Case based Arabic Morphological Analysis with Compositional Non-deterministic Automata

Abstract

Natural language processing uses morphological analysis to abstract and annotate text. Often times, and in particular in the context of the Arabic language, annotations resulting from an ongoing branch of morphological analysis may not be appropriate for the case study at hand. In this paper we present Sarf, a case based morphological analyzer for Arabic text. Sarf allows a case-based controller to control and refine the state of the analysis on the fly. It uses compositional non-deterministic finite state machines to analyze a stream of text. Each alive machine corresponds to a valid analysis. Sarf uses concatenative analysis based on recursive affixes to better preserve part of speech information. We automated the analysis of three books of Islamic narrations (hadith) using Sarf where we abstracted each book into a vector of complex structures. Our results show high accuracy and efficiency.

1 Introduction

Automated analysis of Arabic data sets, including texts, publications, records and digital media is essential with the huge digital Arabic content available nowadays. Morphological analysis is key in current automated analysis techniques for Arabic text. Current morphological analyzers (Al-Sughaiyer and Al-Kharashi, 2004) use concatenative analysis to analyze an Arabic word and consider its internal structure composed of several *morphemes*. A morpheme can be a *stem*, or an *affix*. An affix can be a *prefix*, *suffix*, or an *infix*. The analysis of one word may lead to several possible morphological solutions. For instance, the word أحَمَده may have two valid morphological analyses. The letter أَ may be prefix and the word means “I praise him”. The letter أَ may also be part of the stem أحَمَد (a proper noun) and the word means “his Ahmad”.

The accuracy of the solutions suffer due to inherent difficulties of morphological analysis of the Arabic language. For example, it is common practice to write Arabic text without short vowels. This greatly increases the ambiguity of Arabic text. Another difficulty is that Arabic letters can have up to four different forms corresponding to their position in a word, i.e, beginning, middle, end of word forms and separate forms. This allows the phrase المدرسةالى to be visually recognizable as two separate words الى and المدرسة without the need of a delimiter such as a space in between. This happens because the first word المدرسة ends with ة a non-connecting letter. These words occur often in text and greatly increase the difficulty of tokenization and are referred to as “run-on” words (Buckwalter, 2004).

Current morphological analyzers such as Buckwalter (2002), Beesly (2001), SAMA (Kulick et al., 2010), and ElixirFM (Smrž, 2007) exist. They take as input white space delimited tokens (Kulick et al., 2010), consider them as words, and enumerate all possible solutions. This approach has several problems. First, a white space delimited token may have more than one word. Second, the exhaustive enumeration may hurt performance and may not be necessary or appropriate in some case studies as noted in (Maamouri et al., 2010). Other morphological analyzers such as Amira (Mona Diab and Jurafsky, 2007; Benajiba et al., 2008), MAGEAD (Habash et al., 2005), and MADA+TOKAN (Habash et al., 2009) also use machine learning and support vector machines (SVM) to enhance the accuracy of the morphological analysis at the expense of performance.

We hypothesize that many NLP case studies need the morphological analyzer to answer simple queries that do not need high complexity and accuracy at the low morphological analysis level and that can often compensate at higher level for tolerable inaccuracy at low levels. For example, if a query concerns proper names and the prefix in question in the analysis connects only to a verb, we do not need to further analyze the rest of the word to provide an answer. We find evidence to our hypothesis in (Maamouri et al., 2010) where the addition of a new corpus to the Arabic Tree Bank (Maamouri and Bies, 2004) with features demanding resolution at an abstraction level higher than the text itself led to a refined analyzer (Kulick et al., 2010). We also find evidence in (Habash and Sadat, 2006) that different types of morphological analyses behave differently on the same case study. We strongly believe and we find evidence in our results that a case-based morphological analyzer that allows a case-based controller to intervene at every decision to guide, use, prune, and refine the morphological analysis is of high utility.

1.1 Sarf

In this paper, we present Sarf, a *novel case-based efficient morphological analyzer* that uses compositional non-deterministic Automata driven by a case based controller. Each alive machine in Sarf represents a valid morphological analysis. Sarf keeps alive all possible analyses of the text and gives opportunity to the controller to intervene at every single input letter. The decisions of the case-based controller can thus prune false positives early in the run. Each alive machine in Sarf takes as input one letter at a time from a text stream. Sarf does not assume that the word at hand is a token and performs tokenization on the fly based on morphological correctness. Up to our knowledge, this is the first morphological analyzer that handles the "run-on" words problem.

