ISMLA Project: Malayalam Glosser

Thora Daneyko

March 19, 2018

Abstract

Miau

1 Introduction

Bla bla bla test മലയാളി bla bla.

2 Malayalam Language Processing

2.1 About Malayalam

Malayalam is a Dravidian language spoken by over 30 million people in the southern Indian state Kerala. Like most Dravidian languages, Malayalam has a very free SOV word order and a rich agglutinative exclusively suffixing morphology. The verbal morphology is especially complex, as verbs can be marked for various tenses, aspects and moods and may be chained together into long compounds to express subtle differences in meaning (Asher and Kumari 1997).

2.2 NLP challenges

2.2.1 Parsing the Malayalam script

Malayalam is written in Malayalam script, an abugida descended from the Brahmi script. The basic characters represent a syllable composed of a consonant and the inherent vowel /a/. The inherent vowel can be changed by attaching a vowel diacritic to the base character. Hence, the symbol ക represents the syllable /ka/, but with the diacritic for /i/ or /ē/ it becomes കി /ki/ or കേ /kē/. Similarly, the inherent vowel may be deleted using the diacritic that is known as *candrakkala* 'half moon' in Malayalam and *virama* or *halant* in many other Indic languages to represent a consonant without vowel, as in ക് /k/, or to type consonant clusters, as in ക്ഷ /kṣa/ (usually displayed as the ligature ക്ഷ). In Malayalam, however, the *candrakkala* has a phonetic value of its own at the end of a word, often transcribed as a short close or mid unrounded vowel ([ɨ] or [ə]), as in കാട് *kāṭŭ* 'forest' being pronounced [kaːɖɨ], not [kaːɭ],

with intervocalic voicing applying just as between any other two vowels. The *candrakkala* therefore serves two quite different purposes. The only consonants that can appear at the end of a word without being followed by the *candrakkala* vowel are /m/, /n/, /ɳ/, /l/, /ɭ/ and /r/. For this reason, Malayalam has its own characters for these sounds without the inherent vowel (except /m/, which is represented by the *anusvāraṁ* diacritic ം), called *cillu*: ൻ, ൺ, ൽ, ൾ and ർ.

Each base character and diacritic has its own Unicode code point (The Unicode Consortium 2007, p. 334ff). Hence, the syllable ക /ka/ consists of one, കി /ki/ of two and ക് /k/ also of two code points. A simple one-to-one mapping on Latin characters is therefore not possible. Vowel diacritics which are visually composed of two others, but denote a single vowel, also have their own code points. For example the diacritic for /o/ ൊ (as in കൊ /ko/) is not a sequence of /e/ െ (as in കെ /ke/) and /ā/ ാ (as in കാ /kā/), but a single, independent code point (The Unicode Consortium 2007, p. 334f). However, the sequence base glyph + /o/ is visually indistinguishable of base glyph + /e/ + /ā/ in most fonts, so both variants can be observed in Malayalam texts. The *cillu*s now have their own code points as well (The Unicode Consortium 2008), however, before Unicode 5.1, these were typed as base glyph + *candrakkala* + zero-width joiner (U+200D) (The Unicode Consortium 2007, p. 336f), remnants of which are also still commonly present in Malayalam texts on the web.

Conversion from Malayalam script into some other format therefore holds a few difficulties that one must be aware of. However, converting Malayalam script into some alphabetic representation is an important preprocessing step for morpheme splitting, since Malayalam morphemes are not necessarily syllabic and can therefore only hardly be represented and analyzed in the Malayalam script.

2.2.2 Tokenization

The Malayalam script generally separates words by whitespaces, just like the Latin script. However, there is a strong tendency to merge adjacent words in writing. Thus, the two-word sentence ടീച്ചർ ആണ് *ṭīccar āṇŭ* 'is a teacher' may also be written as a single word: ടീച്ചറാണ് *ṭīccaṟāṇŭ*. This may include any number of words from any part of speech and does not only occur in literature, as in (1), but also in everyday speech and writing, as in (2). Bindu and Idicula (2011) estimate that "80-85% of words in Malayalam text documents are compound words", though it is not entirely clear whether they only refer to the merging of independent words or also to suffixing.

(1) മേഘം പോലെ കറുപ്പുനിറഞ്ഞോടുകൂടിയവർ ആണ്.
Mēghaṁ pōle kaṟuppuniṟaññōṭukūṭiyavar āṇŭ.

