFST syntax (and not the FSA). It is done for simplicity reasons, and since we assume that most of the readers of this paper are more proficient in XFST syntax than in FSA syntax. We have not been able to find the equivalent translation from XFST into FSA for all XFST operators, mainly since we could not obtain full documentation for the creating algorithms for all XFST operators, and in particular for the more complicated ones (e.g., restriction, markup and replace rules). For some operators the translation could not be done from other reasons that will be specified for each case separately.

2.1 Symbols

- XFST syntax: a

  Meaning: Single symbol a.

  Translation into FSA syntax: 'a'. The characters ' ' function as escape characters in FSA. We use the escape character to avoid of cases were a legitimate XFST character is not so in FSA without the escape character (a good example are capitals letters that must be escaped in FSA but not in XFST).

- XFST syntax: $\%$

  Meaning: Escape character. The escape character, $\%$, eliminates any special meaning of the following character. Thus $|$, 0, ?, etc. can be used as ordinary symbols by prefixing them with $\%$. The ordinary percent sign can be expressed by $\%\%$.

  Translation into FSA syntax: The escape character $\%$ is eliminated since FSA uses ' ' as escape syntax, and any string or character is escaped when being translated into FSA to avoid mistranslation. In the case of literally question mark in XFST ($\%?$) the translation into FSA is escape(?) and not '?' since '?' is not accepted by FSA.

- XFST syntax: $\%+$

  Meaning: Single symbol $+$. The $\%$ means that the following symbol will be taken literally, otherwise, $+$ would be interpreted as a regular expression operator. Note that $\%a$ is the same as a.

  Translation into FSA syntax: '+'. The explanation is as above.

- XFST syntax: "a"

  Meaning: Single symbol a. Double quotes have no special meaning for symbols composed of ordinary letters.

  Translation into FSA syntax: 'a'. The double quotes are replaced with ' ', the escape syntax of FSA.

- XFST syntax: $\% + ab\% -$

  Meaning: Multi-character symbol +ab-. The % means that the following symbol will be taken literally. Otherwise, + and - would be interpreted as regular-expression operators.

  Translation into FSA syntax: '+ab-'. The escape syntax is added as explained above and the XFST escape character $\%$ is removed except in cases where $\%\%$ appears and then only one is removed (since in XFST $\%\%$ means the literal percent).

- XFST syntax: $''+ab-''$

  Meaning: Multi-character symbol +ab-. Symbols enclosed in double quotes can contain characters that otherwise are used as operators (&, |, -, +, * etc.).

  Translation into FSA syntax: '+ab-'. The double quotes are replaced with the escape syntax of FSA ' ' as explained above.

- XFST syntax: ?

  Meaning: Any symbol. If interpreted as a pair, ? denotes the identity relation. Note that in a displayed network ? only represents unknown symbols. In a regular expression, ? in addition covers all symbols that explicitly occur in the regular expression.

  Translation into FSA: ?

- XFST syntax: 0 or [ ] or $''$ $''$

  Meaning: Epsilon symbol, the empty string.

  Translation into FSA: [ ].

- XFST syntax: $\{abcd\}$

  Meaning: Single character brace. Each character between the braces is taken as a single-character symbol. For example $\{a\ bc\}$ will be represented by the concatenation of four single symbols, a,$''$ $''$ (space), b and c rather then the concatenation of the single symbol a and the multi character symbol bc when there were no braces around.

  Translation into XFST: [a,b,c,d] as [,] is the union operator in FSA. The % character in XFST functions as an escape character also in this context, thus he is removed as explained above.

2.2 Simple operators

- XFST syntax: $A*$

  Meaning: Kleene star. The language or relation A concatenated with itself any number of times, including zero times. Note that ?* denotes the universal language.

  Translation into FSA: $A*$

- XFST syntax: $A+$

  Meaning: Iteration (Kleene plus). The language or relation A concatenated with itself one or more times. A+ includes [A], [A A], [A A A], and so on *ad infinitum*. ?+ is the language of all nonempty strings.

  Translation into FSA: $A+$

- XFST syntax: $A \mid B$

  Meaning: Union of the languages A and B. Set of strings that are in A or in B or in both languages.

  Translation into FSA: $\{A, B\}$

- XFST syntax: $A\ \&\ B$

  Meaning: Intersection of the languages A and B. Set of strings that are in A and in B.

  Translation into FSA: $A\ \&\ B$

- XFST syntax: $A - B$

  Meaning: A minus B. Set of strings that are in A and not in B.

  Translation into FSA: $A - B$

- XFST syntax: $\backslash A$

  Meaning: Term complement. The set of all single-symbol strings that are not in the language A. $\backslash A$ is equivalent to $[? - A]$. Thus $\backslash a$ contains "b" and "c" but not "a" or the empty string, and no strings with more than one symbol like "aa" or "aaab". Note that $\backslash ?$ denotes the null language because ? covers all single-symbol strings.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by $[\sim A]$ & ?

- XFST syntax: $\sim A$

  Meaning: Complement. The set of all strings that are not in the language A. For example, $\sim a$ contains " ", "aa", and "aaa" but not "a". The complement of the null language, $\sim \backslash ?$, is the universal language, and vice versa.

  Translation into FSA: $\sim A$

- XFST syntax: A/B

  Meaning: A ignoring B. Set of all strings that would be in A if all substrings of B (regarded as noise) were removed. (a/x contains "a", "xa", "xaxx", etc.)

  Translation into FSA: ignore(A,B).

- XFST syntax: $A./.B$

  Meaning: A ignoring internally B. Definition: A./.B = [A/B] - [B ?* | ?* B] (e.g. [a a a]/x contains "aaa", "aaxa", "axxxaxa", etc.)

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by [A/B] - [B ?* | ?* B]

- XFST syntax: $\$A$

  Meaning: Containment. The set of strings or string pairs containing at least one instance of A as a subpart. For example, "bac" is a string in the language denoted by $\$a$, speaking informally, "bac" is in $\$a$. Here A can denote a relation. A is equivalent to $[? * A ?*]$. $[ ] denotes the universal language.

  Translation into FSA: $\$A$

- XFST syntax: $\$.A$

  Meaning: Contains one. Set of all strings containing exactly one occurrence of a string from the language A.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by $[\$A]$ - [[A A]/[?*]]

- XFST syntax: $\$?A$

  Meaning: Contains at most one. Set of all strings containing at most one occurrence of a string from the language A.

Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by [[$A] – [[AA] / [?*]]] | [[?*] – [$A]]. This is the union of the language that contains exactly one A and the language that does not contain A at all.

- XFST syntax: $A\ B$

  Meaning: Concatenation of the languages A and B. (a b is the concatenation of the two single-symbol strings ”a” and ”b”. However, ab without a blank in between is a string of the single multi-character symbol ”ab”.)

  Translation into FSA: [A,B]

- XFST syntax: $A\hat{}n$

  Meaning: n-ary concatenation of the language or relation A with itself. The set of strings or pairs of strings obtained by concatenating A with itself n times. For example $A\hat{}3$ is equivalent to [A A A].

  Translation into FSA: This operator does not exist in FSA but can be constructed using prolog macro. Basically, it is equal to concatenating A n times.

- XFST syntax: $A\hat{}\{n,k\}$

  Meaning: n to k concatenations of A. For example $A\hat{}\{3,5\}$ is equivalent to $A\hat{}3 \mid A\hat{}4 \mid A\hat{}5$.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by $A\hat{}n \mid A\hat{}(n+1) \mid ... \mid A\hat{}k$

- XFST syntax: $A\hat{} > n$

  Meaning: More than n concatenations of A.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by $[A\hat{}n]\ [A+]$.

- XFST syntax: $A\hat{} < n$

  Meaning: Less than n concatenations of A.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by [ ] | $A\hat{}1 \mid A\hat{}2 \mid ... \mid A\hat{}(n-1)$.

- XFST syntax: $A.x.B$

  Meaning: Crossproduct of the regular languages A and B. It denotes a regular relation that maps every string in A (upper side) to all strings in B (lower side), and vice versa.

  Translation into FSA: A x B

- XFST syntax: $A.o.B$

  Meaning: Composition of two regular relations A and B. The upper language of A gets directly related to the lower language of B. The intersection of the lower language of A and the upper language of B works like a filter without being present in the result of the composition. (e.g. [a:i | b:j] .o. [i:x | k:y] gives the result [a:x] because of a:i and i:x. Everything else is filtered out by the intersection [i | j] & [i | k] allowing only i on the (disappearing) intermediate level.)

  Translation into FSA: A o B

- XFST syntax: (A)

  Meaning: Optionality (union of the language A with the empty language).

  Translation into FSA: Aˆ

- XFST syntax: $a : b$

  Meaning: Symbol pair with a in the upper language and b in the lower language.

  Translation into FSA: $a : b$

- XFST syntax: $[A]$

  Meaning: Same language as A. These brackets control the order in which operations are performed.

  Translation into FSA: (A)

- XFST syntax: $R.P.Q$

  Meaning: Upper-side priority union. Union of all string pairs from the relation R with all those string pairs from Q that have an upper string that is not in R. Alias: R .|u. Q (Definition: R .|u. Q = R | [Q .-u. R]).

  Translation into FSA: This operator was not translated since its precedence comparing to the other operators is not known. Moreover, it is rarely used.

- XFST syntax: $R.p.Q$

  Meaning: Lower-side priority union. Union of all string pairs from the relation R with all those string pairs from Q that have a lower string that is not in R. Alias: R .|l. Q (Definition: R .|l. Q = R | [Q .-l. R]).

  Translation into FSA: This operator was not translated since its precedence comparing to the other operators is not known. Moreover, it is rarely used.

- XFST syntax: $R. - u.Q$

  Meaning: Upper-side minus. Set of all string pairs from the relation R minus those that have an upper string that is also in Q. (Definition: R .-u. Q = [R.u - Q.u] .o. R).

  Translation into FSA: This operator was not translated since its precedence comparing to the other operators is not known. Moreover, it is rarely used.

- XFST syntax: $R. - l.Q$

  Meaning: Lower-side minus. Set of all string pairs from the relation R minus those that have a lower string that is also in Q. (Definition: R .-l. Q = R .o. [R.l - Q.l]).

  Translation into FSA: This operator was not translated since its precedence comparing to the other operators is not known. Moreover, it is rarely used.

- XFST syntax: $A < B$

  Meaning: A before B. Set of strings where no substring from B can precede any substring from A.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by $\sim \$[B\ A]$.

- XFST syntax: $A > B$

  Meaning: A after B. Set of strings where no substring from B can follow any substring from A.

  Translation into FSA: This operator does not exist in FSA but can be constructed using previous operators by $\sim \$[A\ B]$

- XFST syntax: $A <> B$

  Meaning: Shuffle of A and B. (e.g. [a b <> x y] contains the strings axyb, axby, abxy, xayb, xaby and xyab.)

  Translation into FSA: This operator was not translated into FSA since its meaning is not clear. Moreover, it is rarely used.

- XFST syntax: $A.r$

  Meaning: Reverses the regular language A (e.g. [a b c].r equals c b a).

  Translation into FSA: reverse(A)

- XFST syntax: $'[[A],\ s,\ sl]$

  Meaning: Substitution of all symbols s (being a single symbol or a constituent of a symbol pair) in the language or relation A by all symbols of the list sl. (e.g. '[[a b:0 b:x c], b, p q] gives [a [p:0 | q:0] [p:x | q:x] c].)

