A Purely Surface-oriented Approach to Handling Arabic Morphology

Yousuf Aboamer and Marcus Kracht

Bielefeld University, Postfach 10 10 31, Bielefeld 33501, Germany
{yousuf.aboamer, marcus.kracht}@uni-bielefeld.de

Abstract. In this paper, we introduce a completely lexicalist approach to deal with Arabic morphology. This purely surface-oriented treatment is part of a comprehensive mathematical approach to integrate Arabic syntax and semantics using overt morphological features in the string-to-meaning translation. The basic motivation of our approach is to combine semantic representations with formal descriptions of morphological units. That is, the lexicon is a collection of signs; each sign δ is a triple $\delta = \langle E, C, M \rangle$, such that E is the exponent, C is the combinatorics and M is the meaning of the sign. Here, we are only concerned with the exponents, i.e. the components of a morphosemantic lexicon (for a fragment of Arabic). To remain surface-oriented, we allow for discontinuity in the constituents; constituents are sequences of strings, which can only be concatenated or duplicated, but no rule can delete, add or modify any string. Arabic morphology is very well known for its complexity and richness. The word formation in Arabic poses real challenges because words are derived from roots, which bear the core meaning of their derivatives, formed by inserting vowels and maybe other consonants. The units in the sequences are so-called glued strings rather than only strings. A glued string is a string that has left and right context conditions. Optimally morphs are combined in a definite and non-exceptional linear way, as in many cases in different languages (e.g. plural in English). The process of Arabic word formation is rather complex; it is not just a sequential concatenation of morphs by placing them next to each other. But the constituents are discontinuous. Vowels and more consonants are inserted between, before and after the root consonants resulting in what we call "fractured glued string", i.e. as a sequence of glued strings combined in diverse ways; forward concatenation, backward concatenation, forward wrapping, reduction, forward transfixation and, going beyond the multi-context free grammars (MCFGs), also reduplication.

Keywords: Discontinuity · Arabic Morphology · Surface orientation · Morphosemantics

1 Introduction

Arabic is currently the most spoken Semitic language in the world with a number of speakers approaching 415 million, according to the CIA world factbook

[1], and is the/an official language of 22 countries in a vast area extending from the Arabian Peninsula to the Fertile Crescent, in addition to some other countries in the middle east. It has recently become the focus of an increasing number of projects in natural language processing and computational linguistics [2]. In this regard, we investigate a new approach to integrate Arabic morphology, syntax and semantics. The central claim of this approach is that the argument structure provides an interface between syntax and semantics. The main function of the argument structure is to declare how the functor's semantic arguments are realized on the surface. [3][1] In this paper, we are only concerned with morphology, i.e. we show how to deal with Arabic morphology in a way that allows us to go beyond mere word formation and provide a fully compositional treatment of entire sentences. Sentences are composed from units that are combined in some way. Each unit is either a string, i.e. a sequence of letters, or a sequence of strings. Furthermore, to implement adjacency constraints we use so-called glued strings in place of strings, or sequences thereof. In order to deal with the complexity of Arabic morphology in a completely surface-oriented approach, we have to deal with a number of problems, especially the discontinuity of morphemes, matching among morphemes, and morphophonemic alternations resulting from the interaction among different types of morphs.
This paper consists of four parts in addition to the introduction. In Part 2, we discuss briefly the morphological system of Arabic and the various paradigms proposed for the study of Arabic word formation showing how the proposed approach differs from these paradigms. In Part 3 we give a description of the proposed approach and in Part 4 we show how we handle the morphology of Arabic within the proposed framework. Results and future work are indicated in Part 5.

2 Arabic morphology: theoretical and computational approaches

Arabic morphology is very well known for its complexity and richness. However, it exhibits rigorous and elegant logic [4]. It differs from that of English or other Indo-European languages because it is, to a large extent, based on discontinuous morphemes. Words in Arabic are derived from roots, which bear the core meaning of their derivatives, by inserting vowels and maybe other consonants. Roots are relatively invariable, discontinuous bound morphemes, typically containing three consonants in a certain order, and interlocked with vowels and other consonants to form stems[4]. Let us consider the following example: the triliteral (3 consonantal) root /د ر س/ ['d r s'] in Buckwalter's transliteration model,[2] is supposed to bear the meaning of studying and from which

[1] The basic notions of the proposed approach are introduced in [3]. However, we concentrate in this paper on the most related ones to the nature of Arabic morphology and how they are applied to it.

[2] In this work, we use Buckwalter's transliteration model, but we made some modifications to make our work easier.

words like /درس/ ['daras', 'he studied'] or ['durisa', 'be studied'] and /دارس/ ['dAris', 'student'] are derived. This process has evolved extensively and very productively in order to cover a vast array of meanings associated with each semantic field. The vast majority of Arabic words are derived in this way and therefore can be analyzed as consisting of two different bound morphs: a **root** and a so-called **pattern**, which interlock to form a word and neither of them can occur alone in the sentence. Moreover, after composing stems from discontinuous roots and some vowels or other consonants, they may concatenate with affixes or clitics, as clearly stated, for example, in the Arabic masculine sound plural form /معلمون/ ['muEalolimUna', 'teachers'] obtained from the concatenation of the strings /معلم/ ['muEalolim', 'teacher'] and /ون/ ['Una', a nominal suffix for the masculine plural and a verbal suffix for masculine plural in indicative mood]. Stems may host suffixes that come after the stem as in this example, prefixes that come before the stem as in /يكتب/ ['yakotub', 'he writes'] obtained from /كتب/ ['kotub'] and /ي/ ['ya', third person singular of imperfective form] or both suffixes and prefixes as in /يكتبون/ ['yakotubUna', 'they write'] obtained from /كتب/ ['kotub'], /ي/ ['ya', third person singular of the imperfective form] and /ون/ ['Una', a suffix for the nominative masculine plural] [2].