മേഘം	പോലെ	കറുപ്പ്	നിറഞ്ഞോട്	കൂടി	അവർ	ആണ്
mēghaṁ	*pōle*	*kaṟuppŭ*	*niṟañ-ñ-ōṭŭ*	*kūṭi*	*avar*	*āṇŭ*
cloud	like	black	be.full-PST.PART-SOC	with	they	COP

'They are black like clouds.' (Vēṇugōpālan 2009, p. 179)

(2) അതിന് നിനക്കെന്താ?
Atinŭ ninakkentā?

അതിന്	നിനക്ക്	എന്ത്	ആണ്
at-inŭ	*nin-akkŭ*	*entŭ*	*āṇŭ*
that-DAT	you-DAT	what	COP

'Why do you care?' (Moag 1994, p. 165)

The above examples already indicate that even on the phonetic level this process is not always as simple as in ടീച്ചറാണ് *ṭīccaṟāṇŭ*, where the two words are just merged together. The changes that the affected words undergo when written as one are referred to as *external sandhi* (Devadath et al. 2014). Its counterpart, *internal sandhi*, describes the changes that occur when bound morphemes, such as case endings, are added to a stem. However, these rules are often specific to the suffix in question. The most common *external sandhi* rules that regularly apply when merging arbitrary words in a sentence are the following:

- Insertion of a glide between two vowels (/y/ or /v/ depending on the roundedness of the first vowel), as in (1) കൂടിയവർ *kūṭiyavar* (കൂടി *kūṭi* + അവർ *avar*).
- Dropping of the *candrakkala* vowel when merging with a word starting with a vowel, as in (2) നിനക്കെന്താ(ണ്) *ninakkentā(ṇŭ)* (നിനക്ക് *ninakkŭ* + എന്ത് *entŭ* + ആണ് *āṇŭ*).
- The *candrakkala* vowel becoming /u/ when merging with a word starting with a consonant, as in (1) കുറുപ്പുനിറഞ്ഞോടുകൂടി *kuṟuppuniṟañño̱ṭukūṭi* (കുറുപ്പ് *kuṟuppŭ* + നിറഞ്ഞോട് *niṟañño̱ṭŭ* + കൂടി *kūṭi*).
- Doubling of an initial consonant (especially plosives) when preceded by a vowel or *cillu* consonant. This is very frequent in compounds, such as അരിപ്പെട്ടി *arippeṭṭi* 'rice box' (അരി *ari* + പെട്ടി *peṭṭi*) or പാൽക്കുപ്പി *pālkkuppi* 'milk bottle' (പാൽ *pāl* + കുപ്പി *kuppi*) (Asher and Kumari 1997, p. 397). It also occurs in chains of verbs, e.g. when merging the verb കൊടുക്കുക *koṭukkuka* 'to give' with the past tense form of the verb പെടുക *peṭuka* 'to fall into' to create the passive expression കൊടുക്കപ്പെട്ടു *koṭukkappeṭṭu* 'was given' (കൊടുക്ക *koṭukka* + പെട്ടു *peṭṭu*) (Asher and Kumari 1997, p. 269).
- (Orthographic change only:) The *cillu*s and the *anusvārami* becoming their full counterparts before a vowel, as in സുഖമാണോ? *sukhamāṇō?* 'how are you/are you well?' (സുഖം *sukham* + ആണോ *āṇō*) (Moag 1994, p. 30).
- Dropping of the *anusvārami* before a consonant, as in പുസ്തകപ്രേമം *pustakaprēmami* 'love of books' (പുസ്തകം *pustakam* + പ്രേമം *prēmami*) (Asher and Kumari 1997, p. 398).

For a Malayalam tokenizer, it is therefore not sufficient to extract tokens separated by whitespaces and punctuation, it must also be able to identify and split merged words and reverse the *sandhi* that has altered the participating tokens.

2.2.3 Morphological analysis

Malayalam is a highly agglutinative language and even individual tokens can get quite long under the load of multiple inflectional endings. Luckily, Malayalam is exclusively suffixing, so once the individual words of a sentence have been identified, each of them will always begin with the root or stem and optionally end in a sequence of suffixes. Also, apart from the *internal sandhi* operating at morpheme boundaries, Malayalam grammar is very regular.

Malayalam's core vocabulary mainly consists of nouns and verbs. It only has a handful of non-derived adjectives, while all other adjective-like words have been derived from verb phrases. Also, adjectives do not have any inflections of their own; instead, they are usually nominalized (Asher and Kumari 1997, p. 349ff). Nouns are only marked for number and case, of which Malayalam has seven.