  Translation into FSA: This operator was not translated into FSA.

- XFST syntax: $R.u$ or $R.1$

  Meaning: Denotes the upper language of the regular relation R. (e.g. [A .x. B].1 equals A)

  Translation into FSA: domain(R)

- XFST syntax: $R.l$ or $R.2$

  Meaning: Denotes the lower language of the regular relation R. (e.g. [A .x. B].2 equals B)

  Translation into FSA: range(R)

- XFST syntax: $R.i$

  Meaning: Inverts the regular relation R. (e.g. [A .x. B].i equals [B .x. A])

  Translation into FSA: invert(R)

2.3 Restriction

Restriction rules were not translated into FSA since the documentation for their creating algorithms could not be obtained, nor we succeeded to find them ourselves. Therefore, we only bring their XFST syntax and meaning.

- XFST syntax: $A => L\_ R$

  Meaning: Restriction. Strings of language A are only allowed between a string of language L to the left and a string of language R to the right.

- XFST syntax: $A => L_1 \_ R_1, L_2 \_ R_2, \ldots, L_n \_ R_n$

  Meaning:Restriction. Strings of language A are only allowed between a string of language L_1 to the left and a string of language R_1 to the right or between a string of L_2 and a string of R_2 or ..... or between a string of L_n and a string of R_n.

Notice that restriction rules are not simple for implementation as they might seem at first, since the left and right context might have shared string instances with A, thus special care is required.

2.4 Replacement

Replace rules describe an operation that replaces some symbol or sequence of symbols by another sequence or symbol. Thus, a replace rule results with a relation (and not a language). There is no construct in the FSA Utils for replacement rules, thus, they had to be implemented using the simple operators defined above (or by replace rules that have already been defined). Therefore, in all this section, instead of referring for each operator to its "translation into FSA", we refer to "construction". The construction is done by using XFST operators (as we did in previous cases were the operator was not defined directly in FSA). The XFST replace rules are close relatives of the rewrite rules that were defined at Kaplan and Kay (1994), but they are not identical to it, therefore, we can not rely on Kaplan and Kay (1994) implementation. The construction of the XFST replace rules is based on the available documentation of their accurate implementation. Thus, the following implementations are based on Karttunen (1997), Karttunen (1995), Karttunen and Kempe (1995) and Karttunen (1996). The replace rules can be divided into four groups:

1. Unconditional replace rules (one rule with no context).
2. Unconditional parallel replace rules (several rules with no context that are performed at the same time).
3. Conditional replace rules (one rule and one condition).
4. Conditional parallel replace rules (several rules and/or several contexts).

We deal with most of the rules from the first three groups but with none of the conditional parallel replace rules. The reason for not implementing all of the replace operators is lack of documentation.

2.4.1 $->$ (obligatory, upper to lower replacement)

- XFST syntax: $A -> B$

  Meaning: Unconditional replacement of the language A by the language B. This denotes a relation that consists of pairs of strings that are identical except that every instance of A in the upper-side string corresponds to an instance of B in the lower-side string. For example, $a -> b$ pairs "b" with "b" (no change) and "aba" with "bbb" (replacing both "a"s by "b"s). The replacement always takes place without considering the context.

  Construction: [[NO_ A $[A\ .x.\ B]$]$*$ NO_ A] where NO_ A abbreviates $\sim \$[\ A - [\ ]\ ]$. This construction defines a regular relation were each member of this relation contains any number (which can be also zero) of iterations of $[A\ .x.\ B]$ combined with strings that do not contain A which are mapped to themselves. The strings not containing A (NO_ A) are defined by $\sim \$[\ A - [\ ]\ ]$ and not by $\sim \$A$ to deal with cases were A contains the empty string.

- XFST syntax: $A_1 -> B_1, \ldots, A_n -> B_n$

  Meaning: Unconditional parallel replacement of the language A_1 by the language B_1 and the language A_2 by the language B_2 ... and the language A_n by the language B_n. This is like the relation $A -> B$ except that there are n correspondences to consider simultaneously. For example, $a -> b, b -> a$ relates "abba" to "baab".

  Construction: [[N R] $*$ N] where N denotes the language of strings that do not contain any A_i:

  $$N = \sim \$[\ [A_1 \mid ... \mid A_n] - [\ ]\ ]$$

  and R stands for the relation that pairs every A_i with the corresponding B_i:

  $$R = [\ [A_1\ .x.B_1] \mid ... \mid [A_n\ .x.B_n]\ ]$$

  .

- XFST syntax: $A -> B \mid\mid L \_ R$

  Meaning: Conditional replacement of the language A by the language B. This is like the relation $A -> B$ except that any instance of A in the upper string corresponds to an instance of B in the lower string only when it is preceded by an instance of L and followed by an instance of R. Other instances of A in the upper string remain unchanged. A, B, L, and R must all denote simple languages, not relations. The slant of the double bars indicates whether the precede/follow constraints refer to the instance of A in the upper string or to its image in the lower string. In the $\mid\mid$ version, both contexts refer to the upper string.

  Construction:

  $$InsertBrackets$$
  $$.o.$$
  $$ConstrainBrackets$$
  $$.o.$$
  $$LeftContext$$
  $$.o.$$
  $$RightContext$$
  $$.o.$$
  $$Replace$$
  $$.o.$$
  $$RemoveBrackets$$

  Where:

  - Let $<$ and $>$ be two symbols not in Σ. The escape character $\%$ is used since $<$ and $>$ are saved symbols in XFST.
  - $InsertBrackets = [\ [\ ] < - \% < \mid \% >]$. InserBrackets eliminates from the upper side language all context markers that appear on the lower side.

- $ConstrainBrackets = [\sim \$[\%< \ \%>]\ ]$. ConstrainBrackets denotes the language consisting of strings that do not contain <> anywhere.
- $LeftContext = [\sim [\sim [...LEFT]\ \ [< ...]\ ]\ \&\ \sim [\ [...LEFT]\ \ \sim [< ...]\ ]\ ]$. LeftContext denotes the language in which any instance of $<$ is immediately preceded by LEFT and every LEFT is immediately followed by $<$, ignoring irrelevant brackets. $[...LEFT]$ denotes $[\ [\ ?* L/[\%< \ |\ \%>]\ ] - [?* \ \%<]\ ]$, the language of all strings ending in L, ignoring all brackets except for a final $<$. $[< ...]$ denotes $[\%< /\%> ?*]$, the language of strings beginning with $<$, ignoring the other bracket.
- $RightContext = [\sim [\ [... >]\ \ \sim [RIGHT...]\ ]\ \&\ \sim [\sim [... >]\ \ [RIGHT...]\ ]\ ]$. RightContext denotes the language in which any instance of $>$ is immediately followed by RIGHT and any RIGHT is immediately preceded by $>$, ignoring irrelevant brackets. $[RIGHT...]$ denotes $[\ [\ R/[\%< \ |\ \%>]\ \ ?*\ ] - [\%< ?*]\ ]$, the language of all strings beginning with R, ignoring all brackets except for an initial $>$. $[... >]$ denotes $[?* \ \%> /\%<]$, the language of strings ending with $>$, ignoring the other bracket.
- The definition of Replace divides into three cases:
  1. If A does not contain the empty string (epsilon) then
  $$Replace = [\%< \ \ A/[\%< \ |\ \%>]\ \%> \ -> \ \%< \ \ B/[\%< \ |\ \%>]\ \%>]$$
  This is the unconditional replacement of $< A >$ by $< B >$, ignoring irrelevant brackets.
  2. If A is the empty string (i.e., A=epsilon) then
  $$Replace = [\%> \ \%< \ -> \ \%< \ B* \ \%>]$$
  3. If A contains the empty string but is not equal to it (i.e., contains other strings too) then $Replace =$
  $$\%< \ \ A/[\%< \ |\ \%>]\ \%> \ -> \ \%< \ \ B/[\%< \ |\ \%>]\ \%>$$
  ,
  $$\%> \ \%< \ -> \ \%< \ B* \ \%>$$
  That is, the first two cases are performed in parallel.
- $RemoveBrackets = [\ \%< \ |\ \%> \ -> \ [\ ]\ ]$. RemoveBrackets denotes the relation that maps the strings of the upper language to the same strings without any context markers.

- XFST syntax: $A -> B \ // \ L \_ R$

  Meaning: Conditional replacement of the language A by the language B. This is like the relation $A -> B \ || \ L \_ R$ except that the // variant requires the left context on the lower side of the replacement and the right context on the upper side.

  Construction:
  $$InsertBrackets$$
  $$.o.$$
  $$ConstrainBrackets$$
  $$.o.$$

$$RightContext$$

$$.o.$$

$$Replace$$

$$.o.$$

$$LeftContext$$

$$.o.$$

$$RemoveBrackets$$

Where InsertBrackets, ConstrainBrackets, RightContext, Replace, LeftContext and RemoveBrackets are the same as above.

- XFST syntax: $A -> B \backslash\backslash L \_ R$

  Meaning: Conditional replacement of the language A by the language B. This is like the relation $A -> B \ || \ L \ \_ \ R$ except that the $\backslash\backslash$ variant requires the left context on the upper side of the replacement and the right context on the lower side.

  Construction:

$$InsertBrackets$$

$$.o.$$

$$ConstrainBrackets$$

$$.o.$$

$$LeftContext$$

$$.o.$$

$$Replace$$

$$.o.$$

$$RightContext$$

$$.o.$$

$$RemoveBrackets$$

  Where InsertBrackets, ConstrainBrackets, RightContext, Replace, LeftContext and RemoveBrackets are the same as above.

- XFST syntax: $A -> B \backslash/ L \_ R$

  Meaning: Conditional replacement of the language A by the language B. This is like the relation $A -> B \ || \ L \ \_ \ R$ except that in the $\backslash/$ variant, both contexts refer to the lower string.

  Construction:

$$InsertBrackets$$

$$.o.$$

$$ConstrainBrackets$$

$$.o.$$

$$Replace$$

$$.o.$$

$$LeftContext$$

$$.o.$$

$$RightContext$$

$$.o.$$

$$RemoveBrackets$$

Where InsertBrackets, ConstrainBrackets, RightContext, Replace, LeftContext and RemoveBrackets are the same as above.

The rest of the obligatory upper to lower replace rules are conditional parallel replace rules, thus, as mentioned they were not implemented. We only bring their syntax definition and meaning.

- XFST syntax: $A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \mid\mid L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$

  Meaning: Conditional parallel replacement. Every string of the language A_{11} (upper side) is replaced by all strings of the language B_{11} (lower side; multiple results possible) and ..... and every string of A_{1n} is replaced by all strings of B_{1n} if and only if it occurs between a string of L_{11} to the left and a string of R_{11} to the right or ..... or between a string of L_{1m} and a string of R_{1m}. All these replacements take place simultaneously without interfering with each other. All replacements are obligatory and upper-to-lower (since the replacement symbol is $->$). In the $\mid\mid$ variant both left and right contexts are on the upper side.

- XFST syntax: $A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \;//\; L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$

  Meaning: Conditional parallel replacement. This is like the relation

  $$A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \mid\mid L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$$

  except that the // variant requires the left contexts on the lower side of the replacement and the right contexts on the upper side.