Sarf uses a *recursive* concatenative analysis with a novel refinement. Sarf introduces recursive affixes in order to retain better part of speech information and enhance the efficiency of affix matching.

We validate our hypothesis using a case study from the Islamic literature. A *hadith* is a narration related to the prophet Mohammad through a *sanad* or a sequence of narrators. The authentication of a hadith highly depends on the credibility of each of the narrators as reported in separate biography books. In this paper we consider the problem of automatically segmenting a hadith book into narrations, then segmenting each narration into its content or *matn* and its sanad. We also partition the sanad accurately into the separate narrators so that we can later look each one of them up in the biography books.

The hadith case study is interesting to Sarf since the controller considers most of the analyses where we have a concentration of proper names in the text. It ignores the text where proper names occur rarely as long as the text is valid. The controller is also interested in discovering a subset of words that relate persons to each other such as بن or "the son of" or words that mean narrate such as قال or "said". With Sarf, we successfully automated the analysis of three books of Islamic narrations (Ibn Hanbal, ; Al Tousi, ; Al Kulayni,) and extracted from each one of them a vector of a three level deep complex structure. We also showcase that the results of the accurate high level abstraction results of the case-based analysis was not affected by a major refinement to the analyzer that increased low level analysis accuracy.

We make the following key contributions.

1. **Case-based analysis:** We designed and implemented a case-based morphological analyzer where a case-based controller machine controls and guides the analysis.

2. **Exhaustive analysis:** Since we have a case-based controller that can decide on the fly whether an analysis is accepted or not, we can afford to keep all valid analyses modulo those pruned by the case controller. We do that using non-deterministic FSAs.

3. **Recursive affixes:** We refine the concatenative analysis of Buckwalter (2002) and use recursive affixes. This allows Sarf to perform better and maintain compositional part of speech tags.

4. **Hadith case study:** We evaluated Sarf using the hadith case study. We were able to efficiently extract the sequence of narrators into a complex data structure with three levels of hierarchy with high accuracy.

The rest of this paper is structured as follows. In Section 2 we compare Sarf to related work. Then in Section 3 we discuss Sarf and illustrate the way it works with a running example. We also explain how non-deterministic finite state Automata work. In Section 4 we present the hadith case study. We present our results in Section 5 and conclude with future work in Section 6.

2 Related Work

The survey in (Al-Sughaiyer and Al-Kharashi, 2004) discusses and compares several morphological analyzers. Analyzers such as (Khoja, 2001; Darwish, 2002) target specific applications in the morphological analyzer itself or use a specific set of part of speech tags as their reference. Sarf differs in that it is a general and rich morphological analyzer that reports all possible solutions. It is case-based in the sense that a controller that drives the morphological analyzer prunes these decisions.

Sarf builds upon the lexicon of Buckwalter(2002). It differs from Buckwalter in that it defines a shorter list of affixes and a longer list of affix compatibility rules to allow recursive affixes so that we can have prefix-prefix an suffix-suffix concatenations. This allows Sarf to better maintain and propagate the part of speech information tags associated with each prefix and suffix. This information is highly redundant with Buckwalter and that may lead to consistency issues. Sarf keeps all the possible analyses alive where each analysis corresponds to a set of decisions and positions in the text. Buckwalter produces a set of solutions for each token and the several analysis threads of the text need to be generated as a product of the token solutions.

Sarf is similar to Beesley (2001) in that it uses finite-state Automata (FSA). Beesley reports that the number of the Automata generated by a compiler from Xerox rules can not be controlled and also reports a difficulty to compose the FSAs into a single FSA framework. Sarf solves the problem with a non-deterministic composition of three FSAs that are the prefix, stem, and suffix FSAs. With Sarf it is easy to control the FSAs with a custom user case controller. Up to our understanding, Beesley requires recompiling the Xerox rules.

Sarf is similar to The MADA+TOKAN toolkit in that it thinks of the several valid morphological analyses as a richness that should be exploited at a higher abstract level rather than an inaccuracy that should be corrected. Sarf differs in that it provides an interface for the case study to interfere in judging the analyses much earlier and at a finer granularity level. Also in Sarf there is no need for a separate tokenizer such as TOKAN as each alive Automata carries with it the positions in text where a composition decision was made. The case-based controller can build the trace when needed.