Verbs, on the other hand, display a rather rich and complex morphology. They can have up to three causatives, passive voice and various aspects, moods and tenses. (3) is an example of a heavily inflected Malayalam verb.

(3) പാഠങ്ങൾ പഠിപ്പിക്കപ്പെട്ടുകൊണ്ടിരുന്നിട്ടുണ്ടാകണം.
pāṭhaṅṅaḷ paṭhippikkappeṭṭukoṇṭirunniṭṭuṇṭākaṇaṁ.

pāṭhaṅ -ṅaḷ paṭhi -ppi -kka -ppeṭ -ṭu -koṇṭ -irunn -iṭṭ -uṇṭāk
lesson -PL learn -CAU -CAU -PASS -PST -PROG -PST -PERF -be
-aṇaṁ
-DES.PRS

'Lessons must have been being taught.' (Asher and Kumari 1997, p. 304)

Even though most verbs that actually occur in texts come with a much smaller number of suffixes, every verb will be inflected somehow, and the possibilities are vast. As Asher and Kumari (1997) note, it seems that "all morphological combinations are possible that are semantically interpretable and compatible" (p. 304). Also, Malayalam has a tendency to chain verbs to express even more subtle semantic differences, so a typical Malayalam sentences will often contain multiple verbs. As mentioned above, almost all adjectives are actually adjectivized verbs, which may be inflected as well.

When processing Malayalam morphology, one can take advantage of the fact that Malayalam exclusively uses suffixes which are also rather regular. However, one must also beachten the *internal sandhi* between suffixes which is sometimes peculiar for a certain suffix. For verbs, the amount of possible combinations of suffixes is huge, which is a besondere hindrance for paradigm generation.

3 Previous work

In their 2011 paper on automatic machine translation between Malayalam and Tamil, Jayan, Rajeev, and Rajendran draw a pessimistic conclusion regarding

Malayalam morphological analysis: "A sandhi splitter demands a morphological analyzer and a morphological analyzer demands a sandhi splitter. There is a dead lock between the two." While it is true that there is a certain dependency between resolving sandhi-merged words into individual tokens and analyzing the morphology of these tokens, many researchers following Jayan, Rajeev, and Rajendran (2011) have now overcome the "dead lock" and found successful ways to perform sandhi splitting and morphological analysis separately.

3.1 Sandhi splitting

The importance of sandhi splitting for the processing of Dravidian languages and especially Malayalam has recently been recognized and addressed by several researchers. Devadath et al. (2014) note that "[s]andhi acts as a bottle-neck for all term distribution based approaches for any NLP and IR task". The developed applications serve as preprocessors for POS Taggers (Manju, Soumya, and Idicula 2009; Bindu and Idicula 2011), Parsers (Devadath 2016) and Morphological Analyzers (Sebastian and Kumar 2018). Further Anwendungsgebiete for sandhi splitters are "document indexing and topic modeling" (Nisha and Raj 2016) and machine translation (Jayan, Rajeev, and Rajendran 2011).

Manju, Soumya, and Idicula (2009) and Bindu and Idicula (2011) use a dictionary lookup approach for sandhi splitting. They maintain a lexicon of Malayalam words and recursively search for the longest known substring in an input string. For each possible substring, they also reverse any sandhi rule that might have applied and thus generate a number of forms to look up. Since their sandhi splitters are only a preprocessing step for their POS Taggers, they do not report any performance measures.

Statistical methods for sandhi splitting are much more popular that rule based methods. Devadath et al. (2014) explore a hybrid approach where they first determine the split points statistically relying on n-gram frequencies and then modify the identified tokens using predefined sandhi rules. Their system reaches an accuracy of 91.1 % (meaning words that were split exactly as in the gold standard).

Kuncham et al. (2015) develop a purely statistical language independent sandhi splitter which they evaluate on Telugu and Malayalam. The train a Conditional Random Fields model to identify split points and applicable sandhi rules based on the characters of the word and surrounding segments to resolve ambiguous splits and sandhi processes. They reach an accuracy of 89.07 % for Telugu and 90.50 % for Malayalam.

Nisha and Raj (2016) employ Memory Based Language Processing to create a sandhi splitter and morphological analyzer for Malayalam. Their system divides words in the training corpus into a root and suffix part and matches unseen data against the already encountered suffixes, finding the closest match using a distance measure. They report an accuracy of 90 %.