- XFST syntax: $A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \setminus\setminus L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$

  Meaning: Conditional parallel replacement. This is like the relation

  $$A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \mid\mid L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$$

  except that the $\setminus\setminus$ variant requires the left contexts on the upper side of the replacement and the right contexts on the lower side.

- XFST syntax: $A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \setminus/ L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$

  Meaning: Conditional parallel replacement. This is like the relation

  $$A_{11} -> B_{11}, \ldots, A_{1n} -> B_{1n} \mid\mid L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$$

  except that in the $\setminus/$ variant both left and right contexts are on the lower side.

- XFST syntax:
  $A_{11} -> B_{11}\ ,\ \ldots\ ,\ A_{1n} -> B_{1n}\ ||\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21} -> B_{21}\ ,\ \ldots\ ,\ A_{2p} -> B_{2p}\ ||\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1} -> B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> B_{kr}\ ||\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

  Meaning: Conditional parallel replacement. Every string of the language A_{11} (upper side) is replaced by all strings of the language B_{11} (lower side; multiple results possible) and ..... and every string of A_{1n} is replaced by all strings of B_{1n} if and only if it occurs between a string of L_{11} to the left and a string of R_{11} to the right or ..... or between a string of L_{1m} and a string of R_{1m}. At the same time, every string of A_{21} is replaced by all strings of language B_{21}, etc., if it occurs between a string of L_{21} and a string of R_{21}, etc. All these replacements take place simultaneously without interfering with each other. Note the double comma separating the groups of related replacements. A single comma is used as the separator inside the replace and context lists. All replacements are obligatory and upper-to-lower (since the replacement symbol is $->$). In the $||$ variant both left and right contexts are on the upper side.

- XFST syntax:
  $A_{11} -> B_{11}\ ,\ \ldots\ ,\ A_{1n} -> B_{1n}\ //\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21} -> B_{21}\ ,\ \ldots\ ,\ A_{2p} -> B_{2p}\ //\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1} -> B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> B_{kr}\ //\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

  Meaning: Conditional parallel replacement. This is like the above relation except that the // variant requires the left contexts on the lower side of the replacement and the right contexts on the upper side.

- XFST syntax:
  $A_{11} -> B_{11}\ ,\ \ldots\ ,\ A_{1n} -> B_{1n}\ \backslash\backslash\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21} -> B_{21}\ ,\ \ldots\ ,\ A_{2p} -> B_{2p}\ \backslash\backslash\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1} -> B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> B_{kr}\ \backslash\backslash\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

  Meaning: Conditional parallel replacement. This is like the above relation except that the $\backslash\backslash$ variant requires the left contexts on the upper side of the replacement and the right contexts on the lower side.

- XFST syntax:
  $A_{11} -> B_{11}\ ,\ \ldots\ ,\ A_{1n} -> B_{1n}\ \backslash/\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21} -> B_{21}\ ,\ \ldots\ ,\ A_{2p} -> B_{2p}\ \backslash/\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1} -> B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> B_{kr}\ \backslash/\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

  Meaning: Conditional parallel replacement. This is like the above relation except that in the $\backslash/$ variant both the left contexts and the right contexts are on the lower side.

2.4.2 $(->)$ (optional, upper to lower replacement)

The following operations are the same as in section 2.4.1, except that instead of $->$ appears $(->)$. As $->$ represented an obligatory upper to lower replacement, $(->)$ represent an optional upper to lower

replacement, i.e., the meaning of the following operators is the same as with $->$ with the extra element of optionality. For example, $a -> b$ paired "b" with "b" (no change) and "aba" with "bbb" (replacing both "a"s by "b"s), but $a(->)b$ pairs "b" with "b" (no change) and "aba" with "bbb" (replacing both "a"s by "b"s), "bba" (replacing only the first "a" with a "b", leaving the second one unchanged), "abb" (replacing only the second "a" with a "b"', leaving the first one unchanged) and "aba" (leaving both of the "a"s unchanged). Note that $A(->)B$ is not equal to $A -> [B \mid A]$ nor to $[A -> B] \mid A$.

- XFST syntax: $A(->)B$

  Construction: [[?* [A .x. B]]* ?*]. This construction defines a regular relation were each member of this relation contains any number (which can be also zero) of iterations of [A .x. B] combined with ?* that can contain A too.

- XFST syntax: $A_1(->)B_1, \ldots, A_n(->)B_n$

  Construction: [[?* R]* ?*] where R stands for the relation that pairs every A_i with the corresponding B_i: $R =$ [[A_1 .x.B_1] | ... | [A_n .x.B_n]].

- XFST syntax: $A(->)B \mid\mid L \_ R$

  Construction: The same as the construction for $A -> B \mid\mid L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $->$ is replaced by the optional upper to lower replace operator $(->)$.

- XFST syntax: $A(->)B \,//\, L \_ R$

  Construction: The same as the construction for $A -> B \,//\, L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $->$ is replaced by the optional upper to lower replace operator $(->)$.

- XFST syntax: $A(->)B \setminus\setminus L \_ R$

  Construction: The same as the construction for $A -> B \setminus\setminus L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $->$ is replaced by the optional upper to lower replace operator $(->)$.

- XFST syntax: $A(->)B \setminus/ L \_ R$

  Construction: The same as the construction for $A -> B \setminus/ L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $->$ is replaced by the optional upper to lower replace operator $(->)$.

The rest of the optional upper to lower replace rules are conditional parallel replace rules, thus, as mentioned they were not implemented. We only bring their syntax definition. Their meaning was explained in the beginning of this section.

- XFST syntax: $A_{11}(->)B_{11}, \ldots, A_{1n}(->)B_{1n} \mid\mid L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11}(->)B_{11}, \ldots, A_{1n}(->)B_{1n} \,//\, L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11}(->)B_{11}, \ldots, A_{1n}(->)B_{1n} \setminus\setminus L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11}(->)B_{11}, \ldots, A_{1n}(->)B_{1n} \setminus/ L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$

- XFST syntax:
  $A_{11}(->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(->)B_{1n}\ ||\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21}(->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(->)B_{2p}\ ||\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1}(->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(->)B_{kr}\ ||\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

- XFST syntax:
  $A_{11}(->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(->)B_{1n}\ //\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21}(->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(->)B_{2p}\ //\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1}(->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(->)B_{kr}\ //\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

- XFST syntax:
  $A_{11}(->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(->)B_{1n}\ \backslash\backslash\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21}(->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(->)B_{2p}\ \backslash\backslash\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1}(->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(->)B_{kr}\ \backslash\backslash\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

- XFST syntax:
  $A_{11}(->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(->)B_{1n}\ \backslash/\ L_{11}\_\ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_\ R_{1m}$
  ,,
  $A_{21}(->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(->)B_{2p}\ \backslash/\ L_{21}\_\ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_\ R_{2q}$
  ,, ... ,,
  $A_{k1}(->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(->)B_{kr}\ \backslash/\ L_{k1}\_\ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_\ R_{ks}$

2.4.3 $<-$ (obligatory, lower to upper replacement)

The following operations are the same as in section 2.4.1, except that instead of $->$ appears $<-$. As $->$ represented an obligatory upper to lower replacement, $<-$ represent an obligatory lower to upper replacement. Thus, the meaning of the $<-$ replacement is that if a string of lower appears in the lower language, it must be paired with a string of upper in the upper language. Notice that a string from upper can be mapped either to itself or to a string from lower. For example, $a->b$ paired "b" with "b" (no change) and "a" with "b" (replacing "a" by "b"), but $a<-b$ does not pair "b" with anything (since the pair b:b is not legal) and pairs "a" with "a" and with "b".

- XFST syntax: $A<-B$

  Construction: $[B->A].i$

- XFST syntax: $A_1<-B_1, \ldots, A_n<-B_n$

  Construction: $[\ B_1->A_1, \ldots, B_n->A_n\ ].i$

- XFST syntax: $A<-B\ ||\ L\ \_\ R$

  Construction: The same as the construction for $A->B\ ||\ L\_\ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $->$ is replaced by the obligatory lower to upper replace operator $<-$.

- XFST syntax: $A < -B \;//\; L \_ R$

  Construction: The same as the construction for $A- > B \;//\; L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the obligatory lower to upper replace operator $< -$.

- XFST syntax: $A < -B \;\backslash\backslash\; L \_ R$

  Construction: The same as the construction for $A- > B \;\backslash\backslash\; L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the obligatory lower to upper replace operator $< -$.

- XFST syntax: $A < -B \;\backslash/\; L \_ R$

  Construction: The same as the construction for $A- > B \;\backslash/\; L \_ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the obligatory lower to upper replace operator $< -$.

The rest of the obligatory lower to upper replace rules are conditional parallel replace rules, thus, as mentioned they were not implemented. We only bring their syntax definition. Their meaning was explained in the beginning of this section.

- XFST syntax: $A_{11} < -B_{11}, \ldots, A_{1n} < -B_{1n} \;||\; L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} < -B_{11}, \ldots, A_{1n} < -B_{1n} \;//\; L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} < -B_{11}, \ldots, A_{1n} < -B_{1n} \;\backslash\backslash\; L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} < -B_{11}, \ldots, A_{1n} < -B_{1n} \;\backslash/\; L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax:
  $A_{11} < -B_{11}\ ,\ \ldots\ ,\ A_{1n} < -B_{1n} \;||\; L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} < -B_{21}\ ,\ \ldots\ ,\ A_{2p} < -B_{2p} \;||\; L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, ⋯ ,,
  $A_{k1} < -B_{k1}\ ,\ \ldots\ ,\ A_{kr} < -B_{kr} \;||\; L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} < -B_{11}\ ,\ \ldots\ ,\ A_{1n} < -B_{1n} \;//\; L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} < -B_{21}\ ,\ \ldots\ ,\ A_{2p} < -B_{2p} \;//\; L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, ⋯ ,,
  $A_{k1} < -B_{k1}\ ,\ \ldots\ ,\ A_{kr} < -B_{kr} \;//\; L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} < -B_{11}\ ,\ \ldots\ ,\ A_{1n} < -B_{1n} \;\backslash\backslash\; L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} < -B_{21}\ ,\ \ldots\ ,\ A_{2p} < -B_{2p} \;\backslash\backslash\; L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, ⋯ ,,
  $A_{k1} < -B_{k1}\ ,\ \ldots\ ,\ A_{kr} < -B_{kr} \;\backslash\backslash\; L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$

- XFST syntax:
  $A_{11} < -B_{11}$, ... , $A_{1n} < -B_{1n}$ \/ $L_{11}\_ R_{11}$, ... , $L_{1m}\_ R_{1m}$
  , ,
  $A_{21} < -B_{21}$, ... , $A_{2p} < -B_{2p}$ \/ $L_{21}\_ R_{21}$, ... , $L_{2q}\_ R_{2q}$
  , , ... , ,
  $A_{k1} < -B_{k1}$, ... , $A_{kr} < -B_{kr}$ \/ $L_{k1}\_ R_{k1}$, ... , $L_{ks}\_ R_{ks}$

2.4.4 $(< -)$ (optional, lower to upper replacement)

The following operations are the same as in section 2.4.3, except that instead of $< -$ appears $(< -)$. As $< -$ represented an obligatory lower to upper replacement, $(< -)$ represents optional lower to upper replacement. Thus, the meaning of the $(< -)$ replacement is the same as of $< -$ replacement with the extra element of optionality. For example, $a < -b$ does not pair "b" with anything (since the pair b:b is not legal) and pairs "a" with "a" and with "b", but $a(< -)b$ pairs "b" with "b" (the pair b:b is legal now since there is optionality) and pairs "a" with "a" and with "b".