We strongly feel that the refinement of SAMA (Maamouri et al., 2010) is the closest to our approach. SAMA was refined to interact with the ATB (Maamouri and Bies, 2004) project after the addition of a large new corpus. We find supporting evidence in this decision to our hypothesis about the utility of a case-based analyzer. The refinements that resulted in algorithmic changes in SAMA were done manually and worked in integration with the ATB format. Sarf is a case-based controller approach that can modify the behavior of the analyzer on the fly and can interact with the any case study. In Sarf, the analyzer remains general, and the case-based controller specializes the analysis.

The AMIRA (Mona Diab and Jurafsky, 2007) analyzer uses a language independent SVM based analysis. While the SVM analysis learns features from the text passed to the case study, it does not make use of the type of the query the case study targets in the text.

Like ElixirFM (Smrž, 2007) Sarf builds on the lexicon of the Buckwalter analyzer. We also use deterministic parsing with tries in our analysis to run the prefix, suffix and stem FSAs. We think that the inferential-realizational approach of ElixirFM that is highly compatible with the Arabic linguistic description (Badawi et al., 2004) can benefit from many features unique to the Arabic language. Sarf leaves implementing that to the case-based controller since in many cases the abstraction needed may not need the text to be grammatically correct.

3 Sarf

Sarf extends the lexicon of Buckwalter (2002) with proper names and location names extracted from different online sources [1] as well as biblical sources [2].

Sarf encodes the prefix, suffix and stem lexicons into three linear FSAs and uses a non-deterministic composition of the three FSAs to compute all valid morphological analyses. We first present a running example of Sarf. The running example explains how the non-deterministic composition works and illustrates how Sarf implements recursive affixes and handles "run-on" words. Then we discuss how Sarf builds and composes the different FSAs.

[1] http://alasmaa.net/ , http://ar.wikipedia.org/

[2] Genesis 4:17-23; 5:1-32; 9:28-10:32; 11:10-32; 25:1-4, 12-18; 36:1-37:2; Exodus 6:14-25; Ruth 4:18-22; 1 Samuel 14:49-51; 1 Chronicles 1:1-9:44; 14:3-7; 24:1; 25:1-27:22; Nehemiah 12:8-26; Matthew 1:1-16; Luke 3:23-38

3.1 Sarf example

The compositional FSA diagram in Figure 1 illustrates the operation of Sarf when parsing the string وسيلعبهاالّلاعبون. [3] The square and circle legends are states of the FSA and the directed edges are transitions driven by an input letter that matches the label of the edge. The square denotes an accept state, the circle denotes a regular state, and the reject states are omitted for clarity. Figure 1(a) represents the prefix FSA and Figures 1(b) and 1(c) represent the stem and suffix FSAs respectively. The symbol ϵ represents an empty string and is the source of non-determinism in this example.

We start at the root square state in the prefix FSA which is an accept state. The edge labeled ϵ connects the prefix FSA to the stem FSA and transitions from any accept state in the prefix FSA to the root state in the stem FSA. The edge that connects the stem FSA to the suffix FSA follows the same behavior.

When there are two valid transitions such as و and ϵ in the start case, the non-deterministic FSA Φ spawns an exact copy of itself Ψ. FSA Φ makes one transition, and FSA Ψ makes the other. Each of the FSAs now represent a valid analysis so far. When the FSA is at a state where the next input letter leads to a reject state, the FSA dies. The reject states are omitted from Figure 1 and are denoted by the absence of the corresponding letters. In our example, if we assume there were no stems that start with the letter و, Ψ will die then. In reality probably Ψ will die when the input string is at وسيلع.

Note that the prefix and affix FSAs allow recursive prefixes. For example, و will result in an accept state that transitions to the stem FSA. When س follows, we move to another accept state in the prefix FSA corresponding to the suffix وس. The same applies to the suffix FSA.

Lets consider FSA Φ after it consumed وسي and transitioned into the stem FSA root node. Now Φ will transition with لعب to reach an accept state. Before moving with the letter ب to the accept state, Φ needs to make sure that the stem لعب is compatible with the prefix وسي. It does that by associating a category value for each accept state. Sarf considers the value of the category as part of the state.