Machine learning is by far the preferred method for building a sandhi splitter and the systems reach a high accuracy. However, while token merging and

sandhi processes are frequent in Malayalam, the involved sandhi rules are rather few and usually very simple. Collecting large training sets and building complex statistical models seems exaggerated for this task. Since Malayalam (external) sandhi is either simple insertion or only affects the final characters of the preceding word and leaves the following word untouched, a recursive lookup strategy from right to left, as employed by Manju, Soumya, and Idicula (2009) and Bindu and Idicula (2011), seems to be fitting the task quite nicely. Of course, this requires a large dictionary that either contains all possible inflected forms or comes with a morphological analyzer, and will also fail on unknown words or forms. The big advantage of the statistical models here is that they easily generalize to unseen data.

3.2 Morphological analysis

Bla bla

Rajeev, Rajendran, and Sherly (2007) and Jayan, Rajeev, and Rajendran (2011) make use of Malayalam's suffixing nature and employ a suffix stripping method: On a tokenized sentence, they recursively remove recognized suffixes from the word, paying attention to sandhi processes, until the remaining stem can be found in a dictionary. For Malayalam, this method is very effective, but requires a predefined set of suffixes and a large dictionary of stems. Also, it is not generalizable to languages with prefixes or a non-agglutinative morphology.

Manju, Soumya, and Idicula (2009) take a less specialized direction by parsing and analyzing Malayalam words using a Finite State Transducer (FST). FSTs have long proven to be very suitable for morphological analysis, especially of agglutinative languages (Beesley and Karttunen 2003). They are also very fast, producing an analysis in the time that is needed to read the input string once. However, FSTs can quickly get very complex and they require hand-crafted rules for recognizing the individual morphemes, just as the suffix stripping method.

In contrast to these manual methods, Sebastian and Kumar (2018) employ a machine learning approach to their Malayalam morphological analyzer. They train a Naive Bayes classifier on a split point and sandhi rule annotated data set. The overall performance of their system is not entirely clear, as they only provide accuracy measures for words ending in *-yalla* (negation), *-yuṭe* (genitive case), *-yāṇŭ* and *-yāyi* (two forms of *ākuka* 'to be', actually cases of *external sandhi*), only covering one rather predictable type of *sandhi* (glide insertion). For these four examples, their analyzer recognizes and applies 92.06 % of the desired splits.

4 The Malayalam Glosser

Bla bla

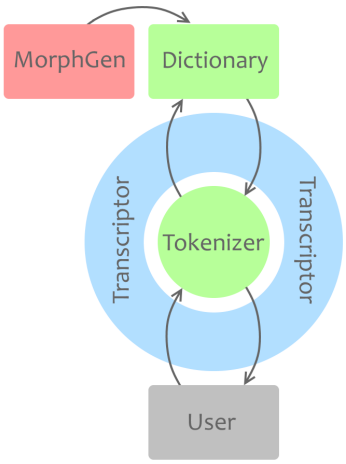

Figure 1: The architecture of the Malayalam Glosser.

4.1 Transliteration

Due to its syllabic nature, morpheme splitting is a tedious task in the Malayalam script and is best carried out in an alphabetic transcription. Also, while the Malayalam script has been included in Unicode for quite some time and Malayalam keyboard layouts are preinstalled on most modern machines, even native speakers of Malayalam frequently type Malayalam in a Latin romanization. Language learners, the primary target audience of the Malayalam Glosser, often do not know how to use the Malayalam script on a computer, especially when they are beginners. It is therefore necessary to convert Malayalam text between the script and various romanizations to be able to support the most popular input formats and display the finished glosses in a readable way.

4.1.1 Supported Scripts

Apart from the Malayalam script, the Malayalam Glosser currently supports two additional romanization schemes, Mozhi and ISO-15919. The Mozhi romanization is very popular especially among Malayalis to write Malayalam on the web. It consists only of ASCII characters and utilizes capitalization to enlarge the set of available characters (Cibu 2008). The ISO-15919 or National Library at Kolkata romanization is the default romanization scheme in scientific texts for all Indic languages (it is also the one used in this paper). It makes heavy use of diacritics and is thus not easily typable on the average English keyboard. Because of this, there also is an ASCII version of ISO-15919 which replaces the diacritics by punctuation characters (ISO 2001). Both variants of ISO-15919, Unicode and ASCII based, are supported by the Malayalam Glosser.

A full table with all Malayalam characters in the different scripts (Malayalam script, ISO-15919 Unicode, ISO-15919 ASCII and Mozhi) can be found in the appendix. (4) is an example of how the sentence ‘all human beings are born free

and equal in dignity and rights' is spelled in the four supported scripts (Ager 2011).