- XFST syntax: $A(< -)B$
  Construction: $[B(- >)A].i$
- XFST syntax: $A_1(< -)B_1, \ldots, A_n(< -)B_n$
  Construction: $[\ B_1(- >)A_1, \ldots, B_n(- >)A_n\ ].i$
- XFST syntax: $A(< -)B \mid\mid L\ \_\ R$
  Construction: The same as the construction for $A- > B \mid\mid L\ \_\ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the optional lower to upper replace operator $(< -)$.
- XFST syntax: $A(< -)B \,//\, L\ \_\ R$
  Construction: The same as the construction for $A- > B \,//\, L\ \_\ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the optional lower to upper replace operator $(< -)$.
- XFST syntax: $A(< -)B \setminus\setminus L\ \_\ R$
  Construction: The same as the construction for $A- > B \setminus\setminus L\ \_\ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the optional lower to upper replace operator $(< -)$.
- XFST syntax: $A(< -)B \setminus/ L\ \_\ R$
  Construction: The same as the construction for $A- > B \setminus/ L\ \_\ R$ except for the $Replace$ stage where for each one of the three cases the obligatory upper to lower replace operator $- >$ is replaced by the optional lower to upper replace operator $(< -)$.

The rest of the optional lower to upper replace rules are conditional parallel replace rules, thus, as mentioned they were not implemented. We only bring their syntax definition. Their meaning was explained in the beginning of this section.

- XFST syntax: $A_{11}(< -)B_{11}, \ldots, A_{1n}(< -)B_{1n} \mid\mid L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}(< -)B_{11}, \ldots, A_{1n}(< -)B_{1n} \,//\, L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$

- XFST syntax: $A_{11}(<-)B_{11}, \ldots, A_{1n}(<-)B_{1n}$ \\ $L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$
- XFST syntax: $A_{11}(<-)B_{11}, \ldots, A_{1n}(<-)B_{1n}$ \/ $L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$
- XFST syntax:
  $A_{11}(<-)B_{11}$, ... , $A_{1n}(<-)B_{1n}$ || $L_{11}\_R_{11}$, ... , $L_{1m}\_R_{1m}$
  ,,
  $A_{21}(<-)B_{21}$, ... , $A_{2p}(<-)B_{2p}$ || $L_{21}\_R_{21}$, ... , $L_{2q}\_R_{2q}$
  ,, ... ,,
  $A_{k1}(<-)B_{k1}$, ... , $A_{kr}(<-)B_{kr}$ || $L_{k1}\_R_{k1}$, ... , $L_{ks}\_R_{ks}$
- XFST syntax:
  $A_{11}(<-)B_{11}$, ... , $A_{1n}(<-)B_{1n}$ // $L_{11}\_R_{11}$, ... , $L_{1m}\_R_{1m}$
  ,,
  $A_{21}(<-)B_{21}$, ... , $A_{2p}(<-)B_{2p}$ // $L_{21}\_R_{21}$, ... , $L_{2q}\_R_{2q}$
  ,, ... ,,
  $A_{k1}(<-)B_{k1}$, ... , $A_{kr}(<-)B_{kr}$ // $L_{k1}\_R_{k1}$, ... , $L_{ks}\_R_{ks}$
- XFST syntax:
  $A_{11}(<-)B_{11}$, ... , $A_{1n}(<-)B_{1n}$ \\ $L_{11}\_R_{11}$, ... , $L_{1m}\_R_{1m}$
  ,,
  $A_{21}(<-)B_{21}$, ... , $A_{2p}(<-)B_{2p}$ \\ $L_{21}\_R_{21}$, ... , $L_{2q}\_R_{2q}$
  ,, ... ,,
  $A_{k1}(<-)B_{k1}$, ... , $A_{kr}(<-)B_{kr}$ \\ $L_{k1}\_R_{k1}$, ... , $L_{ks}\_R_{ks}$
- XFST syntax:
  $A_{11}(<-)B_{11}$, ... , $A_{1n}(<-)B_{1n}$ \/ $L_{11}\_R_{11}$, ... , $L_{1m}\_R_{1m}$
  ,,
  $A_{21}(<-)B_{21}$, ... , $A_{2p}(<-)B_{2p}$ \/ $L_{21}\_R_{21}$, ... , $L_{2q}\_R_{2q}$
  ,, ... ,,
  $A_{k1}(<-)B_{k1}$, ... , $A_{kr}(<-)B_{kr}$ \/ $L_{k1}\_R_{k1}$, ... , $L_{ks}\_R_{ks}$

2.4.5 $<->$ (obligatory, upper to lower, lower to upper replacement)

The following operations are the same as in section 2.4.1, except that instead of $->$ appears $<->$. As $->$ represented an obligatory upper to lower replacement, $<->$ represents an obligatory upper to lower and lower to upper replacement. Thus, the meaning of the $<->$ replacement is that if a string of upper appears in the upper language, it must be paired with a string of lower in the lower language and if a string of lower appears in the lower language, it must be paired with a string of upper in the upper language. In the $<->$ replacement the replacement is done in both ways ($->$ and $<-$). For example, $a -> b$ paired "b" with "b" (no change) and "a" with "b" (replacing "a" by "b"), $a <- b$ paired "b" with nothing and "a" with "a" and "b", but $a <-> b$ does not pair "b" with anything (since the pair b:b is not legal) and pairs "a" only with "b".

- XFST syntax: $A <-> B$

  Construction: Let @ be a character not in Σ. We use the escape character % to precede @, since @ is a reserved character in XFST. Thus, $A <-> B$ is defined as

  $$\sim \$[\%@]$$

$$.o.$$

$$A -> \ \%@$$

$$.o.$$

$$\%@ \ <- B$$

$$.o.$$

$$\sim \$[\%@]$$

- XFST syntax: $A_1 <- > B_1, \ldots, A_n <- > B_n$

  Construction: Let $@1, \ldots, @n$ be characters not in Σ. We use the escape character $\%$ to precede each $@i$, since @ is a reserved character in XFST. Thus, $A <- > B$ is defined as

$$\sim \$[\%@1 \mid \ \ldots \ \mid \%@n]$$

$$.o.$$

$$A_1 -> \ \%@1 \ , \ \ldots \ , \ A_n -> \ \%@n$$

$$.o.$$

$$\%@1 \ <- B_1 \ , \ \ldots \ , \ \%@n \ <- B_n$$

$$.o.$$

$$\sim \$[\%@1 \mid \ \ldots \ \mid \%@n]$$

The rest of the obligatory upper to lower and lower to upper replace rules are conditional replace rules and they were not implemented. We only bring their syntax definition. Their meaning was explained in the beginning of this section.

- XFST syntax: $A <- > B \ || \ L \ \_ \ R$
- XFST syntax: $A <- > B \ // \ L \ \_ \ R$
- XFST syntax: $A <- > B \ \backslash\backslash \ L \ \_ \ R$
- XFST syntax: $A <- > B \ \backslash / \ L \ \_ \ R$
- XFST syntax: $A_{11} <- > B_{11}, \ldots, A_{1n} <- > B_{1n} \ || \ L_{11} \_ \ R_{11}, \ldots, L_{1m} \_ \ R_{1m}$
- XFST syntax: $A_{11} <- > B_{11}, \ldots, A_{1n} <- > B_{1n} \ // \ L_{11} \_ \ R_{11}, \ldots, L_{1m} \_ \ R_{1m}$
- XFST syntax: $A_{11} <- > B_{11}, \ldots, A_{1n} <- > B_{1n} \ \backslash\backslash \ L_{11} \_ \ R_{11}, \ldots, L_{1m} \_ \ R_{1m}$
- XFST syntax: $A_{11} <- > B_{11}, \ldots, A_{1n} <- > B_{1n} \ \backslash / \ L_{11} \_ \ R_{11}, \ldots, L_{1m} \_ \ R_{1m}$
- XFST syntax:
  $A_{11} <- > B_{11} \ , \ \ldots \ , \ A_{1n} <- > B_{1n} \ || \ L_{11} \_ \ R_{11} \ , \ \ldots \ , \ L_{1m} \_ \ R_{1m}$
  , ,
  $A_{21} <- > B_{21} \ , \ \ldots \ , \ A_{2p} <- > B_{2p} \ || \ L_{21} \_ \ R_{21} \ , \ \ldots \ , \ L_{2q} \_ \ R_{2q}$
  , , $\cdots$, ,
  $A_{k1} <- > B_{k1} \ , \ \ldots \ , \ A_{kr} <- > B_{kr} \ || \ L_{k1} \_ \ R_{k1} \ , \ \ldots \ , \ L_{ks} \_ \ R_{ks}$

- XFST syntax:
  $A_{11} <- > B_{11}\ ,\ \ldots\ ,\ A_{1n} <- > B_{1n}$ // $L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21} <- > B_{21}\ ,\ \ldots\ ,\ A_{2p} <- > B_{2p}$ // $L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, ... ,,
  $A_{k1} <- > B_{k1}\ ,\ \ldots\ ,\ A_{kr} <- > B_{kr}$ // $L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$

- XFST syntax:
  $A_{11} <- > B_{11}\ ,\ \ldots\ ,\ A_{1n} <- > B_{1n}$ \\ $L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21} <- > B_{21}\ ,\ \ldots\ ,\ A_{2p} <- > B_{2p}$ \\ $L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, ... ,,
  $A_{k1} <- > B_{k1}\ ,\ \ldots\ ,\ A_{kr} <- > B_{kr}$ \\ $L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$

- XFST syntax:
  $A_{11} <- > B_{11}\ ,\ \ldots\ ,\ A_{1n} <- > B_{1n}$ \/ $L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21} <- > B_{21}\ ,\ \ldots\ ,\ A_{2p} <- > B_{2p}$ \/ $L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, ... ,,
  $A_{k1} <- > B_{k1}\ ,\ \ldots\ ,\ A_{kr} <- > B_{kr}$ \/ $L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$

2.4.6 $(<->)$ (optional, upper to lower, lower to upper replacement)

The following operations are the same as in section 2.4.5, except that instead of $<->$ appears $(<->)$. As $<->$ represented an obligatory upper to lower and lower to upper replacement, $(<->)$ represents an optional upper to lower and lower to upper replacement. Thus, the meaning of the $(<->)$ replacement is the same as of $<->$ with the extra element of optionality. For example $a <- > b$ does not pair "b" with anything (since the pair b:b is not legal) and pairs "a" only with "b", but $a(<->)b$ pairs "b" with "b" and "a" with "a" and with "b".

- XFST syntax: $A(<->)B$

  Construction: Let @ be a character not in Σ. We use the escape character % to precede @, since @ is a reserved character in XFST. Thus, $A(<->)B$ is defined as

$$\sim \$[\%@]$$
$$.o.$$
$$A\ (->)\ \%@$$
$$.o.$$
$$\%@\ (<-)\ B$$
$$.o.$$
$$\sim \$[\%@]$$

The rest of the optional upper to lower and lower to upper replace rules are conditional and parallel replace rules and they were not implemented. We only bring their syntax definition. Their meaning was explained in the beginning of this section.

