Text Normalisation in Text-to-Speech Synthesis for South African Languages: Native Number Expansion

Georg I. Schlünz, Nkosikhona Dlamini, Alfred Tshoane and Stan Ramunyisi
Human Language Technology Research Group
CSIR Meraka Institute
Pretoria, South Africa
gschlunz@csir.co.za, ndlamini3@csir.co.za, atshoane@csir.co.za, sramunyisi@csir.co.za

***Abstract*—Text normalisation in text-to-speech synthesis comprises the segmentation and classification of the incoming text and the subsequent expansion of non-standard words into their standard word, spoken forms. We present a rule-based implementation for the 11 official South African languages that uses native number expansion. We discuss the architecture and performance of the implementation based on examples of cardinal and ordinal numbers, money, dates and times. Although the implementation scores well, it is currently limited to handling non-standard words in isolation. Future work will need to address sentence context in order to normalise the African languages correctly.**

I. Introduction

The frontend of a text-to-speech (TTS) system processes the incoming text in various phases. These include segmenting the text into tokens, expanding non-standard words into standard words [1], mapping graphemes to phonemes and extracting prosodic markers from the text. *Text normalisation* encompasses the first two phases, namely text segmentation and non-standard word expansion. We employ the terminology from [2] and [3] and refer to and distinguish between them as *tokenisation and classification* and *verbalisation.*

The *tokenisation and classification* phase splits the text into tokens mostly delimited by whitespace and punctuation symbols. It simultaneously classifies the tokens as standard words or non-standard words such as numbers (123), money (R99), dates (13/03/2012) and times (08:45). The latter are called *semiotic classes*. The *verbalisation* phase expands the non-standard words into their standard word, spoken form, shown in Figure 1.

123	→	one hundred and twenty three
R99	→	ninety nine rand
13/03/2012	→	thirteen march twenty twelve
08:45	→	eight forty five a_letter m_letter

Fig. 1. Verbalisation of non-standard words

In related work, Google has implemented the Kestrel TTS text normalisation system [3] in their commercial TTS offering. They use text-normalization grammars that are compiled into libraries of weighted finite-state transducers (WFSTs). Input text is first tokenized and different tokens classified using WFSTs, where semiotic classes are parsed into data structures called protocol buffers. The protocol buffers are then verbalised, with possible reordering of the elements, again using WFSTs.

Microsoft designed an algorithm, called FlashNormalize [4], that learns programs for text normalization through examples. A domain-specific programming language (DSL) represents the concept space and offers abstractions for expressing such normalisation. The DSL is structured around four kind of expressions, namely (1) a *parse* expression to extract appropriate substrings from the input string, (2) a *process* expression to transform the substrings using table lookups or functions provided by the language designer, (3) a *concat* expression to concatenate various process expressions, and (4) *decision lists* that allow for conditional behaviour. A deductive top-down search algorithm is used to learn both the decision lists and concat expressions that are consistent with the set of input-output examples.

Recent research into the application of deep learning to text normalisation include [5], [6] and [7]. The latter authors concede that "Though our conclusions are largely negative on this point, we are actually not arguing that the text normalisation problem is intractable using a pure recurrent neural network (RNN) approach, merely that it is not going to be something that can be solved merely by having huge amounts of annotated text data and feeding that to a general RNN model. And when we open-source our data, we will be providing a novel data set for sequence-to-sequence modeling in the hopes that the the community can find better solutions." [8][9].

In this paper, we present a rule-based implementation of text normalisation for the Speect TTS system, which is commercialised under the brand name "Qfrency TTS" [10]. The rule engine is separated from the rules in a modular design to allow user-defined extensions and customisations to the rules. We cover all 11 official South African languages, in particular using the native spoken forms of number-based non-standard words during verbalisation. We describe the implementation in Section II, its evaluation in Section III and conclude in Section IV.