(4) **Malayalam:** മനുഷ്യരെല്ലാവരും തുല്യാവകാശങ്ങളോടും അന്തസ്സോടും സ്വാതന്ത്ര്യത്തോടുംകൂടി ജനിച്ചവരാണ്.

ISO-15919 Unicode: manuṣyarellāvarum̐ tulyāvakāśaṅṅaḷōṭum̐ antassōṭum̐ svātantryattōṭum̐kūṭi janiccavarāṇŭ.

ISO-15919 ASCII: manu.syarellaavaru;m tulyaavakaa;sa;n;na.loo.tu;m antassoo.tu;m svaatantryattoo.tu;mkuu.ti janiccavaraa.nˆu.

Mozhi: manushyarellaavarum thulyaavakaaSangngaLOTum anthassOTum svaathanthryaththOTumkuuTi janichchavaraaN~.

The ISO-15919 ASCII romanization is also the underlying representation of all dictionary entries, since the Malayalam rules of the `MorphGen` are written in this format.

4.1.2 Transliterators

The main class handling all transliterations between the different scripts in the `MalayalamTranscriptor`. However, it mostly serves as an interface to the transliterator system designed for the NorthEuraLex database by the EVOLAEMP project (Jäger and Dellert 2017). To display phonetic transcriptions for the lexical entries in their database, they developed automatic rule-based transliterators that are able to convert from orthography to IPA based on language-specific rule sets (Daneyko 2016). Since phonetic transcription is just another type of transliteration, the same infrastructure can also be used to convert between different romanization schemes. The EVOLAEMP transliterators also have an efficient FST based implementation, the Java version of which is quite platform-dependent, so the basic Java implementation (called 'simple transliterators' in Daneyko (2016)) was used in a slightly altered form.

Transliterator rules from Malayalam script to ISO-15919 Unicode and from ISO-15919 Unicode to IPA were already written for the NorthEuraLex database. Hence, ISO-15919 Unicode was selected as the intermediate representation for the transliterators and additional rules were written for ISO-15919 Unicode to Malayalam script, ISO-15919 Unicode from and to ISO-15919 ASCII, and ISO-15919 Unicode from and to Mozhi.

4.2 Morphological generation

MorphGen

4.2.1 File format

`MorphGen` requires one file containing the rules for generating the inflected words from a given gloss. The rule file is a simple text file, with one rule per line, input

```
[*]^u|DAT       [1]|in^u
[*]u|DAT        [1]u|vin^u
[*];m|DAT       [1]tt|in^u
[*][!. _]l|DAT  [1][2]l|kk^u
[*][!l r]|DAT   [1][2]|in^u
[*]n|DAT        [1]n|^u
[*]|DAT         [1]|kk^u
```

Figure 2: The dative rules from the Malayalam `MorphGen` rule set.

and output side of each rule separated by a tab stop. Figure 2 shows an excerpt from the Malayalam rule file.

Since some scripts, notably the ISO-15919 ASCII romanization for Malayalam, may use the `-` and `.` characters that are usually displayed in glosses, `MorphGen` operates on `|` and `&` instead. Hence, the gloss 'mouse.PL-GEN' would be written `mouse&PL|GEN` in the `MorphGen` format. For infixes, `<>` is used (e.g. `mouse<>PL|GEN`).

A couple of special characters can be used inside rules for easier matching:

- `[*]` is a wildcard matching any number (including none) of characters. The matched characters can be referred to on the output side by an integer corresponding to the position of the wildcard on the left side. Applying the rule `[*]x[*]y[*]  x[3][2]z[2]` to the input `aaxbyccc`, for example, would assign `aa` to 1, `b` to 2 and `ccc` to 3 and hence produce the output `xcccbzb`. These wildcards can also be named and referred to by their name on the right side, as in `[*]x[name]y[*]  x[2][name]z[name]`. Note that these names do not increase the counter for the wildcard labels: The variable previously referred to as 3 on the right side is now labeled 2.
- By default, `MorphGen` inserts wildcards at the beginning and end of the string if not present and matches them once at the beginning and end on the right side. The rule `a  b` gets translated to `[*]a[*]  [1]b[2]`, for example. To prevent this, word boundaries may explicitly be matched by a hash tag `#` on the left side. Thus, the rule `a#  b` will be converted to `[*]a  [1]b`, matching only `a`s at the end of a string.
- A frequently recurring group of characters to match can be defined on top of the file using the keyword `#def` followed by the variable name followed by the group contents in square brackets, as in `#def  #V  [aa ai au ee ii oo uu a e i o u]` which define the set of vowels in Malayalam. Note that the `#def` keyword, the group name and the group definition are tab separated, while the strings inside the group definition are separated by whitespaces. The name of a group variable must always begin with a hash tag `#` to distinguish it from the named wildcards. These groups can be referenced on the left side of a rule with `[#name]` and on the right side with their integer label just like wildcards, as in `[*][#V]t[#V][*]  [1][2]d[3][4]`.
- Ad-hoc groups for a single rule may be created with `[!item1 item2 ...]`, as in the example in Figure 2.