- XFST syntax: $A_1(<->)B_1, \ldots, A_n(<->)B_n$
- XFST syntax: $A(<->)B \mid\mid L \_ R$
- XFST syntax: $A(<->)B \mathbin{//} L \_ R$
- XFST syntax: $A(<->)B \setminus\setminus L \_ R$
- XFST syntax: $A(<->)B \setminus/ L \_ R$
- XFST syntax: $A_{11}(<->)B_{11}, \ldots, A_{1n}(<->)B_{1n} \mid\mid L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}(<->)B_{11}, \ldots, A_{1n}(<->)B_{1n} \mathbin{//} L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}(<->)B_{11}, \ldots, A_{1n}(<->)B_{1n} \setminus\setminus L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}(<->)B_{11}, \ldots, A_{1n}(<->)B_{1n} \setminus/ L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax:
  $A_{11}(<->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(<->)B_{1n} \mid\mid L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}(<->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(<->)B_{2p} \mid\mid L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}(<->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(<->)B_{kr} \mid\mid L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$
- XFST syntax:
  $A_{11}(<->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(<->)B_{1n} \mathbin{//} L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}(<->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(<->)B_{2p} \mathbin{//} L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}(<->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(<->)B_{kr} \mathbin{//} L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$
- XFST syntax:
  $A_{11}(<->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(<->)B_{1n} \setminus\setminus L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}(<->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(<->)B_{2p} \setminus\setminus L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}(<->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(<->)B_{kr} \setminus\setminus L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$
- XFST syntax:
  $A_{11}(<->)B_{11}\ ,\ \ldots\ ,\ A_{1n}(<->)B_{1n} \setminus/ L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}(<->)B_{21}\ ,\ \ldots\ ,\ A_{2p}(<->)B_{2p} \setminus/ L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}(<->)B_{k1}\ ,\ \ldots\ ,\ A_{kr}(<->)B_{kr} \setminus/ L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$

2.4.7 @-> (obligatory, upper to lower, left to right, longest match replacement)

The following operations are the same as in section 2.4.1, except that instead of $->$ appears @$->$. As $->$ represented an obligatory upper to lower replacement, @$->$ represents an obligatory upper to lower left to right longest match replacement. Instances of the language A on the upper side of the relation are replaced selectively. The selection factors each upper language string uniquely into A and non-A substrings.

The factoring is based on a left-to-right and longest match regimen. The starting locations are selected from left-to-right. If there are overlapping instances of A starting at the same location, only the longest one is replaced by all strings of B. For example, $a + -> b$ pairs "aaa" with "bbb", "bb" and "b", but $a@-> b$ pairs "aaa" only with "b".

- XFST syntax: $A@-> B$

  Construction:

$$InitialMatch$$
$$.o.$$
$$LeftToRight$$
$$.o.$$
$$LongestMatch$$
$$.o.$$
$$Replacement$$

  Where:

  - Let $\hat{}, <, >$ be characters not in Σ. We use the escape character $\%$ to precede them since they are reserved characters in XFST.
  - $InitialMatch =$

$$\sim \$[\ \%\hat{}\ |\ \%<\ |\ \%>\ ]$$
$$.o.$$
$$[\ .\ .]\ ->\ \%\hat{}\ ||\ \_\ A$$

    where $[\ .\ .]\ ->\ LOWER\ ||\ LEFT\_\ RIGHT$ is a version of empty string replacement that allows only one application between any LEFT and RIGHT. The construction for $[\ .\ .] -> LOWER\ ||\ LEFT\_\ RIGHT$ is the same as for $UPPER -> LOWER\ ||\ LEFT\_\ RIGHT$ except that $Replace = [\%>\ \%<\ ->\ \%<\ LOWER\ \%>]$.
  - $LeftToRight =$

$$[\ \sim \$[\%\hat{}]\ [\%\hat{}:\%<\ \ UPPER'\ \ 0:\%>\ ]\ ]*\ \sim \$[\%\hat{}]$$
$$.o.$$
$$\%\hat{}\ ->\ [\ ]$$

    where $UPPER' = [\ A/[\%\hat{}] - [?*\ \%\hat{}]\ ]$
  - $LongestMatch = \sim \$[\ \%<\ [\ UPPER''\ \&\ \$[\%>]\ ]\ ]$ where

$$UPPER'' =\ A/[\%<\ |\ \%>]\ -\ [\ ?*\ [\ \%<\ |\ \%>]\ ]$$

  - $Replacement =\ \%<\sim \$[\%>]\ \%>\ ->\ B$

The rest of the obligatory upper to lower left to right longest match replace rules are conditional and parallel replace rules and they were not implemented. We only bring their syntax definition. Their meaning was explained in the beginning of this section.

- XFST syntax: $A_1 @-> B_1, \ldots, A_n @-> B_n$
- XFST syntax: $A @-> B \,||\, L \_ R$
- XFST syntax: $A @-> B \,//\, L \_ R$
- XFST syntax: $A @-> B \,\backslash\backslash\, L \_ R$
- XFST syntax: $A @-> B \,\backslash/\, L \_ R$
- XFST syntax: $A_{11} @-> B_{11}, \ldots, A_{1n} @-> B_{1n} \,||\, L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} @-> B_{11}, \ldots, A_{1n} @-> B_{1n} \,//\, L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} @-> B_{11}, \ldots, A_{1n} @-> B_{1n} \,\backslash\backslash\, L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} @-> B_{11}, \ldots, A_{1n} @-> B_{1n} \,\backslash/\, L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax:
  $A_{11} @-> B_{11}\ ,\ \ldots\ ,\ A_{1n} @-> B_{1n} \,||\, L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} @-> B_{21}\ ,\ \ldots\ ,\ A_{2p} @-> B_{2p} \,||\, L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} @-> B_{k1}\ ,\ \ldots\ ,\ A_{kr} @-> B_{kr} \,||\, L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} @-> B_{11}\ ,\ \ldots\ ,\ A_{1n} @-> B_{1n} \,//\, L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} @-> B_{21}\ ,\ \ldots\ ,\ A_{2p} @-> B_{2p} \,//\, L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} @-> B_{k1}\ ,\ \ldots\ ,\ A_{kr} @-> B_{kr} \,//\, L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} @-> B_{11}\ ,\ \ldots\ ,\ A_{1n} @-> B_{1n} \,\backslash\backslash\, L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} @-> B_{21}\ ,\ \ldots\ ,\ A_{2p} @-> B_{2p} \,\backslash\backslash\, L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} @-> B_{k1}\ ,\ \ldots\ ,\ A_{kr} @-> B_{kr} \,\backslash\backslash\, L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} @-> B_{11}\ ,\ \ldots\ ,\ A_{1n} @-> B_{1n} \,\backslash/\, L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} @-> B_{21}\ ,\ \ldots\ ,\ A_{2p} @-> B_{2p} \,\backslash/\, L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} @-> B_{k1}\ ,\ \ldots\ ,\ A_{kr} @-> B_{kr} \,\backslash/\, L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$

2.4.8 @> (obligatory, upper to lower, left to right, shortest match replacement)

The following operations are the same as in section 2.4.7, except that instead of $@->$ appears $@>$. As $@->$ represented an obligatory upper to lower left to right longest match replacement, $@>$ represents an obligatory upper to lower left to right shortest match replacement, i.e., the only difference is that the replacement is done on shortest match basis instead of longest match basis. For example, $a + @-> b$ pairs

"aaa" with "b", but $a + @ > b$ pairs "aaa" with "bbb". These rules were not implemented. We only bring their XFST syntax.

- XFST syntax: $A@ > B$
- XFST syntax: $A_1@ > B_1, \ldots, A_n@ > B_n$
- XFST syntax: $A@ > B \mid\mid L \_ R$
- XFST syntax: $A@ > B \,//\, L \_ R$
- XFST syntax: $A@ > B \setminus\setminus L \_ R$
- XFST syntax: $A@ > B \setminus/ L \_ R$
- XFST syntax: $A_{11}@ > B_{11}, \ldots, A_{1n}@ > B_{1n} \mid\mid L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}@ > B_{11}, \ldots, A_{1n}@ > B_{1n} \,//\, L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}@ > B_{11}, \ldots, A_{1n}@ > B_{1n} \setminus\setminus L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax: $A_{11}@ > B_{11}, \ldots, A_{1n}@ > B_{1n} \setminus/ L_{11}\_ R_{11}, \ldots, L_{1m}\_ R_{1m}$
- XFST syntax:
  $A_{11}@ > B_{11}\ ,\ \ldots\ ,\ A_{1n}@ > B_{1n} \mid\mid L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}@ > B_{21}\ ,\ \ldots\ ,\ A_{2p}@ > B_{2p} \mid\mid L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}@ > B_{k1}\ ,\ \ldots\ ,\ A_{kr}@ > B_{kr} \mid\mid L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$
- XFST syntax:
  $A_{11}@ > B_{11}\ ,\ \ldots\ ,\ A_{1n}@ > B_{1n} \,//\, L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}@ > B_{21}\ ,\ \ldots\ ,\ A_{2p}@ > B_{2p} \,//\, L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}@ > B_{k1}\ ,\ \ldots\ ,\ A_{kr}@ > B_{kr} \,//\, L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$
- XFST syntax:
  $A_{11}@ > B_{11}\ ,\ \ldots\ ,\ A_{1n}@ > B_{1n} \setminus\setminus L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}@ > B_{21}\ ,\ \ldots\ ,\ A_{2p}@ > B_{2p} \setminus\setminus L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}@ > B_{k1}\ ,\ \ldots\ ,\ A_{kr}@ > B_{kr} \setminus\setminus L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$
- XFST syntax:
  $A_{11}@ > B_{11}\ ,\ \ldots\ ,\ A_{1n}@ > B_{1n} \setminus/ L_{11}\_ R_{11}\ ,\ \ldots\ ,\ L_{1m}\_ R_{1m}$
  ,,
  $A_{21}@ > B_{21}\ ,\ \ldots\ ,\ A_{2p}@ > B_{2p} \setminus/ L_{21}\_ R_{21}\ ,\ \ldots\ ,\ L_{2q}\_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1}@ > B_{k1}\ ,\ \ldots\ ,\ A_{kr}@ > B_{kr} \setminus/ L_{k1}\_ R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_ R_{ks}$

2.4.9 $->@$ (obligatory, upper to lower, right to left, longest match replacement)

The following operations are the same as in section 2.4.7, except that instead of $@->$ appears $->@$. As $@->$ represented an obligatory upper to lower left to right longest match replacement, $->@$ represents an obligatory upper to lower right to left longest match replacement, i.e., the only difference is that the replacement is done from right to left instead of from left to right. For example, $[\ [a\ b] \mid [b\ a]\ ] -> c$ pairs "aba" with "ac" and "ca", $[\ [a\ b] \mid [b\ a]\ ] @-> c$ pairs "aba" only with "ca"', but $[\ [a\ b] \mid [b\ a]\ ] -> @c$ pairs "aba" only with "ac". These rules were not implemented. We only bring their XFST syntax.