II. Implementation

A. Algorithm Overview

1) Tokenisation and Classification: Firstly, tokens are segmented and classified by regular expressions in decision lists. Each semiotic class specifies an unordered inner list of hand-crafted regular expressions to match against the incoming text stream. An outer list iterates over the semiotic classes, in a hand-selected order governed by their uniquely differentiating contexts. For example in Figure 2, a (simplified) money class must be placed before a (simplified) decimal number class that, in turn, must be placed before a cardinal number class, in order to leverage the differentiating contexts of the currency symbol "R" and the decimal point ".", respectively.

```
"money": {
        "mask": [
                "^R\\d+(\\.\\d\\d)?\\b" ] },
"decimal": {
        "mask": [
                "^\\d+\\.\\d+\\b" ] },
"card": {
        "mask": [
                "^\\d+\\b" ] },
```

Fig. 2. Tokenisation and classification rules for semiotic classes

2) Verbalisation: Secondly, the classified tokens are verbalised using hand-crafted regular expressions that break down the tokens in a particular, mutually exclusive order of its constituents. The regular expressions form the left-hand side of rewrite rules that perform substitutions on the token constituents, in a style similar to the productions of nonterminals in a context-free grammar, where:

- "<*class*<" means recurse with the rewrite rule for *class* on the first matched substring of the current token constituent, and substitute with the result
- ">*class*>" means recurse with the rewrite rule for *class* on the rest/remainder of the original string (i.e. the inverse of the match), and substitute with the result
- "#*class*#" means recurse with the rewrite rule for *class* on a split of the matched substring into its constituent characters, and substitute with the result

Analogous to the terminals in a context-free grammar, the right-hand side of the rewrite rules represent standard word, spoken form expansions of the constituents of the non-standard words, where:

- "@*class*@" means execute the number expansion rule for *class* on the matched substring

The (simplified) rewrite rules for the semiotic classes of money, decimal and cardinal numbers in English are illustrated in Figure 3.

```
"money": {
            "^R\\d+": "<card< rand",
            "\\.\\d\\d$": " and <card< cents" },
"decimal": {
            "^\\d+": "<card<",
            "\\.\\d+$": " point #card#" },
"card": {
            "[^\\d]+": ">card>",
            "^\\d+$": "@card@" },
```

Fig. 3. Verbalisation rules for semiotic classes in English

3) Number Expansion: Numbers specifically are expanded by exploiting their inherently recursive verbal structure, through repeated division and modulo operations. We drew up a representative list of number-based non-standard words that fall in the various semiotic classes of cardinal numbers, ordinal numbers, money, dates and times. We contracted language practitioners to expand the list items to their native, standard word, spoken forms. We used this gold standard list to extract patterns unique to each class for the development of the number expansion rules. We implemented the recursive rewrite rule design of [11] and describe it by way of English in Figure 4.

```
%card
 0: zero;
 1: one;
 2: two;
 3: three;
...
10: ten;
11: eleven;
12: twelve;
13: thirteen;
...
20: twenty; twenty >>;
30: thirty; thirty >>;
40: forty; forty >>;
50: fifty; fifty >>;
...
100: << hundred; << hundred >>;
1000: << thousand; << thousand >>;
1000000: << million; << million >>;
1000000000: << billion; << billion >>;
...
```

Fig. 4. Cardinal number expansion rules for English

A rewrite rule maps a base value to text representing the associated standard word, spoken form expansion. A rule applies to all numbers from its base value to one less than the next rule's base value. If a rule does not specify a base value, its base value is the previous rule's base value plus one. The associated text can contain major ("<<") and/or minor (">>") substitutions.

For numbers greater than the base value in each of the number ranges of twenties, thirties, up until nineties, we recurse with the previous rules to expand the digit in the ones place, denoted by the minor substitution ">>". The rules for hundreds follow the same recursive principle to leverage the existing rules to expand tens and ones digits, using the additional major substitution "<<". Whereas the minor substitution is filled by taking the number being expanded *modulo* 10 to the rule's power of 10, the major substitution is filled by taking the number being expanded *divided* by 10 to the rule's power of 10 and truncated to an integer. Thousands, millions, billions and beyond follow suit.

For the 2 Germanic languages of English and Afrikaans, morphology does not pose any challenge in the application of this algorithm. However, when doing native number expansion for the 9 African languages, certain morphological constraints need to be considered. Plurality is one such constraint, for example, the number 200 is plural, since there are two hundreds present, whereas the number 100 is singular, since there is just a single hundred in the number. This distinction is necessary because the prefix used for a singular number is different from that of a plural number.

It is also important to consider the semiotic class of the number, whether it is cardinal, ordinal, money, date, time, et cetera. The prefix needed for either singular or plural depends on the class of the number in question. Numbers can be realised in general as ones, tens, hundreds, thousands, et cetera. We can refer to these as the root of the number. All ones have unique roots, all tens share a root and all hundreds share a root, et cetera. A root does not change when the number is expanded; what changes is the prefix.