As a result of such degree of richness and the complexity of word formation, Arabic morphology has been the focus of research in natural language processing for a long time[5–9, 2]. Researchers have adopted different approaches in the treatment of Arabic morphology, both theoretically and computationally [7]. Most of the efforts have been particularly devoted to addressing morphological analysis, generation and disambiguation. Arabic morphotactics have been sufficiently described and handled using finite state operations [10–13]. Al-Sughaiyer and Al-Kharashi [5] provide a comprehensive survey for Arabic morphological analysis techniques. Also, Dichy and Farghaly [7] count several approaches such as root and pattern, stem-based, machine learning among others. Depending on one of these approaches, many Arabic morphological analysers and generators have been developed in the last two decades particularly by the works of Tim Buckwalter [14, 15], Habash and Rambow [16], the Linguistics Data Consortium [17], Sawalha [18] and Attia [12]. Recent works have attempted either to improve the accuracy of the analyser [19], to add some other features [20] or to focus on a specific dialect [21]. We argue that there is a gap between morphology and semantics in the current approaches to Arabic morphology and the focus has been placed only on the issues of word formation and decomposition. We cannot go higher than morphology. Morphologically speaking, the finite state approach [11] can be seen as relatively close to our approach. However, a finite state machine does not know units, it is just a sequence of states. Our approach has both a context free grammar on the one hand and a mechanics to deal with discontinuity on the other hand. This allows the computer to understand $\langle$k, t, b$\rangle$ and $\langle$a, a$\rangle$ as units, not states, because they have meanings. A clearer example can be seen, for example, in German where the "trennbare Präfix" (separable prefix), like /auf/ in /aufmachen/, and the verb appear in some cases as two parts. This can occur, for instance, in the imperative, as in "Mach bitte das Fenster auf!" (*Open the window, please.*). However, /mach/ and /auf/ together form one unit (as they can only be interpreted together) but this unit is discontinuous. This also applies to Arabic roots. Our approach is different from previous and current approaches in (i) it is compositional; morphological units have meanings and the meaning of complex expressions is determined by the meaning of these smaller units and their order; (ii) morphology is not distinguished from syntax. Rather, it is a lexicalist approach on a par with categorial grammar; (iii) it is more restrictive; since it purely surface-oriented, it allows only grammatical constituents. This eliminates the overgeneration.

3 A purely surface-oriented approach

At the simplest level of description, a natural language is simply a set of strings over a given alphabet A [22]. A^* denotes the set of all strings over A. The concatenation of two strings x and y is denoted by $x^\frown y$ or simply xy. Concatenation is associative, that is, $(x^\frown y)^\frown z = x^\frown(y^\frown z)$, $\langle A^*, {}^\frown, \varepsilon\rangle$ constitutes a monoid [22]. In the sequel, variables for strings are formed using a vector arrow, e.g. $\vec{x}$.

3.1 Glued strings

We propose the notion "glued string" rather than just string. A glued string is a string with two context conditions: one for the left context and one for the right context. These conditions specify the properties of the string $\vec{x}$ such that $\vec{x}$ can appear in $\vec{u}\,\vec{x}\,\vec{v}$. A very good and clear example in Arabic is /ة/ ['p'], which is followed only by space (forget now about case markers) and consequently no other strings can follow it directly. This is a left-hand condition for the suffix and a right-hand condition for the /ة/. Moreover, the / ة/ is not allowed to be preceded by a space. So, before we can formally define glued strings we should, firstly, define the notion of a requirement. A **requirement** is a pair $(s, \vec{x})$, where s is a sign[3] and $\vec{x}$ a string. Roughly, the context must contain one of the strings with sign $+$ as suffix (if on the left) and as prefix (if on the right), while avoiding all the strings with sign $-$. We can now define the glued string as follows.

Definition 1 ***(Glued string)*** *A* ***glued string*** *is a triple $j = \langle L, \vec{x}, R\rangle$, where L is a set of left requirements, $\vec{x}$ is a string, and R is the set of right requirements.*

We give three explanatory examples of glued strings in English, Hungarian (taken from [3]) and in Arabic (/␣/ denotes the blank):

[3] There shouldn't be a confusion with the term sign as used in the abstract. A sign here is just plus $+$ or minus $-$.

1. $j = \langle \{(+,\text{ch}),(+,\text{s}),(+,\text{sh}),(+,\text{x}),(+,\text{z})\},\text{es},\{(+,␣)\}\rangle$
   This example codes the fact that the plural morph /es/ in English is suffixed only to words ending in ch, s, sh, x or z, while it must strictly be at the end of the word.
2. $j_1 = \langle \{(+,\text{b})\},\text{bal},\varnothing\rangle$
   This example from Hungarian specifies that the instrumental form /bal/ appears only after b.
3. The previous example / ة/ with its right and left requirements as a glued string j_2: $j_2 = \langle \{(-,␣)\},$ ة $,\{(+,␣)\}\rangle$

3.2 Occurrence

A string occurs in another string if the first is a substring of the second. Let $\overrightarrow{x}$ and $\overrightarrow{y}$ be two strings. An occurrence of $\overrightarrow{x}$ in $\overrightarrow{y}$ is a pair $o = \langle \overrightarrow{u}, \overrightarrow{v}\rangle$ such that $\overrightarrow{y} = \overrightarrow{u}\,\overrightarrow{x}\,\overrightarrow{v}$. A similar definition of occurrence can be also found in [22]. For example, let $\overrightarrow{y}$ = / معلمون / ['muEalolimUna', 'teachers'], $\overrightarrow{u}$ = / م /['mu'] and $\overrightarrow{v}$ = / ون / ['Una']. Then $\overrightarrow{x}$ = / علم /['Ealolim'].