- XFST syntax: $A -> @B$
- XFST syntax: $A_1 -> @B_1, \ldots, A_n -> @B_n$
- XFST syntax: $A -> @B \ || \ L \_ R$
- XFST syntax: $A -> @B \ // \ L \_ R$
- XFST syntax: $A -> @B \ \backslash\backslash \ L \_ R$
- XFST syntax: $A -> @B \ \backslash / \ L \_ R$
- XFST syntax: $A_{11} -> @B_{11}, \ldots, A_{1n} -> @B_{1n} \ || \ L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} -> @B_{11}, \ldots, A_{1n} -> @B_{1n} \ // \ L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} -> @B_{11}, \ldots, A_{1n} -> @B_{1n} \ \backslash\backslash \ L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax: $A_{11} -> @B_{11}, \ldots, A_{1n} -> @B_{1n} \ \backslash / \ L_{11} \_ R_{11}, \ldots, L_{1m} \_ R_{1m}$
- XFST syntax:
  $A_{11} -> @B_{11}\ ,\ \ldots\ ,\ A_{1n} -> @B_{1n} \ || \ L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} -> @B_{21}\ ,\ \ldots\ ,\ A_{2p} -> @B_{2p} \ || \ L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} -> @B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> @B_{kr} \ || \ L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} -> @B_{11}\ ,\ \ldots\ ,\ A_{1n} -> @B_{1n} \ // \ L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} -> @B_{21}\ ,\ \ldots\ ,\ A_{2p} -> @B_{2p} \ // \ L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} -> @B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> @B_{kr} \ // \ L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} -> @B_{11}\ ,\ \ldots\ ,\ A_{1n} -> @B_{1n} \ \backslash\backslash \ L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,
  $A_{21} -> @B_{21}\ ,\ \ldots\ ,\ A_{2p} -> @B_{2p} \ \backslash\backslash \ L_{21} \_ R_{21}\ ,\ \ldots\ ,\ L_{2q} \_ R_{2q}$
  ,, $\cdots$,,
  $A_{k1} -> @B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> @B_{kr} \ \backslash\backslash \ L_{k1} \_ R_{k1}\ ,\ \ldots\ ,\ L_{ks} \_ R_{ks}$
- XFST syntax:
  $A_{11} -> @B_{11}\ ,\ \ldots\ ,\ A_{1n} -> @B_{1n} \ \backslash / \ L_{11} \_ R_{11}\ ,\ \ldots\ ,\ L_{1m} \_ R_{1m}$
  ,,

$A_{21} -> @B_{21}\ ,\ \ldots\ ,\ A_{2p} -> @B_{2p} \setminus\!/ L_{21}\_R_{21}\ ,\ \ldots\ ,\ L_{2q}\_R_{2q}$
$,,\ \cdots\ ,,$
$A_{k1} -> @B_{k1}\ ,\ \ldots\ ,\ A_{kr} -> @B_{kr} \setminus\!/ L_{k1}\_R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_R_{ks}$

2.4.10 $>$ @ right (obligatory, upper to lower, right to left, shortest match replacement)

The following operations are the same as in section 2.4.7, except that instead of $@->$ appears $>$ @. As $@->$ represented an obligatory upper to lower left to right longest match replacement, $>$ @ represents an obligatory upper to lower right to left shortest match replacement, i.e., the difference is that the replacement is done from right to left on shortest match basis instead of from left to right on longest match basis. These rules were not implemented. We only bring their XFST syntax.

- XFST syntax: $A > @B$
- XFST syntax: $A_1 > @B_1, \ldots, A_n > @B_n$
- XFST syntax: $A > @B \mid\mid L\_R$
- XFST syntax: $A > @B \,//\, L\_R$
- XFST syntax: $A > @B \setminus\!\setminus L\_R$
- XFST syntax: $A > @B \setminus\!/ L\_R$
- XFST syntax: $A_{11} > @B_{11}, \ldots, A_{1n} > @B_{1n} \mid\mid L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$
- XFST syntax: $A_{11} > @B_{11}, \ldots, A_{1n} > @B_{1n} \,//\, L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$
- XFST syntax: $A_{11} > @B_{11}, \ldots, A_{1n} > @B_{1n} \setminus\!\setminus L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$
- XFST syntax: $A_{11} > @B_{11}, \ldots, A_{1n} > @B_{1n} \setminus\!/ L_{11}\_R_{11}, \ldots, L_{1m}\_R_{1m}$
- XFST syntax:
  $A_{11} > @B_{11}\ ,\ \ldots\ ,\ A_{1n} > @B_{1n} \mid\mid L_{11}\_R_{11}\ ,\ \ldots\ ,\ L_{1m}\_R_{1m}$
  $,,$
  $A_{21} > @B_{21}\ ,\ \ldots\ ,\ A_{2p} > @B_{2p} \mid\mid L_{21}\_R_{21}\ ,\ \ldots\ ,\ L_{2q}\_R_{2q}$
  $,,\ \cdots\ ,,$
  $A_{k1} > @B_{k1}\ ,\ \ldots\ ,\ A_{kr} > @B_{kr} \mid\mid L_{k1}\_R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_R_{ks}$
- XFST syntax:
  $A_{11} > @B_{11}\ ,\ \ldots\ ,\ A_{1n} > @B_{1n} \,//\, L_{11}\_R_{11}\ ,\ \ldots\ ,\ L_{1m}\_R_{1m}$
  $,,$
  $A_{21} > @B_{21}\ ,\ \ldots\ ,\ A_{2p} > @B_{2p} \,//\, L_{21}\_R_{21}\ ,\ \ldots\ ,\ L_{2q}\_R_{2q}$
  $,,\ \cdots\ ,,$
  $A_{k1} > @B_{k1}\ ,\ \ldots\ ,\ A_{kr} > @B_{kr} \,//\, L_{k1}\_R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_R_{ks}$
- XFST syntax:
  $A_{11} > @B_{11}\ ,\ \ldots\ ,\ A_{1n} > @B_{1n} \setminus\!\setminus L_{11}\_R_{11}\ ,\ \ldots\ ,\ L_{1m}\_R_{1m}$
  $,,$
  $A_{21} > @B_{21}\ ,\ \ldots\ ,\ A_{2p} > @B_{2p} \setminus\!\setminus L_{21}\_R_{21}\ ,\ \ldots\ ,\ L_{2q}\_R_{2q}$
  $,,\ \cdots\ ,,$
  $A_{k1} > @B_{k1}\ ,\ \ldots\ ,\ A_{kr} > @B_{kr} \setminus\!\setminus L_{k1}\_R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_R_{ks}$

- XFST syntax:
  $A_{11} > @B_{11}\ ,\ \ldots\ ,\ A_{1n} > @B_{1n} \setminus/ L_{11}\_R_{11}\ ,\ \ldots\ ,\ L_{1m}\_R_{1m}$
  ,,
  $A_{21} > @B_{21}\ ,\ \ldots\ ,\ A_{2p} > @B_{2p} \setminus/ L_{21}\_R_{21}\ ,\ \ldots\ ,\ L_{2q}\_R_{2q}$
  ,, ... ,,
  $A_{k1} > @B_{k1}\ ,\ \ldots\ ,\ A_{kr} > @B_{kr} \setminus/ L_{k1}\_R_{k1}\ ,\ \ldots\ ,\ L_{ks}\_R_{ks}$

2.5 Markup

Markup rules take an input string and mark it by inserting some strings before and after it. There is no construct in the FSA Utils for markup rules, thus, they had to be implemented using the operators that have already been defined. Therefore, in all this section, instead of referring for each operator to its "translation into FSA", we refer to "construction". The construction is done by using XFST operators (as we did in previous cases were the operator was not defined directly in FSA). The construction of the XFST markup rules is based on the available documentation of their accurate implementation. Thus, the following implementations are based on Karttunen (1996).

- XFST syntax: $A -> L \ldots R$

  Meaning: Markup. Instances of the language A on the upper side of the relation are selected for markup. Each selected A string is marked by inserting all strings of L to its left and all strings of R to its right. The selected A strings themselves remain unchanged, along with the non-A segments.

  Construction: $A -> L\ A\ R$

- XFST syntax: $A @ -> L \ldots R$

  Meaning: Directed markup. Instances of the language A on the upper side of the relation are selected for markup under left-to-right, longest match regimen. Thus, The starting locations are selected from left-to-right. If there are overlapping instances of A starting at the same location, only the longest one is replaced by all strings of B. Each selected A string is marked by inserting all strings of R to its left and all strings of S to its right. The selected A strings themselves remain unchanged, along with the non-A segments.

  Construction:

  $$InitialMatch$$
  $$.o.$$
  $$LeftToRight$$
  $$.o.$$
  $$LongestMatch$$
  $$.o.$$
  $$Insrtion$$

  Where:

  - Let $\hat{}, <, >$ be characters not in Σ. We use the escape character $\%$ to precede them since they are reserved characters in XFST.

– $InitialMatch =$

$$\sim \$[\ \%\hat{}\ |\ \%<\ |\ \%>\ ]$$

$$.o.$$

$$[.\ .] -> \ \%\hat{}\ ||\ \_ A$$

where $[.\ .] -> \ \ LOWER\ ||\ LEFT\_\ RIGHT$ is a version of empty string replacement that allows only one application between any LEFT and RIGHT. The construction for $[.\ .] -> LOWER\ ||\ LEFT\_ RIGHT$ is the same as for $UPPER -> \ LOWER\ ||\ LEFT\_ RIGHT$ except that $Replace = [\%> \ \%< \ -> \ \%< \ LOWER\ \%>]$.

– $LeftToRight =$

$$[\ \sim \$[\%\hat{}]\ [\%\hat{}:\%<\ \ UPPER'\ \ 0:\%>\ ]\ ]*\ \sim \$[\%\hat{}]$$

$$.o.$$

$$\%\hat{}\ -> \ [\ ]$$

where $UPPER' = [\ A/[\%\hat{}] - [?*\ \%\hat{}]\ ]$

– $LongestMatch = \sim \$[\ \%<\ [\ UPPER''\ \&\ \$[\%>]\ ]\ ]$ where

$$UPPER'' = \ A/[\%<\ |\ \%>] \ - \ [\ ?*\ [\ \%<\ |\ \%>]\ ]$$

– $Insrtion = \ \%<\ -> \ L\ ,\ \%>\ -> \ R$

The rest of the markup rules were not implemented since we could not obtain any documentation of the creating algorithms. We bring only their XFST syntax and meaning.

- XFST syntax: $A@>L \ldots R$

  Meaning: Directed markup. Instances of the language A on the upper side of the relation are selected for markup under left-to-right, shortest match regimen. Thus, The starting locations are selected from left-to-right. If there are overlapping instances of A starting at the same location, only the shortest one is replaced by all strings of B. Each selected A string is marked by inserting all strings of R to its left and all strings of S to its right. The selected A strings themselves remain unchanged, along with the non-A segments.

- XFST syntax: $A->@L \ldots R$

  Meaning: Directed markup. Instances of the language A on the upper side of the relation are selected for markup under right-to-left, longest match regimen. Thus, The starting locations are selected from right-to-left. If there are overlapping instances of A starting at the same location, only the longest one is replaced by all strings of B. Each selected A string is marked by inserting all strings of R to its left and all strings of S to its right. The selected A strings themselves remain unchanged, along with the non-A segments.