The next subsections elaborate on the algorithm with selected examples for the cardinal number, money and date semiotic classes in the various languages.

B. Cardinal Numbers

Tokenisation and classification:

```
"card": {
        "mask": [
             "^\\d+((\\s+|,)\\d\\d\\d)*\\b" ] },
```

Verbalisation:

```
"card": {
            "[^\\d]+": ">card>",
            "^\\d+$": "@card@" },
```

Fig. 5. Cardinal number semiotic class

Figure 5 shows the tokenisation and classification, and verbalisation rules for the cardinal number semiotic class. English and Afrikaans follow the same patterns of cardinal number expansion. It is also consistent within the African languages. We define a rule set for numbers between zero and ten, and then start making use of the previously defined rule sets, as well as other rules to define rule sets for numbers 11 and higher. This procedure works until we get to 20, where a new rule comes into play; however, the previously defined rules for numbers less than 11 are reused whenever applicable. The rule for 20 can then be used until we get to 99, and the same pattern follows for thousands, millions and billions.

As an example, the cardinal number 123 is broken down into hundreds, tens and ones. The rule for 100 will be applied first, followed by that for 20 and then that for 3. Table I compares the roots used for these numbers across a subset of the African languages.

TABLE I
Root usage in the cardinal number expansion rules

Base	**Root in expansions**					
	nso	**tsn**	**zul**	**ssw**	**ven**	**tso**
100	kgolo	kgolo	khulu	khulu	ḓana	dzana
20	some	some	shumi	shumi	fumi	khume
3	tharo	tharo	ntathu	tsatfu	tharu	nharhu
2	pedi	pedi	bili	bili	mbili	mbirhi

For Sepedi and Setswana, we consider how many hundreds and tens the number 123 has, so that we can identify whether they are singular or plural and select the corresponding prefix. For 100, we obtain "*le* (singular prefix) + kgolo (hundreds root) + *le* (conjunction) = *le*kgolo *le*". Then 23 is looked up again and the rule with a base value of 20 is applied. For 20, we obtain "*ma* (plural prefix) + some (tens root) + pedi (two root) = *ma*somepedi". This is then appended to "*le*kgolo *le*" to form "*le*kgolo *le ma*somepedi". Finally, we look up 3 as "tharo (three root)" and append it again to produce "*le*kgolo *le ma*somepedi tharo". Table II shows the cardinal number 123 expanded in all 11 languages.

TABLE II
Cardinal number expansion

Lang	**Expansion**
eng	one hundred and twenty three
afr	een honderd drie en twintig
nso	*le*kgolo *le ma*somepedi tharo
tsn	*le*kgolo *le ma*somepedi tharo
sot	lekgolo mashome mabedi metso meraro
zul	*i*khulu *nama*shumi *ama*bili *na*ntathu
xho	ikhulu elinamashumi amabini anantathu
ssw	*li*khulu *nema*shumi *lama*bili *naku*tsatfu
nbl	*li*khulu *nama*sumi *ama*bili *na*ntathu
ven	ḓanafumbiliraru
tso	dzana na khumembirhinharhu

C. Money

Figure 6 shows the tokenisation and classification, and verbalisation rules for the money semiotic class in English. The African languages historically used to employ a

Tokenisation and classification:

```
"money": {
        "mask": [
                "^(R|r|\\$|£|€)\\s*\\d+((\\s+|,)\\d\\d\\d)*(\\.\\d\\d)?\\b",
                "^\\d+((\\s+|,)\\d\\d\\d)*(\\.\\d+((\\s+|,)\\d\\d\\d)*)?\\s*c\\b" ] },
```

Verbalisation:

```
"money": {
            "[Rr]\\s*\\d+((\\s+|,)\\d\\d\\d)*": "<card< rand",
            "\\$\\s*\\d+((\\s+|,)\\d\\d\\d)*": "<card< dollars",
            "£\\s*\\d+((\\s+|,)\\d\\d\\d)*": "<card< pounds",
            "€\\s*\\d+((\\s+|,)\\d\\d\\d)*": "<card< euros",
            "\\.01": " and one cent",
            "\\.((0[2-9])|([1-9][0-9]))": " and <card< cents",
            "^\\d+((\\s+|,)\\d\\d\\d)*(\\.\\d+((\\s+|,)\\d\\d\\d)*)?\\s*c$": "<decimal< cents" },
```