[3] و س ي ل ع ب ه ا ا ل ل ا ع ب و ن in separate form to ease following the example

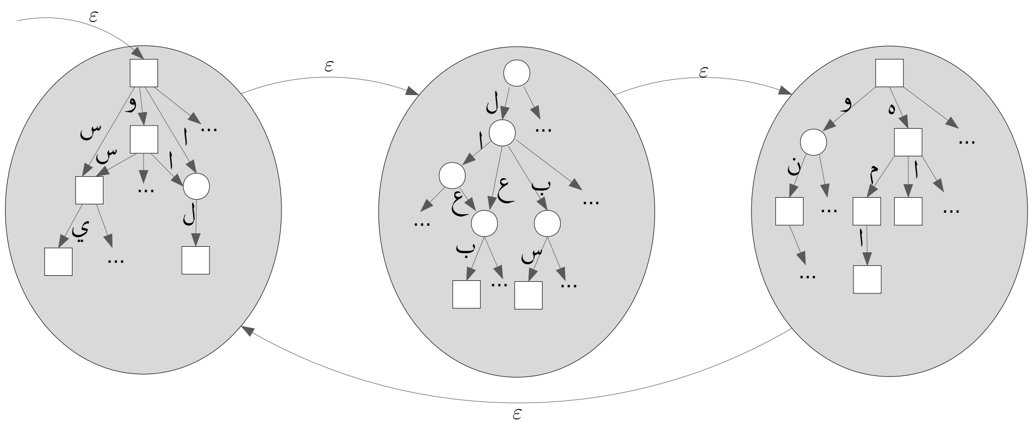

Figure 1: Example composition of the prefix, stem and suffix FSAs.

Thus each state shown in Figure 1 actually represents more than one state. If the category of لعب is compatible with the category of وسي then ب will move Φ to an accept state. Otherwise, it will move it to a regular or a reject state.

Since Φ is now in an accept state in the stem FSA, it will spawn Σ that transitions into the root of the suffix FSA. When ه is consumed, Φ will die since there is no edge from the current state labeled with ه . Now Σ represents a valid analysis and can proceed.

Note that Σ reports a full word at any accept state by spawning a new Automata using the ϵ transition to the prefix FSA. In Sarf, this only happens with white space, delimiters, and non-connecting letters. We left that out in Figure 1 for clarity.

Finally, the case-based controller can interfere at any point in the analysis to make a decision like ignoring an Automata that is not interesting. It may also decide to correct the analysis of one Automata based on the decision of the others.

3.2 Recursive affixes

Morphological analysis partitions a given Arabic word into a set of morphemes. A morpheme is either an affix or a stem. We call two morphemes compatible if their concatenation forms a legal morpheme. Buckwalter (2002) keeps separate lists of prefixes, suffixes and stems and assigns a category to each morpheme. The relation between morphemes and categories is one to many.

A compatibility rule is a pair of categories $\langle c_1, c_2 \rangle$ stating that morphemes with category c_1 can be concatenated with morphemes of category c_2. Buckwalter keeps three lists L_{ps}, L_{sx}, and L_{px} of compatibility rules relating prefixes to stems, stems to suffixes, and prefixes to suffixes respectively. Consider a string $s = \alpha\beta\gamma$ where α is a prefix, β is a stem, and γ is a suffix. The $\alpha\beta\gamma$ partition of the string is a valid morphological analysis if and only if we have $\langle c(\alpha), c(\beta) \rangle \in L_{ps}$ and $\langle c(\beta), c(\gamma) \rangle \in L_{sx}$ and $\langle c(\alpha), c(\gamma) \rangle \in L_{px}$ where $c()$ returns the category of the morpheme.

Many affixes are composed of other affixes. For example, the prefix وسي is composed of three other prefixes namely و, س, and ي. The POS tag for وسي is a concatenation of the POS tags of each of the three morphemes. Keeping these morphemes separate in a list is redundant and may produce consistency issues. We add two lists of compatibility rules L_{pp} and L_{xx} for prefix-prefix and suffix-suffix compatibility respectively. This allows us to keep a shorter list of affixes. We need one more additional modification to accommodate a category for the morphemes accepted by the L_{pp} and the L_{xx} rules. For that we introduce the concept of a *resulting category*. The resulting category in all practical cases is that of the second morpheme in the pair.