- An optional group, i.e. a group matching one or none of the contained characters can be introduced with `[?item1 item2 ...]`. Predefined groups may also be optionalized by referring to them with `[?#name]`. Consider for example the rule `#[?#C][#V].tuka|PST [1][2].t|.tu` used to produce the past tense form of Malayalam verbs of the (C)Vṭuka. The optionally matched initial consonant is reprinted in the `[1]` position on the right side only if it was actually found.
- Sometimes the realization of same form may differ between words. For instance, the past tense of the verb വിൽക്കുക *vilkkuka* 'to sell' is *vi͟r͟ru*, while that of നിൽക്കുക *nilkkuka* 'to stand' is *ninnu*. The phonological cues for selecting the appropriate past tense form have long been lost on these verbs, hence to get a complete paradigm, we may want to generate both forms. Multiple output sides for a single input side are separated by `||`, as in this past tense rule for verbs ending in *-lkkuka*: `[*]lkkuka|PST [1]_r|_ru || [1]n|nu`.

`MorphGen` optionally requires a second file specifying the templates for paradigm generation (see section 4.2.3). In this file, the possible inflections for each part of speech and the order in which they may occur are defined in a regular expression-like notation. This is the specification for Malayalam nouns:

```
[n] PL (NOM || ACC || DAT || GEN || SOC || INS || LOC)
```

This means that a Malayalam word labeled with the part of speech tag `n` can optionally have the feature `PL`, optionally followed by any of the case features. It spells out as: `n`, `n PL`, `n PL NOM`, `n NOM`, `n PL ACC`, etc. A whitespace is used to separate two features or feature groups that optionally occur in this order. `||` means 'or'. The whitespace takes precedence over the 'or' operator, hence the case labels in the above have to be grouped together by parentheses.

This is part of the specification for Malayalam verbs (the actual one is much larger and more complex):

```
[v] PASS (((PRS || PST) (Nn || Nm || Nf) || PST_STAT) (NEG || A)
```

Here, a verb can (optionally) take the passive (`PASS`). This may (optionally) be followed by either the present/past tense (`PRS || PST`) and (optionally) a nominalizer (`Nn || Nm || Nf`), or the past tense obligatorily followed by the stative perfect marker *iṭṭŭ* (`PST_STAT`). Finally, the verb can (optionally) either be negated or adjectivized (`NEG || A`). Note that the underscore `_` is used to delete the optionality of the whitespace and force the two features to occur together. The above rule will produce `v` and `v PST STAT` (and `v PST` due to the earlier mentioning of `PST`), but not `v STAT`.

4.2.2 Automated inflection

4.2.3 Paradigm generation

4.3 Dictionary lookup

MalayalamDictionary

4.3.1 Efficiency considerations

Considering that the dictionary may be very large and that the main function of the Glosser is to look words up in this dictionary, being able to load and query it very quickly is essential for the performance of the Glosser. Hence, I experimented with a few alternatives for storing the dictionary data and investigated their efficiency. The tests elaborated below are not very exact or well-designed and were only meant to quickly assess the usefulness of the considered methods.

`HashMap` vs. `ReverseTrie`

The straightforward way to represent a dictionary as a Java object is a `HashMap`. Apart from being readily available and easy to use, querying a `HashMap` is fast. However, this also means that all entries are stored as their complete `String` representation, which may consume quite a lot of space. Considering that the inflected forms of the words share most of their characters, a trie representation seemed quite suitable and might be able to save space compared to a simple `HashMap`. Since Malayalam is exclusively suffixing, I programmed a `ReverseTrie` which reads and retrieves the strings from last to first character, in order to save as much space as possible. A useful side effect of this is that the tokenizer does not need to look up all suffixes of a compound word in the dictionary, but can simply do a suffix search of the `ReverseTrie` to get the longest contained suffix.

