Sarrif – The Elegant Arabic Morphology Parser

Suhel Jaber and Rodolfo Delmonte

University Ca' Foscari, Dept. Language Sciences, Laboratory Computational Linguistics,
Ca' Bembo, Dorsoduro 1705, 30123 Venezia
Italy
{jaber,delmont}@unive.it

Abstract

In this paper we present Sarrif, our Arabic Morphology Parser, featuring a novel approach to the description of Arabic morphology with 2-tape finite state transducers, based on a particular and systematic use of the operation of composition in a way that allows for incremental substitutions of concatenated lexical morpheme specifications with their surface realization for non-concatenative processes (the case of Arabic templatic interdigitation and non-templatic circumfixation).
We argue that:

1. the method of incremental substitutions through compositions allows for an elegant description of all main morphological processes present in natural languages including non-concatenative ones in strict finite-state terms, without the need to resort to extensions of any sort;
2. our approach allows for the most logical encoding of every kind of dependency, including traditional long-distance ones (mutual exclusiveness), circumfixations and idiosyncratic root and pattern combinations;
3. a smart usage of composition such as ours allows for the creation of a same system that can be easily accomodated to fulfil the duties of both a stemmer (or lexicon development tool) and a full-fledged lexical transducer.

Introduction

In this paper we present Sarrif, our Arabic Morphology Parser, featuring a novel approach to the description of Arabic morphology with 2-tape finite state transducers, based on a particular and systematic use of the operation of composition in a way that allows for incremental substitutions of concatenated lexical morpheme specifications with their surface realization for non-concatenative processes (the case of Arabic templatic interdigitation and non-templatic circumfixation).

We argue that:

1. the method of incremental substitutions through compositions allows for an elegant description of all main morphological processes present in natural languages including non-concatenative ones in strict finite-state terms, without the need to resort to extensions of any sort;
2. our approach allows for the most logical encoding of every kind of dependency, including traditional long-distance ones (mutual exclusiveness), circumfixations and idiosyncratic root and pattern combinations;
3. a smart usage of composition such as ours allows for the creation of a same system that can be easily accomodated to fulfil the duties of both a stemmer (or lexicon development tool) and a full-fledged lexical transducer.

Preliminaries

In this section we specify only the technical parameters needed by the reader who's already acquainted with the generalities of Arabic language script and grammar and finite state calculus to find his way through our implementation details.
For the unacquainted reader willing to tackle these topics from the beginning we suggest Bohas & Guillaume (1984) as the most exhaustive and detailed account of Arabic word formation rules and transformation processes to date and Beesley & Karttunen (2003) as the best hands-on introductory tutorial to finite state machine techniques applied to the field of morphology.

Morphological Framework

In the examples in this paper we treat Arabic morphology according to the analysis outlined in Harris (1941), that considers Arabic words as the combination of pattern morphemes, root bundle morphemes and affixes. For instance, a word such as اِجْتَمَعَ in this framework is decomposed into

a. root bundle morpheme ج م ع;
b. pattern morpheme اِ ـْـتَـَـ (including placeholders);
c. suffix َ.

In any case, the novel approach to word formation that we present in this paper can be applied to any particular morphological theory.

Buckwalter Transliteration

In regular expressions we use a transliteration system instead of the original Arabic script. We've decided to employ that of Buckwalter (2002) because of its widespread usage in existing implementations and its one-to-one correspondence to the Arabic script.

We give a small fragment of it in Table 1, including only the characters significantly differing from those used in other systems.

Arabic character	ئ	ا	ح	ش	ض	ط	ظ	ع	ء
Buckwalter transliteration	}	A	H	$	D	T	Z	E	o

Table 1: A partial transliteration of Arabic characters using the Buckwalter system

Xerox Finite State Calculus Syntax

The syntax of regular expressions presented in this paper is that of *xfst*, the Xerox Finite State Tool. We give a summary of the relevant operator and symbols in Table 2.