3.3 Affix linear FSA

We analyzed the prefixes and suffixes of Buckwalter (2002) and extracted our refined lists of recursive affixes. We encoded the lists into an FSA similar to those in Figure 1(a) and Figure 1(c). The two FSAs are directed acyclic graphs. The absence of cycles in these FSAs reduces the traversal of the affix FSAs to a linear traversal. This is computationally superior to the approach of Buckwalter where the analyzer considers all possible substrings of the word in question and looks it up in the affix tables.

The prefix and suffix FSAs encode the L_{pp} and L_{xx} compatibility lists. An accept state in the FSA corresponds to the last letter of a prefix in the prefix and suffix lists. The accept states hold category state information as well as POS and other tags.

3.4 Stem linear FSA

We built our stem lexicon using the stem lexicon of Buckwalter. We augmented the lexicon with a set of proper names and a set of location names. We obtained the set of proper names from online and biblical sources.

The stems share lots of substrings. We encoded them in an efficient double array trie structure (Aoe, 1989). The structure is also a linear FSA where the accept states are the terminal nodes corresponding to the last letters in stems. This approach is superior to Buckwalter since it consists of a linear walk in the trie while with Buckwalter we need a number of hash lookups in the order of all possible partitions of the string.

3.5 Non-deterministic composition of FSAs

The diagram in Figure 2 shows an abstraction of the non-deterministic composition of the prefix, stem and suffix FSAs. The circle represents regular states, the square represents accept states and the triangle represents reject states. Note that the concrete prefix, stem and suffix FSAs are all linear and do not have cycles as discussed is Sections 3.4 and 3.3. The cycles are introduced in the abstract machine for brevity and illustrative purposes.

```
NDSarf algorithm

NDSarf(string $L$, Controller $A$)
  FSAVec $M$; -- collection of FSA Automata
  FSA $\Phi_1$;
  $M$.insert($\Phi_1$);
  foreach $c$ in $L$ {
    foreach $\Phi$ in $M$ {
      if ( $\Phi$.state.isAccept() ) {
        if ($\Phi$.isSuffix())
          if ($c$.isWhite()) or
            $A$.report();
          if ($\Phi$.lastChar().isNonConnecting())
            $A$.report();
        $\Psi$ = $\Phi$.clone();
        $M$.insert($\Psi$); }

      if ( $\Phi$.isWalkable($c$) ) {
        $\Phi$.transition($c$);
      } else {
        $M$.remove($\Phi$);
        $\Phi$.die();
      } }
    $A$.control($M$, $c$); }
```

The algorithm NDSarf below controls Sarf. It takes as input a text string L and a case-based controller A. It starts with one FSA Φ_1 similar to the one in Figure 2 and inserts it into a collection of FSAs M. The algorithm reads a letter c at a time from L and iterates over all machines in M. For each Φ in M, if Φ is in an accept state and it is in the suffix phase, then the algorithm checks whether it should report a full word. This happens when c is a white space or a delimiter, or when the last letter leading to the current state was a non-connecting letter.

In all cases, if Φ is in an accept state, it spawns another FSA Ψ and adds it to M. If c leads to a reject state, then Φ dies and we remove it from M. Otherwise, Φ transitions using the c edge. After iterating over all FSAs in M, we invoke the controller A and grant it full access and control over all FSAs in M.

The arrows in Figure 2 that transition from prefix to stem, stem to suffix, and suffix to prefix require compatibility constraints for the transition.

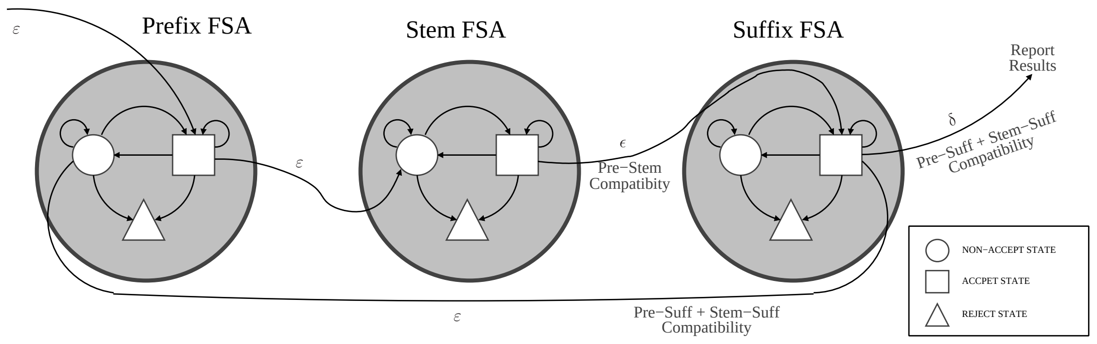

Figure 2: An abstraction of the non-deterministic composition of the prefix, stem and suffix FSAs.