Fig. 6. Money semiotic class for English

different way of handling number expansion in monetary context, where money was counted in R2 rather than R1, in contrast to English and Afrikaans. An amount such as R20 would be expanded as "ten two rands". In Tshivenda, R30 would be expanded as "fumi ḽa bonndo na nṱhanu", which translates to "ten two rands and five two rands". However, this procedure has become almost obsolete and the new generation has adopted a procedure similar to that of English and Afrikaans.

To handle number expansion of money, the algorithm uses the cardinal number rules to expand the number part of the amount, and then prefix the expanded number with a monetary unit specifying the quantity of this monetary value. However, for some languages like isiZulu and siSwati, the cardinal number rule set does not apply correctly all the time. A new rule had to be added specifically dealing with money expansion. The following examples illustrate the difference when expanding the cardinal number 2 versus the amount R2.

For isiZulu, the expansion of the cardinal number 2 is "*ku*bili" and that of the amount R2 is "amarandi *ama*bili". If cardinal number rules were to be used, then the amount would be incorrectly expanded as "amarandi *amaku*bili". For siSwati, the expansion of the cardinal number 2 is "*ku*bili" and that of the amount R2 is "emarandi *lama*bili". If cardinal number rules were to be used, then the amount would be incorrectly expanded as "emarandi *lamaku*bili". Table III outlines the expansion and added prefixes of the example amount R123 in all 11 languages.

TABLE III
MONEY EXPANSION

Lang	**Expansion**
eng	one hundred and twenty three rand
afr	een honderd drie en twintig rand
nso	*diranta tše* lekgolo le masomepedi tharo
tsn	*diranta tse* lekgolo le masomepedi tharo
sot	diranta tse lekgolo mashome mabedi metso meraro
zul	*amarandi* ayikhulu namashumi amabili nantathu
xho	iirandi ikhulu elinamashumi amabini anantathu
ssw	*emarandi* lalikhulu nemashumi lamabili nakutsatfu
nbl	amaranda eziyikhulu namasumi amabili nantathu
ven	*rannda dza* ḓanafumbiliraru
tso	dzana na khumenharhu wa tirhandi

D. Dates

Figure 7 shows the tokenisation and classification, and verbalisation rules for the date semiotic class. The indigenous languages share commonalities when expanding the date and time. In order to expand dates, the algorithm splits a date into day, month and year, and applies separate rules for day, month and year. To handle months, the algorithm looks up how the months are written in the different languages. The way the day and the year are expanded differs. Some of the languages will reuse rules from cardinals and ordinals, other languages like isiZulu and siSwati again require that a new rule set be defined specifically for the date. This difference entails which prefix needs to be added to the root of the number.

For isiZulu, the prefix "*wezi*" is added to the root in the thousands rules and the prefix "*ezi*" is added to the root in the tens rules. For siSwati, the prefix "*weti*" is added to the root in the thousands rules and the prefix "*leti*" to the root in the tens rules. Table IV illustrates the expansions for the date 13/03/2012.

III. Evaluation

A. Methodology

To evaluate the accuracy of our implementation, we compared the generated output to the gold standard produced by the language practitioners. We used the python "difflib" module [12] to calculate string similarity ratios on the character level rather than the word level, in order to normalise very loosely over the morphology of the languages on the disjunctive to conjunctive spectrum.

Tokenisation and classification:

```
"date": {
"mask": [
"^((\\d?\\d[.\\-/]\\d?\\d[.\\-/]\\d\\d\\d\\d)|(\\d\\d\\d\\d[.\\-/]\\d?\\d[.\\-/]\\d?\\d))\\b" ] },
```