`define variable regular-expression ;`	defines a variable containing a regular expression
`read regex regular-expression ;`	compiles a regular expression and stores it on the stack
`"`	character surrounding sequences that need to be escaped as a single unit
`?`	wildcard
`0`	ε-transition
`*`	0 or more times iteration operator commonly known as "Kleene star"
`|`	union or disjunction operator
`.o.`	composition operator

Table 2: A summary of *xfst* symbols relevant to this paper's examples

Note that in our approach we use a finite state calculus that is classical (as opposed to the Two-Level one of Koskenniemi (1983)) and strict (as opposed to the extended one including algorithms such as those of Beesley & Karttunen (2000), which allow also for the resolution of problems normally exceeding finite-state power), without using the classical intersection operation at all.
For a description of the drawbacks of resorting to the aforementioned techniques for Arabic morphology parsing, see Jaber & Delmonte (2008).

Arabic Morphology Parsing in Sarrif

The Compositional Approach to Morphology

The main insight leading our implementation of Arabic morphology is that every morphological process can be modelled in terms of the composition of regular languages.
We call our approach the "Incremental Substitutions" Compositional Approach.

In the rest of this section we explain this concept by showing all the stages of the process which maps the word نَقْتُلُ among others to its morphological analysis.

Templatic Interdigitation (Idiosyncratic Root and Pattern Combinations)
We now show how to obtain a mapping from the substring قْتُل among others to its analysis as " `Form_I_Impf_Act_u`".

```
define C [' | b | t | v | j | H | x | d
|"*" | r | z | s | "$" | S | D | T | Z | E
| g | f | q | k | l | m | n | h | w | y];

read regex [[q t l | k t b | T r q]
" Form_I_Impf_Act_u"]
.o. [C 0:o C 0:u C " Form_I_Impf_Act_u":0];
```

From an 'analytical' (as opposed to 'generative') point of view we can interpret this last regular relation as a two-phase mapping:

1. `[C 0:o C 0:u C " Form_I_Impf_Act_u":0]` makes it so that the vowels in the Verb Form I Imperfect Active pattern ـْـُـ get 'filtered' in the passage from surface to lexical representation, 'erased' and 'substituted' by the agreeing tag which is in fact concatenated to the end of the remaining lexical material made up of those `[C]` roots which were allowed to 'pass through';

2. the resulting lexical string is 'passed' as an argument to a second regular expression `[[q t l | k t b | T r q] " Form_I_Impf_Act_u"]` by means of composition, which will operate on the remaining material if and only if the tags (in this case only 1) concatenated at the end of the regular expression correspond to those generated in or passed through the previous phase of analysis; in this case all it would do on the remaining material would be constraining its quality to that of the actual root morphemes which are allowed to combine with the pattern represented by the concatenated tag.

Notice that in this case we don't even need to previously define the `[C]` language, even if we did it in the previous example. Indeed the following regular expression denotes exactly the same relation as the previous one.

```
read regex [[q t l | k t b | T r q]
" Form_I_Impf_Act_u"]
.o. [? 0:o ? 0:u ? " Form_I_Impf_Act_u":0];
```

With the following expression we show how it is possible to organize a lot of idiosyncratic root and pattern combinations together in one compact structure:

```
read regex [
[[k t b | q t l] " Form_I_Perf_Act_a"] |
[[D r b | H s b] " Form_I_Perf_Act_i"] |
[["$" r f | H s n] " Form_I_Perf_Act_u"]
] .o. [
[? 0:a ? 0:a ? " Form_I_Perf_Act_a":0] |
[? 0:a ? 0:i ? " Form_I_Perf_Act_i":0] |
```

```
[? 0:a ? 0:u ? " Form_I_Perf_Act_u":0]
];
```

Non-templatic Circumfixation

Let's now have a look at how circumfixation can be efficiently handled through the operation of composition:

```
read regex
[[q t l] " Form_I_Impf_Act_u"
[" 2_Pers_Sing_Fem_Ind_a" |
" 1_Pers_Plur_Ind_a"]] .o.
[? 0:o ? 0:u ? " Form_I_Impf_Act_u":0
[" 2_Pers_Sing_Fem_Ind_a" |
" 1_Pers_Plur_Ind_a"]] .o.
[0:t 0:a ?* 0:i 0:y 0:n 0:a
" 2_Pers_Sing_Fem_Ind_a":0 |
0:n 0:a ?* 0:u " 1_Pers_Plur_Ind_a":0];
```

In `[0:t 0:a ?* 0:i 0:y 0:n 0:a " 2_Pers_Sing_Fem_Ind_a":0 | 0:n 0:a ?* 0:u " 1_Pers_Plur_Ind_a":0]` an arbitrary string (`?*`) surrounded by a given circumfix (i.e. preceded and followed by a given prefix and suffix respectively) is mapped to the same arbitrary string and a tag representing the analysis of the circumfix consumed by the ε-transitions.