If $o_1 = \langle \overrightarrow{u_1}, \overrightarrow{v_1}\rangle$ is an occurrence of $\overrightarrow{x_1}$ in $\overrightarrow{y}$ and $o_2 = \langle \overrightarrow{u_2}, \overrightarrow{v_2}\rangle$ is an occurrence of $\overrightarrow{x_2}$ in $\overrightarrow{y}$ then it is said that o_1 is to **the left of** of o_2 and (o_2 is to the **the right of** o_1) if $\overrightarrow{u_1}\,\overrightarrow{x_1}$ is a prefix of $\overrightarrow{u_2}$. If $\overrightarrow{u_2} = \overrightarrow{u_1}\,\overrightarrow{x_1}$, it is said that o_1 is **immediately to the left of (left adjacent to)** o_2 and o_2 is **immediately to the right of (right adjacent to)** o_1. The two occurrences o_1 and o_1 are said to be **contiguous** if one of them is **left** or **right adjacent** to the other. Otherwise, they **overlap**.

3.3 Morphological class

The notion of a glued string has been developed to handle the conditions of combination. However, in many cases this notion doesn't suffice to handle all possible conditions. The feminine suffix in Arabic / ة / ['p'] is used in most cases to differentiate between masculine and feminine as in / معلم / ['muEalolim', 'teacher (male)'] and / معلمة / ['muEalolimap', 'teacher (female)'] respectively. However, in some cases this suffix is ignored because the noun or the adjective refers by nature to a female as in / طالق / ['TAliq', 'divorced'] and / حائض / ['HA'iD', 'menstruant']. Also, in some other cases the / ة / ['p'] is ignored because the same word is used to refer to persons of both genders as in / جريح / ['jarYH', 'injured'] and / قتيل / ['qatYl', 'killed'].

This means that the combination process depends in some cases on information that cannot be captured only by phonology. Therefore, we need an additional mechanism to capture such cases. This is the notion of "morphological class". These are properties of individual morphs, not morphemes, that control the behavior of a morph under combination. When two morphs are to be combined, one of them takes the role of the argument, while the second takes the role of the functor. When a morph m_1 takes a morph m_2 as its argument, the two give a third morph m_3. This can be written in a form of a function:

$$m_1(m_2) = m_3$$

In the combination process between root morph and pattern morph, for example, the pattern morph takes the role of the functor, let say as above m_1 and the root morph, m_2, takes the role of the argument. Their combination produces a stem or m_3 as in the following example:

$$m_1(m_2) = m_3$$

$$pattern(root) = stem$$

$$\langle \mathrm{a}, \mathrm{a} \rangle(\langle \mathrm{k}, \mathrm{t}, \mathrm{b} \rangle) = /\mathrm{katab}/$$

As we can see from the example, to properly handle the combination between roots and patterns, we have to first deal with the properties of m_1 and m_2 before the combination; i.e. to be combined. Then, we have the properties of m_1 and m_2 after combination; i.e. m_3. The combination in the previous example results in a perfective stem in the active form. Compare that, now, with the following combination:

$$m_1(m_2) = m_3$$

$$pattern(root) = stem$$

$$\langle \mathrm{u}, \mathrm{i} \rangle(\langle \mathrm{k}, \mathrm{t}, \mathrm{b} \rangle) = /\mathrm{kutib}/$$

The result of the combination, in this case, is the perfective stem in the passive form. This suggests that each morph that takes the role of the functor (m_1, or the pattern morph in these examples) has two classes, an "ingoing class" and the "outgoing class". The ingoing class specifies what class the argument (m_2, or the root morph) must have to combine with m_1. The outgoing class states what class the combination of m_1 and m_2 has. This mechanism has proven sufficient not only in allowing proper combination but also in preventing improper ones. The number of features that guide the combination can be large and the morphological classes themselves can be rather complex. Therefore, an attribute value matrix (AVM) in the following form is suggested:

$$\begin{bmatrix} ATTRIBUTE_1 : valueset_1 \\ ATTRIBUTE_2 : valueset_2 \\ \cdots \quad\quad \cdots \\ ATTRIBUTE_n : valueset_n \end{bmatrix}$$

If $n = 0$, the AVM is empty, and if $n \geq 1$, the $ATTRIBUTE_i$ are names of features such as case, gender, number etc. and the $valueset_i$ are sets of admissible values for each attribute. Here, given an attribute a, $rg(a)$ denotes the set of admissible values for a. So, we require $valueset_i \subseteq rg(ATTRIBUTE_i)$ for every $i \leq n$. For example, in Arabic $rg(NUM) = \{singular, dual, plural\}$. Hence, the following is a legal AVM.

$$\left[NUM : \{dual, pl\} \right]$$

Sets of values encode underspecification. Using logical notation, we may write instead

$$\left[NUM : dual \vee pl \right]$$

$\top$ denotes the set of all values. So we have

$$\left[NUM : \top \right] \equiv \left[NUM : \{sing, dual, pl\} \right]$$

where $\equiv$ denotes logical equivalence. Arabic nouns/adjectives, for instance, are inflected for case, state, gender and number. These four features have the following ranges:

$$\begin{aligned} rg(CASE) &= \{nom, acc, gen\} \\ rg(STATE) &= \{def, indef\} \\ rg(GEN) &= \{masc, fem\} \\ rg(NUM) &= \{sing, dual, pl\} \end{aligned}$$

Conjunction and disjunction may be used to combine AVMs. The following equivalence is evident from the definition of AVMs.

$$\begin{bmatrix} NUM : \{sing\} \\ CASE : \{nom\} \end{bmatrix} \equiv \left[NUM : \{sing\} \right] \wedge \left[CASE : \{nom\} \right]$$