Verbalisation:

```
"date": {
            "(^(([0-2]?[0-9])|(3[0-1]))[.\\-/])|([.\\-/](([0-2]?[0-9])|(3[0-1]))$)": "<day<",
            "[.\\-/]((0?[1-9])|(1[0-2]))[.\\-/]": " <month<",
            "[0-9][0-9][0-9][0-9]": " <year<" },
"day": {
            "\\d+": "<card<" },
"month": {
            "[^\\d]+": ">month>",
            "^0?1$": "january",
            "^0?2$": "february",
            "^0?3$": "march",
            ...
            "^12$": "december" },
"year": {
            "0[0-9][0-9][0-9]": " the year <card<",
            "[1-9]00[0-9]": " <card<",
            "[1-9][1-9]00": " <year_cent<",
            "[1-9](([0-9][1-9][0-9])|([1-9][0-9][1-9])|([1-9][1-9][0-9]))": " <year_other<" },
"year_cent": {
            "^\\d\\d": "<card< hundred" },
"year_other": {
            "^\\d\\d": "<card<",
            "\\d\\d$": " <card<" },
```

Fig. 7. Date semiotic class for English

TABLE IV
DATE EXPANSION

Lang	**Expansion**
eng	thirteen march twenty twelve
afr	dertien maart twintig twaalf
nso	la lesome tharo matšhe ngwaga wa ketepedi le lesome pedi
tsn	la lesome tharo mopitlwe ngwaga wa kete pedi le lesome pedi
sot	la leshome le metso e meraro hlakubele selemo sa dikete tse pedi le leshome le metso e mmedi
zul	ziyishumi nantathu kundasa unyaka wezinkulungwane ezimbili neshumi nambili
xho	umhla weshumi elinantathu kumatshi ngowamawaka amabini aneshumi elinesibini
ssw	tilishumi nakutsatfu indlovu lenkhulu umnyaka wetinkhulungwane letimbili nelishumi nakubili
nbl	ilanga lesumi nakuthathu enyangeni yesithathu ngom-nyaka weenkulungwana ezimatjhumi amabili ne-minyaka elisumi nambili
ven	ḽa vhufumiraru ḽa ṱhafamuhwe ṅwaha wa gidimbilifumimbili
tso	khumenharhu nyenyankulu hi lembe ra gidimbirhi khumembirhi

From the python documentation, "The basic algorithm predates, and is a little fancier than, an algorithm published in the late 1980's by Ratcliff and Obershelp under the hyperbolic name 'gestalt pattern matching.' The idea is to find the longest contiguous matching subsequence that contains no 'junk' elements (the Ratcliff and Obershelp algorithm doesn't address junk). The same idea is then applied recursively to the pieces of the sequences to the left and to the right of the matching subsequence. This does not yield minimal edit sequences, but does tend to yield matches that 'look right' to people."

The module calculates a sequence ratio between 0.0 and 1.0. From the documentation, "Where T is the total number of elements in both sequences, and M is the number of matches, this is 2.0*M / T. Note that this is 1.0 if the sequences are identical, and 0.0 if they have nothing in common".

We utilised 66 test cases each for the cardinal and ordinal number classes across all 11 languages. For the money, date and time classes, we used 51 test cases in total (spread evenly) for all languages except Sesotho, isiXhosa and isiNdebele, which employed 34 test cases each. The resources that had worked on the latter three languages became unavailable during the course of development, limiting our capability to address these languages fully.

B. Results

In the continuous process of evaluating our implementation, we discovered discrepancies in the expansions that we resolved using third party input. Furthermore, the evaluation helped to fix errors in the number expansion rules iteratively. This enabled us eventually to obtain 1.0 scores for all the test cases in all the languages, except for Sesotho, isiXhosa and isiNdebele. We suspect that 1.0 scores will also be possible when we do future iterations on the languages. For the record, the scores are stated in Table V.

TABLE V
String similarity scores of the implementation against the gold standard

Lang	Class scores				
	Card	**Ord**	**Money**	**Date**	**Time**
eng	1.0	1.0	1.0	1.0	1.0
afr	1.0	1.0	1.0	1.0	1.0
nso	1.0	1.0	1.0	1.0	1.0
tsn	1.0	1.0	1.0	1.0	1.0
sot	0.96	0.96	1.0	0.97	1.0
zul	1.0	1.0	1.0	1.0	1.0
xho	0.95	0.95	0.86	0.95	0.84
ssw	1.0	1.0	1.0	1.0	1.0
nbl	0.96	0.94	0.98	0.91	1.0
ven	1.0	1.0	1.0	1.0	1.0
tso	1.0	1.0	1.0	1.0	1.0

IV. Conclusion

The rule-based text normalisation implementation performs well for all the South African languages. Even though the rules are more complex to construct for the morphologically rich African languages than the Germanic languages, it was nonetheless possible to re-use them among the semiotic classes. For example, rules for expanding cardinal numbers were used for ordinal numbers, with minor changes to the prefixes only. Rules defined for a particular class can be applied successfully to possibly all numbers belonging to that class, and the resulting expansion will be correct. This is reflected in the high string similarity scores.