- XFST syntax: $A>@L \ldots R$

  Meaning: Directed markup. Instances of the language A on the upper side of the relation are selected for markup under right-to-left, shortest match regimen. Thus, The starting locations are selected from right-to-left. If there are overlapping instances of A starting at the same location, only the shortest one is replaced by all strings of B. Each selected A string is marked by inserting all strings of R to its left and all strings of S to its right. The selected A strings themselves remain unchanged, along with the non-A segments.

2.6 Boundary symbol for restriction and replacement

In the restriction, $=>$, and conditional replacement, $->, (->), <-, (<-), <->, (<->), @->$, $@>, -> @, > @$ expressions we can use a special boundary marker, $.\#.$, to refer to the beginning or to the end of a string. In the left context, the boundary marker signals the beginning of the string; in the right context it means the end of the string. For example, the restriction expression $a =>\_ \ .\#.$ denotes the language where "a" appears only at the end of a string.

The boundary marker is not allowed as a constituent in a symbol pair; otherwise it can be used in a regular expression like any other symbol. For example, in the replacement expression $[a -> b \ || \ [.\#. \mid a]\_ \ ]$ the union in the left context indicates that "a" is to be replaced by "b" at the beginning of a string and after another "a".

Construction: We do not deal with all the cases where the boundary symbol $.\#.$ can be used. We only deal with boundary cases contexts that are in one of the following forms ($LeftContext$ and $RightContext$ are assumed not to contain $.\#.$):

- $.\#. \ LeftContext \ \_ \ RightContext$
- $[.\#. \ LeftContext] \ \_ \ RightContext$
- $[.\#.] \ LeftContext \ \_ \ RightContext$
- $[[.\#.] \ LeftContext] \ \_ \ RightContext$
- $LeftContext \ \_ \ RightContext \ .\#.$
- $LeftContext \ \_ \ [RightContext \ .\#.]$
- $LeftContext \ \_ \ RightContext \ [.\#.]$
- $LeftContext \ \_ \ [RightContext \ [.\#.]]$
- $.\#. \ LeftContext \ \_ \ RightContext \ .\#.$
- $[.\#. \ LeftContext] \ \_ \ RightContext \ .\#.$
- $[.\#.] \ LeftContext \ \_ \ RightContext \ .\#.$
- $[[.\#.] \ LeftContext] \ \_ \ RightContext \ .\#.$
- $.\#. \ LeftContext \ \_ \ [RightContext \ .\#.]$
- $[.\#. \ LeftContext] \ \_ \ [RightContext \ .\#.]$
- $[.\#.] \ LeftContext \ \_ \ [RightContext \ .\#.]$
- $[[.\#.] \ LeftContext] \ \_ \ [RightContext \ .\#.]$
- $.\#. \ LeftContext \ \_ \ RightContext \ [.\#.]$
- $[.\#. \ LeftContext] \ \_ \ RightContext \ [.\#.]$
- $[.\#.] \ LeftContext \ \_ \ RightContext \ [.\#.]$
- $[[.\#.] \ LeftContext] \ \_ \ RightContext \ [.\#.]$

- $.\#.\ LeftContext\ \_\ [RightContext\ [.\#.]]$
- $[.\#.\ LeftContext]\ \_\ [RightContext\ [.\#.]]$
- $[.\#.]\ LeftContext\ \_\ [RightContext\ [.\#.]]$
- $[[.\#.]\ LeftContext]\ \_\ [RightContext\ [.\#.]]$

As we do not deal with restriction rules we need to deal with boundary cases only in replace rules. The replace rules were constructed from six stages: InsertBrackets, ConstrainBrackets, LeftContext, RightContext, Replace and RemoveBrackets. In boundary cases where the left context is in the beginning of a string, only the LeftContext stage is changed. The LeftContext stage was defined as

$$LeftContext = [\sim[\sim[...LEFT]\ \ [<...]\ ]\ \&\ \sim[\ [...LEFT]\ \ \sim[<...]\ ]\ ]$$

where $[...LEFT]$ denoted

$$[\ [\ ?*\ L/[\%<\ \mid \%>]\ ] - [?*\ \%<]\ ]$$

and $[<...]$ denoted

$$[\%</\%>?*]$$

The definition of LeftContext is not changed but the definition of $[...LEFT]$ is changed into

$$[\ [\ L/[\%<\ \mid \%>]\ ] - [?*\ \%<]\ ]$$

In boundary cases where the right context is at the end of a string, only the RightContext stage is changed. The RightContext stage was defined as

$$RightContext = [\sim[\ [...>]\ \ \sim[RIGHT...]\ ]\ \&\ \sim[\sim[...>]\ \ [RIGHT...]\ ]\ ]$$

where $[RIGHT...]$ denoted

$$[\ [\ R/[\%<\ \mid \%>]\ \ ?*\ ] - [\%<?*]\ ]$$

and $[...>]$ denoted

$$[?*\ \%>/\%<]$$

The definition of RightContext is not changed but the definition of $[RIGHT...]$ is changed into

$$[\ [\ R/[\%<\ \mid \%>]\ ] - [\%<?*]\ ]$$

In boundary cases where both the right context and the left context are at the end and in the beginning of a string respectively, both the RightContext and the LeftContext stages are changed as described above. The idea behind these changes is that the context part of replacement expression can be actually seen as $?*\ \ LEFT\ \_\ RIGHT\ ?*$ and by simply eliminating one of the $?*$ in one of the ends we can relate to a boundary case. The definitions that was changed above did exactly that - eliminated the appropriate $?*$ for each case. More complicated cases, as for example $a->b\ ||\ [.\#.\mid a]\ \_$ should be dealt by conditional parallel replacement. For example, $a->b\ ||\ [.\#.\mid a]\ \_$, should be interpreted as

$$a->b\ ||\ [.\#.]\ \_\ ,,\ a->b\ ||\ [a]\ \_$$

Since we do not deal with conditional parallel replacement, we can not deal with these cases.

2.7 Order of Precedence

The following list states the order of precedence of all above operators in XFST. Operators of same precedence are executed from left to right, except the prefix operators ($\sim$ \ \$ \$? \$.) that are executed from right to left. To enforce another order use angle brackets [ ]. The list begins with the operators of highest precedence, i.e. with the most tightly binding ones. Operators of same precedence are on the same line.

:
$\sim$ \ \$ \$? \$.
$+$ $*$ ^ .1 .2 $.u$ $.l$ $.i$ $.r$
/
concatenation
$>$ $<$
| & $-$
$=>$ $->$ $(->)$ $<-$ $(<-)$ $<->$ $(<->)$ @$->$ @$>$ $->$@ $>$@
$.x.$ $.o.$

The order of precedence in FSA is different of the order of precedence in XFST. To avoid of unexpected results while translating regular expressions from XFST into FSA we use the ( ) brackets that control the precedence order in FSA (as [ ] does that in XFST). During the translation, each operation is rounded with ( ) brackets to enforce the right order.

2.8 Advanced XFST techniques

In the XFST toolbox exist some other techniques that were not mentioned, such as Compile-Replace and Flag-Diacritics. Since these methods are not “regular expressions” in the simple sense but more of an advanced methods and moreover, they are compile time techniques, they are beyond the scope of this paper and we do not relate to them.

3 Results and their analysis

3.1 Theoretical aspect

As a preliminary step for implementing a compiler from XFST to FSA we needed to do two things: the first was to construct a grammar for XFST, and the second was to express each XFST operation using the FSA syntax. Since XFST is propriety, both of the things were not fully available and a lot of theoretical work had to be done in order to achieve them. The grammar we constructed is almost full, and seems to capture almost any XFST syntax. The second part, concerning the XFST operations translations was more problematic. Most of the basic operators exist also in the FSA Utils or could be constructed with not much of an effort. The problem arises when reaching to the more complicated operators - restriction, replacement and markup. These operations does not exist in FSA at all, thus, we needed to construct them from the basic operators. Most of these constructions, as we discovered, are not trivial at all and require some complicated stages. The problem we tackled was that there was only little available documentation about the algorithms for constructing these operations. For some of the operations without an existing available documentation, we succeeded in finding the construction algorithms, but there were still a lot of operations for which we did not find the construction algorithm. Moreover, we found that for some of the known algorithms - they were not fully accurate and did not deal with all the cases, but we were able to find the misaccuracies and fix them and to find the way to deal with the cases that were not dealt with. Furthermore, the main discovery was that the published algorithms gave different results than the XFST gave. The meaning of this is that in some cases given a regular expression, building the matching network using XFST or using the construction algorithms results with two different networks, yet equivalent in the relation they define, i.e., for each input string the two groups of outputs in the two networks are equivalent, but some outputs are produced in more paths in the direct implementation network than in the XFST network. Since when referring to a group there is no meaning to the number of times a member in the group occurs, we can say that the two groups of outputs were equal in the two networks. The main occurrence of this was in the conditional replace rules (i.e., replace rules with context) where for each replace path that the XFST gave, a direct implementation using the construction algorithm resulted with two paths. Specifically, in the network created by direct implementation of the construction algorithm, instead of getting only a path that contains in some stage the pair upper:lower we get two paths, one with upper:lower and one with the concatenation of two pairs: upper:0 and 0:lower (0 represent the empty string in XFST). For example, consider the replace rule $a -> b \,||\, c\_\ d$. Inserting this rule directly into xfst results with the following network:

Sigma: ? a b c d
Size: 5
Net: 31B6F8
Flags: deterministic, pruned, minimized, epsilon_free
Arity: 2
fs0: ? $->$ fs0, a $->$ fs0, b $->$ fs0, c $->$ fs1, d $->$ fs0.
fs1: ? $->$ fs0, a $->$ fs2, b $->$ fs0, c $->$ fs1, d $->$ fs0, <a:b> $->$ s3.
fs2: ? $->$ fs0, a $->$ fs0, b $->$ fs0, c $->$ fs1.
s3: d $->$ fs0.

Compiling the rule to a network using the published algorithm results with the following network:

Sigma: ? a b c d < >
Size: 7
Net: 31B438

Flags: deterministic, pruned, minimized, epsilon_free
Arity: 2
fs0: ? − > fs0, a − > fs0, b − > fs0, c − > fs1, d − > fs0.
fs1: ? − > fs0, a − > fs2, b − > fs0, c − > fs1, d − > fs0, <a:b> − > s3,
<a:0> − > s4.
fs2: ? − > fs0, a − > fs0, b − > fs0, c − > fs1.
s3: d − > fs0.
s4: <0:b> − > s3.

Careful inspection of this two networks results with the conclusion that they describe the same relation but they do not equal. In the first network there is only one stage where the replace a:b occurs as in the second network there are two stages where the replace occurs - one by a:b and the second by a:0 at first and 0:b afterwards. The same happens in all of the conditional replace rules. Thus, it is clear that the XFST toolbox uses somewhat different construction algorithms than we obtained. We feel that this is an important insight regarding the XFST toolbox and replace rules in general. It shows that the problem is more difficult than it sometimes seems and regarded to be.