When an attribute receives the empty set as value, this means that we have an empty disjunction, which is defined to be false ($\bot$):

$$\left[CASE : \varnothing \right] \equiv \left[CASE : \bot \right]$$

We can apply the usual laws of logic. Consider, for example, two attributes (say, *CASE* and *NUM*) and use the law of distribution:

$$\begin{bmatrix} NUM : sing \vee pl \\ CASE : \quad nom \end{bmatrix} \equiv \begin{bmatrix} NUM : sing \\ CASE : nom \end{bmatrix} \vee \begin{bmatrix} NUM : \quad pl \\ CASE : nom \end{bmatrix} \equiv$$

$$\left[CASE : nom \right] \wedge \left(\left[NUM : sing \right] \vee \left[NUM : pl \right] \right) \equiv$$

$$\left(\left[CASE : nom \right] \wedge \left[NUM : sing \right] \right) \vee \left(\left[CASE : nom \right] \wedge \left[NUM : pl \right] \right)$$

Definition 2 ***(Feature space)*** *A **feature space** is a triple $\sigma = \langle A, V, rg \rangle$ such that A is a finite set of attributes, V is a finite set of values and $rg : A \to \wp(V)$ a function such that for all $a \in A$, $rg(a) \neq \varnothing$. For Arabic, as given above, we may put $A := \{CASE, NUM, GEN\}$, $V := \{nom, acc, gen, sing, dual, pl, masc, fem\}$.*

Abstractly, an AVM is a partial function f from attributes to sets of admissible values. If f is undefined on a, we may extend f by putting $f(a) := rg(a)$. Thus, f may also be considered a total function. Consider, for example, the

AVM of nouns/adjectives that follow prepositions in Arabic (in the genitive case). The prepositions behave as a functor that takes nouns/adjectives as argument regardless the number, gender or state and give nouns/adjectives in genitive case. Therefore, in this case, the $f(NUM)$, $f(GEN)$ and $f(STATE)$ are equal to the $rg(NUM)$, $rg(GEN)$ and $rg(STATE)$ as shown in the following AVM:

$$\begin{bmatrix} POS: & noun \vee adj \\ NUM: & \top \\ GEN: & \top \\ STATE: & \top \end{bmatrix}$$

So, by convention we may extend f to $f\,(STATE) = \{def, indef\}$.

3.4 Discontinuity, Reduplication and Handlers

Discontinuity is used in grammatical analysis to refer to the splitting of a construction by the insertion of another grammatical unit [23]. The concept of discontinuity is central for handling Arabic in a completely surface-oriented compositional approach because morphs are the meaningful units of speech but clearly are not continuous, contrary to what is the case in most languages. The plural is formed in English simply by concatenation e.g. of /dog/ and /s/ to get /dogs/. However, the process of Arabic word formation is rather complex; it is not just a sequential concatenation of morphs by placing them next to each other: the constituents can be discontinuous. Vowels and more consonants are inserted between, before and after the root consonants. The idea is clear when we consider, again, the triliteral (3 consonant) root /ك ت ب/ ['k t b'] and some of its derivatives like /كتب/ ['kataba', 'he wrote' or 'kutiba', 'was written'], /كاتب/ ['kAtib', 'writer' or 'kAtab', 'correspond with'] and /مكتوب/ ['makotUb', 'is written']. Both root morphs and pattern morphs are instances of **fractured glued strings**.

Definition 3 ***(Fractured glued string)*** *A* ***fractured glued string*** *is a sequence of glued strings. If $\gamma_0, \gamma_1, \cdots, \gamma_{m-1}$ are glued strings, then $g := \gamma_0 \otimes \gamma_1 \otimes \cdots \otimes \gamma_{m-1}$ denotes the fractured glued string, formed from the γ_i in this order. γ_i is called the ith section of g. m is called the dimension of g, referred to as $dim(g)$. The unique fractured string with dimension 0 is denoted by ζ.*

We can write the Arabic root as k $\otimes$ t $\otimes$ b and the morph of the third person singular in the active form of the past tense a $\otimes$ a. The content of a string is defined as:

Definition 4 ***(String content)*** *If $\gamma = \langle L, \overrightarrow{x}, R\rangle$ is a glued string, then $c(\gamma) = \overrightarrow{x}$. Furthermore, $c(\otimes_{i<n}\gamma_i) = c(\gamma_0)^\frown c(\gamma_1)^\frown ...^\frown c(\gamma_{n-1})$.*

Context free grammars are not equipped to describe discontinuity. To deal with discontinuous constituents or, more particularly, to combine two fractured glued strings, [3], following [24] suggests using a combinatorial function called **handler**. A handler can be defined as follows:

Definition 5 ***(Handler)*** *A **handler** is a sequence H of sequences of pairs (i, b), where i is a natural number and b a boolean. The members of H are called its **sections**. A pair (i, b) is said to **occur** in H, in symbols $(i, b) \in H$, if there is a section of which (i, b) is some member. The pairs occurring in H are called its parts. Parts may have several occurrences. The result of applying H to two fractured strings g and h such that $g = \gamma_0 \otimes \gamma_1 \otimes \cdots \otimes \gamma_{m-1}$ and $h = \eta_0 \otimes \eta_1 \otimes \cdots \otimes \eta_{n-1}$ is defined as follows. Put:*

$$(i, b)\,(g, h) = \begin{cases} \gamma_i & \textit{if } b = \textit{true} \\ \eta_i & \textit{else} \end{cases}$$

Now, for the sequence $h_i = (i_0, b_0), (i_1, b_1), \cdots, (i_{p-1}, b_{p-1})$, we put

$$h_i(g, h) := (i_0, b_0)(g, h)\hat{}(i_1, b_1)(g, h)\hat{}\cdots\hat{}(i_{p-1}, b_{p-1})(g, h)$$

Finally, let $H = (h_0, h_1, \cdots h_{q-1})$ have q sections, then:

$$H(g, h) := h_0(g, h) \otimes h_1(g, h) \otimes \cdots \otimes h_{q-1}(g, h)$$

A handler is used if and only if it is proper. A proper handler is defined as:

Definition 6 ***(Proper Handler)*** *A handler H is proper if for all numbers i, j and Booleans b, if H contains (i, b) and $j < i$ then H also contains (j, b). The dimension of a handler H is defined by:*

$$dimH = (\{i : (i, true) \in H\}, \{i : (i, false) \in H\})$$

If H is proper, $dimH$ is a pair of numbers, such that:
0 is the empty set ϕ and $n + 1 = \{0, 1, ..., n\}$.