Note that other implementations usually deal with certain long-distance dependencies through the use of composition, but in a very different way:

1. all the prefixes, stems and suffixes are concatenated together to form every potential combination (even prohibited ones), and prefixes and suffixes are assigned each a distinctive tag;
2. through the use of composition, patterns featuring mutually exclusive tags are explicitly removed from the network.

Our method, on the other hand, just assigns one tag to each circumfix (for other purposes, moreover) and anyway the correct circumfixation is created in one single process instead of total prefixation plus total suffixation and subsequent pruning.

Summing It All Up

We're now ready to give an interpretation of our "Incremental Substitutions" Compositional Approach from a 'generative'point of view as that of an n-phase mapping:

1. in the first regular expression we enlist in a concatenative way all the morphemes (or rather, their lexical representations) which make up a word, in the order in which we should process their 'merging' with the string we obtain at each phase;
2. in the subsequent regular expressions we process their 'merging' with any intermediate string previously obtained, according to the order of the remaining tags at each point, 'erasing' one tag at a time after its surface counterpart has been created and merged to the rest.

In this way we were able to give a linear rendering of what globally assumes the entity of a hierarchical representation (cfn. 'morphosyntax') or incremental creation of bigger building blocks from already elaborated ones, i.e.:

ق ت ل + ـُـُـ = قُتُل
قُتُل + نَـــــُ = نَقْتُلُ

Using Sarrif as a Stemmer

Sarrif is a flexible implementation. Besides being an elegant parser, it can also work as a stemmer by relaxing the constraints on the allowed root morphemes for each pattern, as in the following regular expression:

```
read regex [
[? ? ? " Form_I_Perf_Act_a"] |
[? ? ? " Form_I_Perf_Act_i"] |
[? ? ? " Form_I_Perf_Act_u"]
] .o. [
[? 0:a ? 0:a ? " Form_I_Perf_Act_a":0] |
[? 0:a ? 0:i ? " Form_I_Perf_Act_i":0] |
[? 0:a ? 0:u ? " Form_I_Perf_Act_u":0]
];
```

By running this kind of machine on an Arabic text input we get an output of all the encountered root bundles classified by the patterns they were found in. This has helped us build our lexicon out of different sources.

Implementation Evaluation

For purposes of evaluation we have written a script composing more than 4700 root morphemes with the verbal patterns they can actually combine with extracted from several databases.

This grammar compiled in real time on an Intel Pentium M 730 1.60 GHz based Microsoft Windows XP system using the Xerox Finite-State Tool version 2.6.2.

Conclusions

In this paper we have presented Sarrif, our Arabic morphology parser featuring an elegant and efficient approach to the encoding of lexical transducers that we have called "Incremental Substitutions" Compositional Approach.

We've given hands-on details on our implementation, exemplifying how most morphological processes and descriptions are actually dealt with by going through some simplified snippets of code.

Moreover, we have designed more than one way our model could be put to practical usage (stemming, field research and lexicon developing, morphological analysis and generation).

Ultimately, we have shown that our model allows for a fair description of Arabic morphology in a strictly finite-state framework without the need to resort to enhancements or extensions of any sort.

References

Beesley, K.R. & Karttunen, L. (2000). Finite-State Non-concatenative Morphotactics. In Proceedings of the Workshop on Finite-State Phonology. 38th Annual Meeting of the Association for Computational Linguistics. Morristown, NJ: Association for Computational Linguistics.

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

Bohas, G. & Guillaume, J.P. (1984). Etude des Théories des Grammairiens Arabes. Damas: Institut Français de Damas.

Buckwalter, T. (2002). Buckwalter Arabic Morphological Analyzer Version 1.0. LDC Catalog Number LDC2002L49. Linguistic Data Consortium.

Harris, Z. (1941). Linguistic Structure of Hebrew. Journal of the American Oriental Society, 62, 143--167.

Jaber, S. & Delmonte, R. (2008). Arabic Morphology Parsing Revisited. In Proceedings of the 9th International Conference on Intelligent Text Processing and Computational Linguistics. Berlin, Heidelberg: Springer.

Koskenniemi, K. (1983). Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. Publication 11. University of Helsinki, Department of General Linguistics, Helsinki.