In order to compare the performance of a `HashMap` and `ReverseTrie` based dictionary, I measured the memory used by the program before loading the dictionary data and after creating the `HashMap` and Trie (calculated as `Runtime.totalMemory() - Runtime.freeMemory()` after a `System.gc()` call). Then I let the dictionary find the longest known suffix of the test String *aviṭeyuḷḷataṟiññu* (*aviṭe uḷḷatŭ aṟiññu* "knew (he) was there") 1,000,000 times and measured the time needed by a MashMap and `ReverseTrie` based dictionary (calculated using `System.currentmillis()`). Finally, I rewrote the tokenizer to also work with a `ReverseTrie` and tested how long tokenization of the short conversation from lesson 11 of Moag (1994, p.164f) took it with the two dictionary types.

Despite the many shared suffixes, the `HashMap` was smaller than the `ReverseTrie`, taking up 8,318,164.8 bytes on average during five test runs, while the Trie required 12,590,051.2 bytes. However, the memory used by the `HashMap` varied greatly, ranging from only 5,160,456 to 9,801,392 bytes, while the Trie always consumed almost exactly the same amount of memory. This indicates that the measurements might have been distorted by background processes such as the garbage collection. However, the `HashMap` still seems to be considerably smaller.

As expected, the `ReverseTrie` outperformed the `HashMap` on the looped suffix search of *aviṭeyuḷḷataṟiññu*. The Map took an average of 999 milliseconds during five test runs, while the Trie only needed 312.4 ms. However, the performance of the Trie was very unstable, ranging from 140 to 518 ms between runs, while

the `HashMap` always needed between 908 and 1049 ms, which is still much slower than the slowest suffix search of the Trie.

On a real Malayalam text, where only few words are long compounds such as *aviṭeyuḷḷataṟiññu*, both methods were equally fast. During 10 glossings of the Moag conversation, the Map based tokenization took 156.3 ms on average and the Trie based tokenization 161.7 ms. Both ran very stable.

All in all, the `HashMap` seems to be the better choice, since it is smaller than the Trie and equally fast on normal Malayalam texts. The Trie is faster when tokenizing long compound words, which however are not frequent enough to justify preferring it over the `HashMap`.

File storage vs. Serialization

Loading the dictionary data into the underlying `HashMap` (or `ReverseTrie`) takes a considerable amount of time at launch. Hence, I considered serializing the Map or Trie object to be able to load it quicker. Since the Java serialization is known to rather slow, I used the FST Fast Serialization library for my tests. I first read the dictionary data from the text file and created the `HashMap` and `ReverseTrie` from it, measuring the time needed. Then I serialized the two objects and took the time required to deserialize them.

During five test runs, parsing the text file into an object took 278.2 ms on average for the `HashMap` and 310.6 ms for the Trie. Deserializing the same objects required 563.4 ms on average for the `HashMap` and 339.8 ms for the Trie. Loading the data from a text file is thus faster than deserializing a previously created object.

The file storing the serialized `ReverseTrie` was twice as large as the file with the `HashMap`. This confirms my assertions from the previous section that the Trie takes more space than the `HashMap`.

4.4 Tokenization

MalayalamGlosser

5 Conclusion

References

Ager, Simon (2011). *Malayalam (മലയാളം)*. URL: `https://www.omniglot.com/writing/malayalam.htm` (visited on 03/19/2018).

Asher, Ronald E. and T. C. Kumari (1997). *Malayalam*. Psychology Press.

Beesley, Kenneth R. and Lauri Karttunen (2003). *Finite state morphology*. Center for the Study of Language and Information.

Bindu, M. S. and Sumam Mary Idicula (2011). "High Order Conditional Random Field Based Part of Speech Tagger and Ambiguity Resolver for Malayalam – a Highly Agglutinative Language." In: *International Journal of Advanced Research in Computer Science* 2.5.

Cibu, C. J. (2008). *Mozhi – Detailed specification*. URL: `https://sites.google.com/site/cibu/mozhi/mozhi2` (visited on 03/18/2018).

Daneyko, Thora (2016). *Using finite state transducers for multilingual rule-based phonetic transcription*. BA thesis. University of Tübingen.

Devadath, V. V. (2016). "A Shallow Parser for Malayalam." MA thesis. Hyderabad: International Institute of Information Technology.

Devadath, V. V., Litton J. Kurisinkel, Dipti Misra Sharma, and Vasudeva Varma (2014). "A Sandhi Splitter for Malayalam." In: *Proceedings of the 11th International Conference on Natural Language Processing*, pp. 156–161.

ISO (2001). *ISO 15919. Information and documentation – Transliteration of Devanagari and related Indic scripts into Latin characters*.