A handler $H(g, h)$ is defined if and only if H is proper and $dimH = (dim(g), dim(h))$ i.e. if all sections of the two fractured strings are used in H. This combinatorial function allows a sequence of glued strings to be combined in diverse ways: forward concatenation, backward concatenation, forward wrapping, reduction, forward transfixation and, beyond the MCFGs [24], reduplication [25] as shown in the following examples:

- Forward Concatenation:
  Put $F := \langle\langle(0, true), (0, false)\rangle\rangle$.
  Then $F(\overrightarrow{x}, \overrightarrow{y}) = \overrightarrow{x}\,\overrightarrow{y}$
- Backward Concatenation:
  Put $B := \langle\langle(0, false), (0, true)\rangle\rangle$.
  Then $B(\overrightarrow{x}, \overrightarrow{y}) = \overrightarrow{y}\,\overrightarrow{x}$
- Forward Wrapping:
  Put $W := \langle\langle(0, true), (0, false), (1, true)\rangle\rangle$.
  Then $W(\overrightarrow{x} \otimes \overrightarrow{v}, \overrightarrow{y}) = \overrightarrow{x}\,\overrightarrow{y}\,\overrightarrow{v}$
- Reduction:
  Put $R := \langle\langle(0, true), (1, true)\rangle\rangle$.
  Then $R(\overrightarrow{x_0} \otimes \overrightarrow{x_1}) = \overrightarrow{x_0}\,\overrightarrow{x_1}$

- Transfixation:
  Put $T := \langle\langle(0, true), (0, false), (1, true), (1, false)\rangle\rangle$.
  Then $T\left(\overrightarrow{x_0} \otimes \overrightarrow{x_1}, \overrightarrow{y_0} \otimes \overrightarrow{y_1}\right) = \overrightarrow{x_0}\,\overrightarrow{y_0}\,\overrightarrow{x_1}\,\overrightarrow{y_1}$

We can see how this works with a sequence of glued strings in Arabic. If we want, for example, to form the word/كاتب/ ['kAtib', 'writer'] from the root k $\otimes$ t $\otimes$ b and the vowels A $\otimes$ i we apply the following handler (note that the pattern morph plays the role of the functor). Put

$$H := \langle\langle(0, false), (0, true), (1, false), (1, true), (2, false)\rangle\rangle$$

If we apply a function that maps from each part of the handler to its corresponding string in the two fractured strings, we get the following:

– $(0, false)$	/k/
– $(0, true)$	/A/
– $(1, false)$	/t/
– $(1, true)$	/i/
– $(2, false)$	/b/

The result of applying this handler to the two fractured glued strings k $\otimes$ t $\otimes$ b and A $\otimes$ i is /kAtib/ as shown:

$$H(\mathrm{A} \otimes \mathrm{i}, \mathrm{k} \otimes \mathrm{t} \otimes \mathrm{b}) := \mathrm{k}^\frown\mathrm{A}^\frown\mathrm{t}^\frown\mathrm{i}^\frown\mathrm{b} = \mathrm{kAtib}$$

Reduplication is also an important feature in Arabic word formation. In some cases, Arabic tends to duplicate a specific letter (string) to get a new word as in /كتّب/ ['katotab', 'made sb write'] from /ك ت ب/ ['k t b'] or /درّس/ ['daroras', 'taught or educated'] from /د ر س/ ['d r s']. In this case, for example, reduplication results in not only a different form of the verb, but also a different meaning. Both /katab/ and /katotab/ have the same root /k t b/ and the same inserted vowels. However, the only difference between the two forms lies in the reduplication of the second consonant in the case of /katotab/ which changes the meaning from 'he wrote' to 'he made [sb] write'. This applies also to /daras/ and /daroras/. In some other cases, reduplication occurs in the root itself when the second consonant is doubled which means that the second and third consonants are the same as in /شدّ/ ['$dod'] and /مرّ/ ['mror']. Reduplication is represented orthographically with shaddah or tashdid above the duplicated letter (/دّ/ and /رّ/) and represented in Buckwalter's transliteration model as /$\sim$/ but we use instead two letters separated by sukoon because the duplicated letter is actually two letters; the first is followed by sukoon and the second by a vowel. This type is referred to as doubled or geminate verbs. Moreover, some Arabic four consonant roots are composed by the reduplication of the first two consonants twice as in /ز ل ز ل/ [z l z l] and /ز ع ز ع/ [z E z E].

Reduplication is not a unique feature of Arabic and it is also not the sole morphological operation. However, there are languages in which processes like reduplication is the primary morphological operation [26]. The plural formation in Malay is obtained, for instance, by doubling the singular form as in

/orang/ ”man” which becomes /orang-orang/ in the plural. The two parts are separated by a hyphen in the written language.