The advantage of a rule-based approach is the transparency in the implementation when a violation of the language is observed during testing against a gold standard. It allows for rapid fixes to existing rules and extensions to new rules to improve the coverage. A rule-based approach has conditions that must be met, though. The developer must have one foot in computer science to understand and use constructs like regular expressions and rewrite rules, and the other foot in (written and spoken) language practice to understand and create rules for the language in question, otherwise the performance of the algorithm will be compromised.

In the development of rules for the African languages, the main challenges surrounded the native constructs of the languages. Although the indigenous languages are widely spoken throughout the country, people often use English to express the number-based semiotic classes. Consequently, people seldom know how to express these properly in their own languages, which leads to difficulty in sourcing the correct standard of verbalisations when developing the rules.

Despite the good performance of this initial version of the text normalisation component in the Speect/Qfrency TTS system, it still has a big limitation. We do not yet account for the context around the non-standard words when they are verbalised. This has a detrimental effect on the resulting meaning in the African languages when the expansion is substituted back into the original sentence. To illustrate in Xitsonga, the year 1652 will be expanded as "hi lembe ra gidi dzanatsevu na khumentlhanumbirhi". However, suppose it were used in a proper Xitsonga sentence, "Ndzi khandziyile xihahampfuka hi lembe ra 1652". If we substitute the expansion above, we obtain the following boundary problem, "Ndzi khandziyile xihahampfuka hi lembe **ra** *hi lembe ra gidi dzanatsevu na khumentlhanumbirhi*". Future work will investigate Grammatical Framework (GF) [13] as a mechanism that can model the morphological and contextual behaviour successfully.

References

[1] R. Sproat, A. W. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards, "Normalization of non-standard words," *Computer Speech and Language*, vol. 15, no. 3, pp. 287–333, Jul. 2001. [Online]. Available: http://dx.doi.org/10.1006/csla.2001.0169

[2] P. Taylor, *Text-to-Speech Synthesis*, 1st ed. Cambridge University Press, 2009.

[3] P. Ebden and R. Sproat, "The kestrel tts text normalization system," *Natural Language Engineering*, vol. 21, no. 03, pp. 333–353, 2015. [Online]. Available: https://github.com/google/sparrowhawk/tree/master/src

[4] D. Kini and S. Gulwani, "Flashnormalize: Programming by examples for text normalization," in *Proceedings of the 24th International Conference on Artificial Intelligence*, 2015.

[5] K. Gorman and R. Sproat, "Minimally supervised number normalization," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 507–519, 2016. [Online]. Available: https://www.transacl.org/ojs/index.php/tacl/article/view/897/213

[6] K. Wu, K. Gorman, and R. Sproat, "Minimally supervised written-to-spoken text normalization," *CoRR*, vol. abs/1609.06649, 2016. [Online]. Available: http://arxiv.org/abs/1609.06649

[7] R. Sproat and N. Jaitly, "RNN approaches to text normalization: A challenge," *CoRR*, vol. abs/1611.00068, 2016. [Online]. Available: http://arxiv.org/abs/1611.00068

[8] Google, "Text normalisation challenge on Kaggle," 2016. [Online]. Available: https://www.kaggle.com/google-nlu/text-normalization

[9] ——, "Text normalisation challenge on Kaggle," 2017. [Online]. Available: https://www.kaggle.com/c/text-normalization-challenge-english-language

[10] J. A. Louw, A. Moodley, and A. Govender, "The speect text-to-speech entry for the blizzard challenge 2016," in *Proceedings of The Blizzard Challenge 2016 Workshop*, 2016.

[11] R. Gillam, "A rule-based approach to number spellout," *Unicode Consortium*, 1998. [Online]. Available: http://site.icu-project.org/

[12] Python, "Difflib module," 2017. [Online]. Available: https://docs.python.org/2/library/difflib.html

[13] A. Ranta, *Grammatical Framework: Programming with Multilingual Grammars.* Stanford: CSLI Publications, 2011, iSBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).