Jäger, Gerhard and Johannes Dellert, eds. (2017). *NorthEuraLex (version 0.9)*. URL: `http://northeuralex.org/`.

Jayan, Jisha P., R. R. Rajeev, and S. Rajendran (2011). "Morphological Analyser and Morphological Generator for Malayalam-Tamil Machine Translation." In: *International Journal of Computer Applications* 13.8, pp. 15–18.

Kuncham, Prathyusha, Kovida Nelakuditi, Sneha Nallani, and Radhika Mamidi (2015). "Statistical Sandhi Splitter for Agglutinative Languages." In: *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, pp. 164–172.

Manju, K., S. Soumya, and Sumam Mary Idicula (2009). "Development of a POS tagger for Malayalam – An Experience." In: *Advances in Recent Technologies in Communication and Computing*. IEEE, pp. 709–713.

Moag, Rodney F. (1994). *Malayalam: A University Course and Reference Grammar*. Austin: University of Texas, Center for Asian Studies.

Nisha, M. and P. C. Reghu Raj (2016). "Sandhi Splitter for Malayalam Using MBLP Approach." In: *Procedia Technology* 24, pp. 1522–1527.

Rajeev, R. R., N. Rajendran, and Elizabeth Sherly (2007). "Morph analyser for malayalam language: A suffix stripping approach." In: Proceedings of 20th Kerala Science Congress.

Sebastian, Mary Priya and G. Santhosh Kumar (2018). "Machine Learning Approach to Suffix Separation on a Sandhi Rule Annotated Malayalam Data Set." In: *Language in India* 18.1, pp. 361–382.

The Unicode Consortium (2007). *The Unicode Standard, Version 5.0*. Boston: Addison-Wesley.

– (2008). *Unicode 5.1.0*. URL: `http://www.unicode.org/versions/Unicode5.1.0/`.

Vēṇugōpālan, P., ed. (2009). *Āścaryacūḍāmaṇi. Sampūrṇṇamāya āṭṭaprakāravum kramadīpikayum*. Tiruvanantapuram: Mārgi.

A Abbreviations used in glosses

CAU	Causative	PL	Plural number
COP	The copula *āṇŭ*	PROG	Progressive aspect
DAT	Dative case	PRS	Present tense
DES	Desiderative mood	PST	Past tense
PASS	Passive voice	PART	Participle
PERF	Perfect aspect	SOC	Sociative case

B Transcription schemes

Script	**ISO (Uni)**	**ISO (ASCII)**	**Mozhi**
അ	a	a	a
ആ	ā	aa	aa
ഇ	i	i	i
ഈ	ī	ii	ii
ഉ	u	u	u
ഊ	ū	uu	uu
ഋ	r̥	,r	R
എ	e	e	e
ഏ	ē	ee	E
ഐ	ai	ai	ai
ഒ	o	o	o
ഓ	ō	oo	O
ഔ	au	au	au
അം	aṁ	a;m	am
അഃ	aḥ	a.h	ah
ക	ka	ka	ka
ഖ	kha	kha	kha
ഗ	ga	ga	ga
ഘ	gha	gha	gha
ങ	ṅa	;na	nga
ച	ca	ca	cha
ഛ	cha	cha	chha
ജ	ja	ja	ja
ഝ	jha	jha	jha
ഞ	ña	~na	nja
ട	ṭa	.ta	Ta
ഠ	ṭha	.tha	Tha
ഡ	ḍa	.da	Da
ഢ	ḍha	.dha	Dha
ണ	ṇa	.na	Na
ത	ta	ta	tha
ഥ	tha	tha	thha
ദ	da	da	da

ധ	dha	dha	dha
ന	na	na	na
പ	pa	pa	pa
ഫ	pha	pha	pha
ബ	ba	ba	ba
ഭ	bha	bha	bha
മ	ma	ma	ma
യ	ya	ya	ya
ര	ra	ra	ra
ല	la	la	la
വ	va	va	va
ശ	śa	;sa	Sa
ഷ	ṣa	.sa	sha
സ	sa	sa	sa
ഹ	ha	ha	ha
ള	ḷa	.la	La
ഴ	ḻa	__la	zha
റ	r̲a	__ra	rra
ൃ	t̲t̲a/r̲r̲a	__t__ta/__r__ra	ta
ന്റ	nt̲a/nr̲a	n__ta/n__ra	nta
ൻ	n	n	n
ൺ	ṇ	.n	N
ർ	r	r	r
ൽ	l	l	l
ൾ	ḷ	.l	L
ൿ	k	k	k
ക്	kŭ	kˆu	k~