In multiple context free grammars [27], no component is allowed to appear in the value of the function more than once. This is not the case in the above-mentioned examples in Arabic and Malay. To handle such cases, the following handler for the plural form in Malay must be used.

$$D := \langle\langle (0, \mathit{false}), (0, \mathit{true}), (0, \mathit{false}) \rangle\rangle.$$

Then

$$D\,(\text{-}, \text{orang}) = \text{orang\^{}-\^{}orang}$$

Arabic is not an exception; we allow the handler to combine the multi occurrence of any substring but we insert a sukoon between the duplicated letter. Put

$$D := \langle\langle (0, \mathit{false}), (0, \mathit{true}), (1, \mathit{false}), (1, \mathit{true}), (1, \mathit{false}), (2, \mathit{true})\ (2, \mathit{false}) \rangle\rangle$$

Then

$$D\,(\text{a} \otimes \text{o} \otimes \text{a}, \text{k} \otimes \text{t} \otimes \text{b}) = \text{k\^{}a\^{}t\^{}o\^{}t\^{}a\^{}b} = \text{katotab}$$

as well as

$$D\,(\text{a} \otimes \text{o} \otimes \text{a}, \text{d} \otimes \text{r} \otimes \text{s}) = \text{d\^{}a\^{}r\^{}o\^{}r\^{}a\^{}s} = \text{daroras}$$

However, in the real implementation, we deal with the reduplication in Arabic from a different perspective for the sake of simplification. We utilize the morphological class technique to sub-categorize the root into several types and each category is allowed to merge with specific patterns as discussed in details in the introduction.

3.5 Morphs and Morphemes

The actual units of expression of language are the morphs. They comprise three components. The first is the exponent, which is a fractured glued string. The second is a sequence of selectors, which determine what arguments the morph takes. And the third is a rank function. This function is only needed for empty morphs, to prohibit infinite derivations, and will concern us no further. Morphemes, which are the only meaning bearing units, are sets of morphs that share a common semantics. Thus, the lexical units pair meaning representations with morphemes, not morphs. This accounts for the fact that morphemes can have many different surface forms.

Definition 7 ***(selector)*** *A* ***selector*** *is defined formally as triple $\sigma = (M, N, H)$, where M and N are morphological classes and H is a handler. M is called the* ***in-class*** *of σ and N is its* ***out-class****.*

The role of the selector is to specify what happens when a morph is applied to another morph. The application of one morph (the functor) to another (the argument) is only defined if the in-class of the functor unifies with the out-class of the argument. Given, for example, a functor $\sigma = (M, N, H)$ and an argument $\sigma' = (M', N', H')$, the application of σ to σ' is defined in first instance as follows:

$$\sigma \cdot \sigma' := (M', N, H \circ H')$$

However, underspecification must be handled properly. The way this is standardly done is that the out-class is not actually underspecified, but is a function of its in-class, which is genuinely underspecified. Underspecified values in the out-class are copies of the actual in-class values. Indeed, the proper way to view selectors is as pairs (f, H) where f is a function from fully specified morphological classes to fully specified morphological classes, and H is a handler. This function is given by the pair of AVSs as follows. For each attribute ATT, the associated function f is the following.

- ATT is given a value a in M and a value b in N. Then f is defined on all nonempty values $a' \subseteq a$ and returns b.
- ATT is given no value in M but value b in N. Then f is defined on all nonempty values $a' \subseteq rg(\text{ATT})$ and returns b.
- ATT is given a value a in M but no value in N. Then f is defined on all nonempty values $a' \subseteq a$ and returns a'.
- ATT is neither given a value in M nor in N. Then f is defined on all $a \subseteq rg(\text{ATT})$ and returns a.

The product of (M, N) and (M', N') is specified by computing the values of each occurring attribute.

Definition 8 ***(Morpheme)*** *A* ***morpheme*** *is a set of morphs that share the same semantics but differ in the form.*

Arabic broken plural is, from this perspective, highly allomorphic; for a given singular pattern two different plural forms may be equally frequent, and sometimes, for some singulars as many as three further statistically minor patterns are also possible [28]. Given morphemes M an N, the combination $M \star N$ is the set of all $m(n)$ such that $m \in M$ and $n \in N$.

4 Arabic Morphology within the Proposed Framework

In order to deal with the complexity of Arabic in a completely surface-oriented approach, we have to deal with two basic problems: root and pattern matching and morphophonemic alternations

4.1 Root and Pattern Matching

Roots behave differently when they combine with patterns to form stems. Each root chooses specific vowels from the following six possibilities to form perfective and imperfective stems:

– (a, a) and (o, u)	– (a, a) and (o, a)	– (a, u) and (o, u)
– (a, a) and (o, i)	– (a, i) and (o, a)	– (a, i) and (o, i)

Arabic traditional grammarians attempted to handle this problem by classifying roots into basic and sub-categories. That is, roots are classified depending on the number of consonants into triliteral and quadrilateral. This subdivision is not necessary since handlers need a fixed length. However, it is better to use this as an explicit feature. Each category is further divided into different terms. Glued strings are not sufficient because the phonology does not provide us with enough information to determine which root interlock with which pattern. We take advantage of the traditional categorization and sub-categorization of morphemes and utilize the notion of morphological classes. Particularly, each root morph has a morphological class with four attributes:

– **Type**: This attribute differentiates between the two basic types of roots; triliteral and quadrilateral. This means that the **type** attribute in the morphological class of any root has a set of values that has two elements and consequently has the following range:

$$rg(TYPE) = \{tri, quad\}$$

– **Subtype**: This attribute divides Arabic roots depending on the nature of the letters of the root, not on their number, and position of specific letters within the root. Roots are divided into sound (free of semi-vowels, hamza and reduplication), geminate (with a duplicated letter), first hamzated (the first letter is hamza), second hamzated (the second letter is hamza), third hamzated (the third letter is hamza), assimilated (the first letter is either wAw or yA), hollow (the second letter is either wAw or yA), defective (the third letter is either wAw or yA), first weak (the first and third letter are either wAw or yA) and second weak (the second and third letter are either wAw or yA). Thus, the **Subtype** attribute has a range of ten elements.
– **Form**: As mentioned before, Arabic roots, in terms of the numbers of letters, are divided into triliteral and quadrilateral. However, other consonants can be added to the two basic types. The triliteral roots can host up to three other consonants while the quadrilateral roots can have only two more consonants. In practice, not every lexical root occurs in all different forms but they vary from one another and dictionaries normally list all the forms in which a lexical root regularly appears [4]. If the root consists of only the three or four consonants, it is in the base form called form I, which is also referred to in Arabic as mujarorad, literally the "stripped" form; otherwise it is in one of the forms II to X, which are referred to as mazId, literally, "increased" forms, i.e., more morphologically complex[4]. Therefore, in terms of the form, the root takes a Roman number extending form I to X in case of triliteral roots and from I to IV in case of quadrilateral roots. Thus the **form** attribute in the morphological class has the following range:

$$rg(FORM) = \{I, II, III, \cdots, X\}$$

– **Stem-Eayon**: It is traditional to refer to the short vowel which follows the second root consonant of a verb as the "stem vowel" [4] and we saw in the beginning of this section that the stem vowel (Eayon) may vary from one stem to another. We capture these possibilities using an attribute with a value set that has six values. These values specify the stem vowel in perfective and imperfective forms and therefore this attribute has the following range: $rg(\mathit{Stem\text{-}Eayon}) = \{au, ai, aa, ia, uu, ii\}$.

The morphological class of the root $\langle$k, t, b$\rangle$

$$\begin{bmatrix} \mathit{TYPE}: & \mathit{tri} \\ \mathit{SUBTYPE}: & \mathit{sound} \\ \mathit{FORM}: & \mathit{I} \\ \mathit{STEM\text{-}EAYON}: & \mathit{au} \end{bmatrix}$$

On the other hand, pattern morphs are provided with an in-class (for the hosted root morph) and an out-class (for the result of the merge). The merge is defined only if the out-class of the root matches the in-class of the pattern. Consider, for example, the in-class of the pattern morph $\langle$o, u$\rangle$.

$$\begin{bmatrix} \mathit{TYPE}: & \mathit{tri} \\ \mathit{SUBTYPE}: & \{\mathit{sound}, \mathit{shamzated}, \mathit{shamzated}, \mathit{thamzated}\} \\ \mathit{FORM}: & \mathit{I} \\ \mathit{STEM\text{-}EAYON}: & \mathit{au} \end{bmatrix}$$

The out-class of the root morph and in-class of the pattern morph match and can be merged using this handler:

$$H := \langle\langle(0, \mathit{false}), (0, \mathit{true}), (1, \mathit{false}), (1, \mathit{true}), (2, \mathit{false})\rangle\rangle$$

$$H(\mathrm{o} \otimes \mathrm{u}, \mathrm{k} \otimes \mathrm{t} \otimes \mathrm{b}) := \mathrm{k\hat{}o\hat{}t\hat{}u\hat{}b} = \mathrm{kotub}$$

This applies to the pattern morph o ⊗ i which merges with the root morph d ⊗ r ⊗ b to give the imperfective stem /dorib/. It is now clear that both */dorub/ and */kotib/ are not allowed because neither the pattern morph o ⊗ u can merge with the root morph d ⊗ r ⊗ b nor the pattern morph o ⊗ i is allowed to merge with the root morph k ⊗ t ⊗ b. This is because the associated classes, in either cases, do not match.

4.2 Morphophonemic Alternations

The interaction among different type of morphs may result in phonological alternations. Arabic roots are broadly classified into two types depending on the presence or the absence of the wAw and yA': sound and weak. Weak roots, in particular, may undergo stem changes when inflected. Let's consider, for example, the following derived forms of the same hollow root /ب ي ع/ b ⊗ y ⊗ E:

– /باع/ ['bAEa', 'he sold'], masculine third person singular in the perfective form.

– /بعت/ [‘biEotu’, ‘I sold’], masculine or feminine first person singular in the perfective form.
– /يبيع/ [‘yabYEu’, ‘He sells’], masculine third person singular in the imperfective form.

Arabic grammarians attribute such morphophonemic changes to the rule of “origin of Alif” in forms like /باع/ [‘bAEa’] from /ب ي ع/ b $\otimes$ y $\otimes$ E and /قال/ [‘qAla’] from /ق و ل/ q $\otimes$ w $\otimes$ l. They argue that the “alif” is returned to its original “yA‘” or “wAw”, however, this says nothing about the dropping of the second letter of the root in /بعت/ [‘biEotu’] for instance. Anyway, in our approach, we are not concerned about the rules that justify such changes because in order to ensure the purely surface-oriented treatment of Arabic word formation, no rule is allowed to delete or modify any string. That is, we cannot say, for example, that the alif of the hollow verb is changed to its origin yA‘ or wAw in specific forms. Therefore, we don‘t add a rule to modify the alif of /قال/ [‘qAla’] to its original wAw and another rule to drop it in /قلت/ [‘qulotu’] and this applies also to many cases in which the interaction among different morphs may result in some changes. Instead, each root is a morpheme, that is, a set of morphs that correspond to all possible forms under merge. Thus, for the root morpheme /ب ي ع / $b \otimes y \otimes E$ we may have up to three morphs:

– /ب ي ع/ b $\otimes$ Y $\otimes$ E, from which we can get /يبيع/ [‘yabYEu’, ‘he sells’], /بيعوا/ [‘bYEUA‘, ‘sell’] etc.
– /ب ع/ b $\otimes$ E, from which we can get /بعت/ [‘biEotu’, ‘I sold’], /يبع/ [‘yabiEo’, ‘sell’] etc.
– /ب ا ع/ b $\otimes$ A $\otimes$ E, from which we can get /باع/ [‘bAEa’, ‘he sold’], /باعوا/ [‘bAEUA’, ‘they sold’] etc.

This permits not only to get the grammatical forms but also to disallow the ungrammatical ones like */يباعوا/ [’yabAEu’]. So far we dealt with challenges that Arabic morphology poses to this approach. We have already developed a lexicon for a fragment of Arabic. It is true that the lexicon is small and there are many other cases of morphophonemic changes, however, they can be handled in the same way.