4 Islamic Literature Case Study

A hadith in Islamic literature is a narration from the prophet Mohammad related by multiple narrators. Establishing the authenticity of a hadith is an important task in Islamic studies. In this paper we use Sarf to successfully automate the analysis of three books of hadith selected arbitrarily (Ibn Hanbal, ; Al Kulayni, ; Al Tousi,)[4]

Our analysis accepts a book as input and segments it into a vector of hadiths. We segment each individual hadith into two parts. The first part is the sanad and it contains the chain of narrators who related the hadith from Mohammad. The second part is the matn or the content of the hadith. We are concerned to further explore the sanad and we detect the chain of narrators and the relation that links each narrator to his ancestor and his predecessor in the chain. We also detect the full name of each narrator that is usually composed of several proper names with connectors in between.

We built our hadith case-based controller to target the detection of proper names. This includes the task of finding compound names composed of several words such as عبد الرحمن often appearing as "run-on" words. The controller also targets words that mean narrate when they appear in the neighborhood of multiple proper names.

4.1 Controller

We illustrate the hadith case-based controller FSA in Figure 3. The transitions in the hadith FSA are excited by inputs from the Sarf morphological analyzer. A NAME label means that Sarf encountered a proper name. An NRC label means that Sarf met a narrator connector such as عن ”on behalf of”, حدث ”narrated”, قال ”said”, أخبر ”told” or one of their derivations. The IBN label corresponds to the word ابن ”the son of” and is a name connector. The NISBA label corresponds to an adjective that points to a person such as المصري ”the Egyptian”.

[4]We obtained the digitized books from online sources such as http://www.yasoob.com/ and http://www.al-eman.com/.

The symbol τ_{NMC} is a threshold that corresponds to the number of tolerated name connectors that may occur between two names. The symbol LIST_{NMC} corresponds to the list of name connectors collected since the controller started looping in the state NMC_S. The symbol λ_{NMC} is a parameter that corresponds to a relaxed tolerance measure that the controller resorts to in case the words separating two names were longer than τ_{NMC} but contained a name connector word such as IBN or NISBA.

The controller has four states that correspond to an abstract position in text relative to the next sanad. State TEXT_S is the initial state and denotes that the controller is outside the context of a sanad. The controller moves to the state NAME_S whenever NAME is reported by Sarf. State NRC_S means that the controller thinks it is in the context of a narrator name after Sarf reports an NRC. State NMC_S indicates that the controller expects a name to appear within a tolerance threshold expressed by τ_{NMC} and λ_{NMC}.

Similarly, the NRC_S state tolerates τ_{NRC} words before it gives up on its expectations.

We reach the NAME_S state only when a valid NAME is detected and we leave when no more NAME's are detected. The NRC_S state can only be reached if an NRC is detected. NMC_S can only be reached from a NAME_S state.