3.2 Practical results - comparison between XFST and FSA

After having the compiler, an interesting thing to do was to take some XFST code, compile it into an equivalent FSA code and compare the two resulting networks. The networks that we got were not the same in cases of replace rules from the reasons mentioned in section 3.1 but behaved the same in the sense that for each input string they produced the same group of outputs. We could not perform a comparison of the networks to inspect their equality since they are not equal. What we could do was to inspect the way they are displayed. XFST and FSA display the resulted network in different ways, using different notations. The XFST network uses two kinds of arc symbols notations: a symbol used in the regular expression and ?, denoting unknown symbol (a symbol that did not appear anywhere in the regular expression). The FSA network uses three kinds of arc symbols notations: a symbol used in the regular expression, ?, denoting any symbol (including those that appear elsewhere in the network) and ’symbol denoting anything that is not symbol. Thus, equivalent networks in XFST and FSA (and also networks denoting the same language but not equivalent) have different representation. It seems that the XFST representation is more compact and more convenient for the user in terms of simplicity of understanding what is the language or relation it describes. In the implementation of markup and replace rules when using symbols assumed not to be in Σ, the symbols do not effect the behavior of the resulting network on an input string. The XFST representation has the advantage of not showing these symbols on the arcs in the resulting network. However, the resulting network in the FSA Utils has these symbols on its arcs since it uses the notation of ’symbol. Thus, the resulting network is less understandable and seems more cumbersome. This is a clear advantage of the XFST toolbox.

4 Possible enhancements and future work

As we mentioned, not all XFST operations were implemented in FSA. Thus, further research can be done in this direction to find the construction of other XFST operators that were not constructed here. Such a research requires an high level understanding of finite state techniques and operations and can lead to further insight of the XFST toolbox.

5 Installer's Guide

System requirements:

- Linux
- Perl v5.8 or above
- FSA

Perl and FSA are not mandatory. Perl is required only if your input file is not in the right form (further explanation can be found in the user's guide - section 6). FSA is required only if you want to run your output file on the FSA Utils.

How to install the program?

No installation process is needed. All you need to do is to unzip the package, there you will find xfst2fsa.exe which is the only file you need in order to run the program. If you want to run preprocessing on your input file (as will be explained in the user's guide - section 6) you need also correct_syntax.perl.

And if I want to build it myself?

Further explanation on how to build it yourself or make changes in the code files can be found in the programmer's guide - section 7)

6 User's Guide

The program gets as input an XFST file and returns an FSA macros file. In order to use the program you need to know:

1. The structure of the input file
2. How to run the program
3. The structure of the output file
4. How to run the output file on FSA

6.1 The structure of the input file

The structure of the input file that the program excepts is more limited than the structure that XFST excepts. The extra restrictions are usually caused from the FSA Utils restrictions. The structure of the input file for this program should be as follows:

- The XFST commands that can be used are: define, ! (remark), clear stack, minimize, echo, read regex and regex.
- Each XFST command should begin in a different line except for a remark (!).
- The read regex (or regex) can be used only once and if used it must be at the end of the file and that line must end with \n (enter).
- In section 2 all XFST operators were presented. All those operators can be used except for the following:
  - $R.P.Q$ (upper side priority union)
  - $R.p.Q$ (lower side priority union)
  - $R.P.Q$ (upper side priority union)
  - $R.-u.Q$ (upper side minus)
  - $R.-l.Q$ (lower side minus)
  - $A <> B$ (shuffle)
  - $'[[A],\ s,\ sl]$
  - The boundary symbol $.\#.$ can be used only in one of the forms described in section 2.6.
  - Operators that were not translated into FSA and are not in this list can be used, but will not be translated into FSA, thus, there is no guaranty to the way the FSA output file will behave in the FSA Utils if using them.
- The input file can contain empty lines everywhere.
- When using the define command, the definition name should be a sequence of letters and numbers, containing at least one letter.
- The characters $\{, \}, \backslash$ and / can not be used as symbols or as a part of multicharacters symbols.

- The following multicharacter symbols are reserved to the program usage. Using them can cause unexpected results. These are: >>>,<<<,^^^ or any multicharacter symbol beginning with either one of the strings regex, lower, upper, stage, n_mac, n2_mac, r_mac or r2_mac.

In practice the structure of the input file is even more restricted, and includes in addition to the above restrictions also the following ones:

- Remarks (!) can appear in separate lines only.
- Each command must begin in the beginning of the line (with no tabs and spaces before).
- When using a define command, only one space (and not tab) should separate between the command define and the definition name.
- The definitions names must not begin with capitals letters (since FSA is built over Prolog, there are some Prolog restriction in FSA).
- The character ^ can not be used as a symbol or as a part of a multicharacter symbol.
- The file must end with a read regex or regex command (with \n at the end as explained above).

If your file does not violate any of these extra requirements you will not need to run the perl script. If your file does violate some of this extra requirements, you do not need to correct your file, just run the perl script (as will be explained later) before running the program. This script will create a corrected file for you.

6.2 How to run the program

1. If you need to run the perl script (see explanation in section 6.1) run the command:
   `% perl correct_syntax.perl input_file output_file`
   where input_file is your input file with the xfst code. output_file is the file into which the corrected code will be written.

2. Run the program by using the command:
   `% ./xfst2fsa.exe correct_input_file output_file.pl`
   where correct_input_file is your input file if you did not perform step 1 above or the output file you got from running step 1 above. Notice that the output file must have the .pl extension. This is from reasons of applicability to FSA. output_file.pl contains the equivalent FSA code.

6.3 The structure of the output file

This section is not necessary for running the output file on the FSA Utils, but if you want to learn a little bit about its structure, you want to read it.

After running the program you get a file containing the equivalent FSA code. If you will look at the output file you will be able to recognize the input file code, but you will find a completely different syntax which is FSA syntax combined with prolog (the FSA Utils is built over prolog). Some of the main changes are listed:

- Each remark is translated into an equivalent prolog remark (the character ! is replaced by %%).
- Each one of the XFST commands clear stack, minimize or echo appears in a prolog remark (since they are not relevant for the FSA code).

- Each definition (the XFST command define) in the XFST input file has a macro (may be using other macros) defining it. The definition name is prefixed with 'xxx' in all the places it occures.
- The read regex or regex XFST command is translated into a macro with the name regex.

6.4 How to run the output file on FSA

Now that you have an equivalent FSA code file, you probably want to run it on the FSA Utils. So, here is how you should do it (this instructions are taken out of the FSA manual, were you can find them yourself):

1. You will need the output file the program generated and the file macros.pl in the package you unzipped. The file macros.pl contains macros that the program uses for the XFST to FSA translation.
2. Open the FSA Utils. If you want to use FSA interactively with the graphical interface, use the command
   ```
   % fsa -tk
   ```
3. You need to let the fsa know that you want it to recognize the code in the files macros.pl and the program output file. You do that by using the -aux[file] command line option, or the AuxFile button of the TK Widget.
4. Now you can use in the command line or at the regex box (in the graphical interface) any of the definitions you had in your XFST file (just remember to prefix it with 'xxx'). The read regex command can be used by referring to 'regex'.

6.5 Example files

The package contains some input file for example. This files are tes1.txt, test2.txt and input.xfst. The file input.xfst contains examples for all the XFST operations. You do not need to run it under the perl script, you can directly run it threw the program. The files test1.txt and test2.txt contains XFST code which is a part of a computational system for morphological tagging of the Qur'an. You need to run them first under the perl script. Running them threw the program will yield some parse error of the program. The reason is that this files contain some operators that are not supported by our program. You can still run the output files on the FSA Utils, but of obvious reasons the results are not correct. You can use these two files to inspect the effect of running the perl script over a file and for inspecting how the program handles parse error occurrences.

7 programmer's guide

This section gives further explanation on the different files constructing the program for programmers who wish to understand the implementations and perhaps introduce some changes in it and for users who wish to build the program their self.

The program was written using the ansi C programming language and using Flex and Bison. A perl script was written for the preprocessing stage of the input file.

The package contains all the program source files:

- **xfst2fsa.l** - Flex file. Responsible for reading the tokens from the input file.
- **xfst2fsa.y** - Bison file. Contains the XFST grammar and responsible for parsing the input file using the tokens read by the flex file into a parsing tree.
- **xfst2fsa.c** - ansi C code file. This is the main program. It calls yyparse() to create the input parsing tree and by walking over the tree generates the appropriate FSA code into the output file given by the user.
- **typedef.h** - a C header file. This files contains all the definition and functions prototypes.
- **correct_syntax.perl** - a perl file. This perl script is responsible for the preprocessing stage of the input file in cases that the XFST code is not in the same form required by the program (as explained in section 6.1).
- **macros.pl** - a prolog file combined with FSA syntax. This file contain macros that are used by the output FSA code file. Most of the macros in this file are for the replace and markup rules. As shown in section 2 the construction of the replace and markup operations requires some stages and is not trivial. Thus, we wrote macros for this operations to make a clearer FSA code file. Moreover, this file can be used in the FSA Utils for using replace and markup rules that are not implemented in FSA.
- **input.xfst, test1.txt, test2.txt** - example input files. An explanation about them was already given in section 6.5.

How can I build the program myself?

The program can be built by running the series of following commands:

```
% flex xfst2fsa.l
```

This command run flex on the file xfst2fsa.l and creates the file lex.yy.c .

```
% bison -d xfst2fsa.y
```

This command runs bison of the file xfst2fsa.y and creates the files xfst2fsa.tab.c and xfst2fsa.tab.h .

```
% gcc -Wall -ansi -pedantic -c lex.yy.c
```

This command compiles the file lex.yy.c and creates the file lex.yy.o .

```
% gcc -Wall -ansi -pedantic -c xfst2fsa.tab.c
```

This command compiles the file xfst2fsa.tab.c and creates the file xfst2fsa.tab.o .

```
% gcc -Wall -ansi -pedantic -c xfst2fsa.c
```

This command compiles the file xfst2fsa.c and creates the file xfst2fsa.o .

```
% gcc xfst2fsa.o xfst2fsa.tab.o lex.yy.o -o xfst2fsa.exe
```

This command links together the files xfst2fsa.o, xfst2fsa.tab.o and lex.yy.o and creates the executable file xfst2fsa.exe (of course it can be named by a different name). Now that you have the file xfst2fsa.exe, you can run the program as explained in section 6.2.

8 Acknowledgements

This program was done as a part of the Laboratory in natural languages processing – Computer science department, University of Haifa - winter semester 2002-2003 under the instruction of Dr. Shuly Wintner. We thank Shlomo Yona for writing the Perl script for this project.

References

Beesley, Kenneth R. and Lauri Karttunen. Forthcoming. *Finite-State Morphology: Xerox Tools and Techniques*.

Gerdemann, Dale and Gertjan van Noord. 1999. Transducers from rewrite rules with backreferences. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 126–133, Bergen Norway.

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

Karttunen, Lauri. 1995. The replace operator. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 16–24.

Karttunen, Lauri. 1996. Directed replacement. In *Proceedings of ACL'96*, pages 108–115.

Karttunen, Lauri. 1997. The replace operator. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, Language, Speech and Communication. MIT Press, Cambridge, MA, chapter 4, pages 117–147.

Karttunen, Lauri and Andre Kempe. 1995. The parallel replacement operation in finite state calculus. Technical Report 1995-021, Rank Xerox research centre – Grenoble laboratory.

van Noord, Gertjan, 2000. *FSA6 Reference Manual*.