5 Results and future work

In this paper, we presented a mathematical complete surface-oriented approach to handle Arabic morphology. In this approach, we dealt with the language as a sequence of strings that are only allowed to be concatenated and reduplicated but no rule is allowed to to add, remove or modify any string. At the very beginning, we reviewed briefly different theoretical and computational approaches to Arabic. We argue that there is a gap in current approaches between morphology and semantics. In order to handle Arabic morphology within this approach, we dealt with two problems: root and pattern matching and morphophonemic alternations. We have already developed a lexicon for a fragment of Arabic, and in the present time we extend our work on Arabic morphology in connection with the demands set by compositional semantics planning to come up with a morphosemantic lexicon for a fragment of Arabic.

References

1. CIA: CIA World Fact Book. Central Intelligence Agency, Washington, D.C.(2018).
2. Habash, N.: Introduction to Arabic natural language processing. Morgan and Claypool Publishers, (2010)
3. Kracht, M.: Agreement morphology, argument structure and syntax. Unpublished manuscript, Revision 8, (2016).
4. Ryding, K.: A Reference Grammar of Modern Standard Arabic. Cambridge University Press, (2005)
5. Al-Sughaiyer, I., Al-Kharashi, I. : Arabic morphological analysis techniques: A comprehensive survey. Journal of the Association for Information Science and Technology **55**(3), 189–213 (2004)
6. Soudi, A., Neumann, G. and Van den Bosch, A.: Arabic Computational Morphology: Knowledge-based and Empirical Methods. (eds). Springer, (2007)
7. Dichy, J., Farghaly, A.: Grammar-lexis relations in the computational morphology of Arabic. In Soudi, A., Neumann, G. and Van den Bosch, A.: Arabic Computational Morphology: Knowledge-based and Empirical Methods. (eds). Springer, 2007. PP. 115–140, Springer, (2007)
8. Boudchiche, M. et al.: AlKhalil Morpho Sys 2: A Robust Arabic Morpho-syntactic Analyzer. Journal of King Saud University-Computer and Information Sciences **29** (2), pp. 141–146 (2017)
9. Sawalha, M., Atwell, E.: Comparative Evaluation of Arabic Language Morphological Analysers and Stemmers. Coling 2008: Companion volume: Posters. 107–110 (2008)
10. Kay, M.: Nonconcatenative Finite-state Morphology. Proceedings of the third conference of the European chapter of the Association for Computational Linguistics. pp 2–10. (1987)
11. Beesley, K.: Finite-state Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans in 2001. ACL Workshop on Arabic Language Processing: Status and Perspective, pp. 1–8. (2001)
12. Attia, M., et al.: A corpus-based Finite-state Morphological Toolkit for Contemporary Arabic. Journal of Logic and Computation. **24**(2), 455–472. Oxford University Press (2013)
13. Aboamer, Y., Farghaly, A.: Mariam ComLex: A Bi-Directional Finite State Morphological Transducer for MSA. The 29th Annual Symposium on Arabic Linguistics. at the University of Wisconsin-Milwaukee. USA (2015)
14. Buckwalter, T.: Buckwalter Arabic Morphological Analyzer, Version 1.0. Linguistic Data Consortium, University of Pennsylvania, LDC Catalog No: LDC 2002 L49. 2002, ISBN 1-58563-257-0.

15. Buckwalter, T.: Buckwalter Arabic Morphological Analyzer, Version 2.0. Linguistic Data Consortium, University of Pennsylvania, LDC Catalog No: LDC 2004 L02. 2004, ISBN 1-58563-324-0.
16. Habash, N., Rambow, O, Roth, R.: MADA + TOKAN: A Toolkit for Arabic Tokenization, Diacritization, Morphological Disambiguation, POS Tagging, Stemming and Lemmatization. Proceedings of the 2nd international conference on Arabic language resources and tools (MEDAR). Cairo, Egypt (2009).
17. Maamouri, M., et al.: LDC Standard Arabic morphological analyzer SAMA v. 3.1. Linguistic Data Consortium, University of Pennsylvania, LDC Catalog No. LDC2010L01. ISBN 1-58563-555-3.
18. Sawalha, M., Atwell, E., Abushariah, M.: SALMA: Standard Arabic Language Morphological Analysis. 1st International Conference on Communications, Signal Processing, and their Applications (ICCSPA), pp. 1–6. (2013).
19. Abdelali, A., et al.,: Farasa: A Fast and Furious Segmenter for Arabic. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations, pp. 11–16 (2016)
20. Taji, D., et al.: An Arabic Morphological Analyzer and Generator with Copious Features. Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology, pp. 140–150 (2018)
21. Habash, N., Eskander, R., Hawwari, A.: A Morphological Analyzer for Egyptian Arabic. Proceedings of the twelfth meeting of the special interest group on computational morphology and phonology, pp. 1–9 (2012).
22. Partee, B., ter Meulen, A., Wall, R.: Mathematical Methods in linguistic. Linguistic Society of America, (1990)
23. Crystal, D.,: A dictionary of linguistics and phonetics, Sixth edition. Blackwell Publishing Ltd (2008).
24. Seki, H., et al.: On Multiple Context-free Grammars. Theoretical Computer Science, **88** (2), pp. 191–229 (1991)
25. Kracht, M., Aboamer, Y.: Argument Structure and Referent Systems. 12th International Conference on Computational Semantics IWCS (2017)
26. McCarthy, J.: A Prosodic Theory of Nonconcatenative Morphology. Linguistic Inquiry **12** (3), pp 373–418 (1981)
27. Kasami, T., Seki, H., Fujii, M.: Generalized Context-free Grammars, Multiple Context-free Grammars and Head Grammars. Preprint of WG on Natural Language of IPSJ, (1987)
28. Soudi, A., Violetta C., and Jamari, A.: The Arabic noun system generation. Proceedings of the International Symposium on the Processing of Arabic (2002).