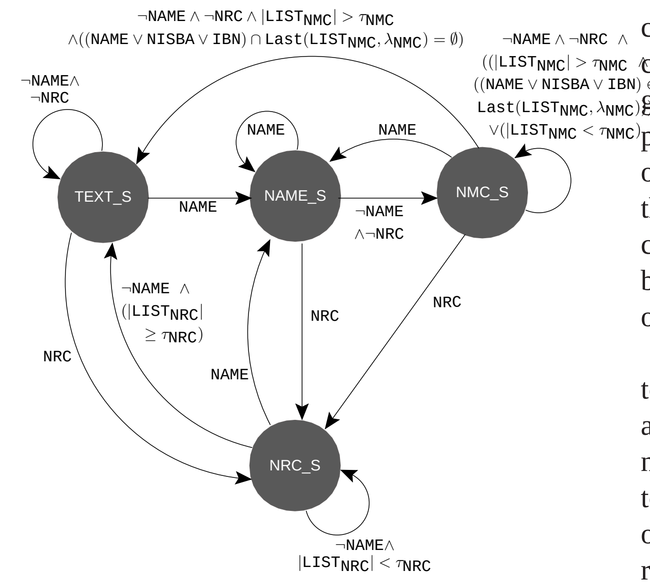
$\neg$NAME $\wedge$ $\neg$NRC $\wedge$ $|$LIST$_{NMC}| > \tau_{NMC}$
$\wedge$((NAME $\vee$ NISBA $\vee$ IBN) $\cap$ Last(LIST$_{NMC}$, λ_{NMC}) $= \emptyset$)
$\neg$NAME $\wedge$ $\neg$NRC $\wedge$
(($|$LIST$_{NMC}| > \tau_{NMC}$ $\wedge$
((NAME $\vee$ NISBA $\vee$ IBN) $\in$
Last(LIST$_{NMC}$, λ_{NMC}))
$\vee$($|$LIST$_{NMC}| < \tau_{NMC}$)
$\neg$NAME$\wedge$
$\neg$NRC
NAME
NAME
TEXT_S
NAME
NAME_S
NMC_S
$\neg$NAME
$\wedge\neg$NRC
$\neg$NAME $\wedge$
($|$LIST$_{NRC}|$
$\geq \tau_{NRC}$)
NRC
NRC
NRC
NAME
NRC_S
$\neg$NAME$\wedge$
$|$LIST$_{NRC}| < \tau_{NRC}$

Figure 3: FSA controller of the hadith case study.

The FSA reports a valid chain of narrators when a sequence of names connected by narrator connectors appears. It marks the beginning of the hadith with the beginning of the current sequence, and marks the end of the hadith with the beginning of the next sequence. It also marks the chain of narrators as the sequence itself.

The definition of input labels such as NAME and IBN depends on the morphological analyzer. However, our case-based controller approach was able to perform well under both Sarf and a refined version of Sarf.

5 Results

To evaluate our hypothesis of the utility of a case-based morphological analyzer, we ran our case study in two modes, simple and refined. In the simple mode, Sarf is controlled by the hadith controller. In the refined mode, we apply several refinements to Sarf. The first refinement removes female proper names since they do not occur often in hadith. We also add a refinement that detects compound names as they are common in narrator names. Finally, we consider a possessive form as a reference to a person and treat it as a narrator.

We report our results in Table 1 on the three hadith books. The word count row is an indicator of the size of data. The number of names, narrators, and chains reflect a hierarchical structure where a chain contains narrators, and a narrator contains names. In general, we observe that the refinements slightly improved performance and accuracy at the lowest level of abstraction by reducing false positives. However, the refinements had little to no effect on the high accuracy measures at higher levels of abstraction. We believe that this is a strong supporting evidence to our hypothesis.

The names row reports the total number of detected names. The names/narrator row reports the average number of names per narrator structure. The narrators row reports the number of narrators detected. The narrators/chain row reports the number of narrators per chain structure. And the chain row reports the number of detected chains in the book. The ignored names row reports the number of names detected outside the chains and thus irrelevant to the case study. The segmentation accuracy reports on the accuracy of segmentation of the book into separate hadiths. The metric is computed manually over a representative sample of each text. The same applies to all other accuracy measures. The chain accuracy reflects the percentage of correct narrators covered by extracted chains. The narrator name accuracy row reflects the percentage of correct proper names detected within narrator structures.

The name false positives row reports the percentage of words recognized as names while they should not. We notice that the low level metric of name false positives improved dramatically with the refined analysis without substantially affecting the high abstract measures such as chain and segmentation accuracy. We notice that the controller in the simple analysis ignored more names and thus the case-based controller compensated for the inaccuracy of the low level analysis with a simple heuristic and without the need of the fancy refinements.

We investigated the case of Al Kafi where the accuracy of the simple analysis was significantly lower than that of the refined analysis. We discovered that the chains detected by the refined analysis were a subset of those detected by the simple analysis. The reason for the lower accuracy is false positive chains. The simple analysis did not miss important information. We hypothesize that a higher level of abstraction such as intersecting the narrators from the hadith books with the narrators in the biographies can

Table 1: Results of the hadith case study with Sarf.

	Al Kafi		Al Istibsar		Ibn Hanbal	
	Simple	Refined	Simple	Refined	Simple	Refined
Word count	98,943		103,835		20,354	
Names	20,473	12,060	20,646	14,613	4,762	3,013
Names/Narrator	2.48	1.97	2.17	1.84	1.49	1.25
Narrators	3,080	2,623	6,164	5,767	2,082	1,755
Narrators/Chain	5.28	4.84	4.98	4.76	4.8	4.05
Chains	583	542	1,238	1,211	433	433
Ignored names	11,801	6,400	5,845	3,348	1,386	642
Segmentation accuracy	85%	96%	96%	96%	92%	92%
Chain accuracy	99%	99%	98%	99%	99%	97%
Narrator name accuracy	93%	91%	92%	90%	90%	90%
Name false positives	28%	7%	9%	4%	34%	4%
Sarf running time (secs.)	24.5	22.67	23.5	23.17	2.25	1.67

reduce the false positives in the chains.

6 Future Work

In the future, we plan to complete the hadith case study and automate narrator discovery and authentication in the biography books. We plan to make use of the name connector and narrator connector stop phrases that our case study extracted to create a learning system that can learn more names. We also plan on developing a domain specific language similar to ElexirFM (Smrž, 2007) that allows users to easily specify their case-based controller to interact with Sarf.

References

[Al Kulayni] Muhammad ibn Yaaqub Al Kulayni. *Kitab al-Kafi*.

[Al-Sughaiyer and Al-Kharashi2004] Imad A. Al-Sughaiyer and Ibrahim A. Al-Kharashi. 2004. Arabic morphological analysis techniques: a comprehensive survey. *American Society for Information Science and Technology*, 55(3):189–213.

[Al Tousi] Mohammad bin Hasan bin Ali Al Tousi. *Al Istibsar*.

[Aoe1989] Jun-ichi Aoe. 1989. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, 15(9):1066–1077, May.

[Badawi et al.2004] Elsaid Badawi, Mike G. Carter, and Adrian Gully. 2004. *Modern Written Arabic: A Comprehensive Grammar*. Routledge, New York.

[Beesley2001] Keneth R. Beesley. 2001. Finite-state morphological analysis and generation of arabic at xerox research: Status and plans. In *Workshop Proceedings on Arabic Language Processing: Status and Prospects*, pages 1–8, Toulouse, France.

[Benajiba et al.2008] Yassine Benajiba, Mona Diab, and Paolo Rosso. 2008. Arabic named entity recognition using optimized feature sets. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 284–293, Morristown, NJ, USA.

[Buckwalter2002] Timothy Buckwalter. 2002. Buckwalter arabic morphological analyzer version 1.0. Technical report, LDC catalog number LDC2002L49.

[Buckwalter2004] Timothy Buckwalter. 2004. Issues in arabic orthography and morphology analysis. In *Semitic '04: Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages*, pages 31–34, Morristown, NJ, USA.

[Darwish2002] Kareem Darwish. 2002. Building a shallow arabic morphological analyzer in one day. In *Proceedings of the ACL-02 workshop on Computational approaches to semitic languages*.

[Habash and Sadat2006] Nizar Habash and Fatiha Sadat. 2006. Arabic preprocessing schemes for statistical machine translation. In *Proceedings of the North American Chapter of the Association for Computational Linguistics(NAACL)*, pages 49–52.

[Habash et al.2005] Nizar Habash, Owen Rambow, and George Kiraz. 2005. Morphological analysis and generation for arabic dialects. In *Semitic '05: Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 17–24, Morristown, NJ, USA.

[Habash et al.2009] Nizar Habash, Owen Rambow, and Ryan Roth. 2009. Mada+tokan: A toolkit for arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization. In Khalid Choukri and Bente Maegaard, editors, *Proceedings of the Second International Conference on Arabic Language Resources and Tools*, Cairo, Egypt, April. The MEDAR Consortium.

[Ibn Hanbal] Ahmad Ibn Hanbal. *Musnad Ahmad Ibn Hanbal*.

[Khoja2001] Shereen Khoja. 2001. Apt: Arabic part of speech tagger. In *NAACL student research workshop*.

[Kulick et al.2010] Seth Kulick, Ann Bies, and Mohamed Maamouri. 2010. Consistent and flexible integration of morphological annotation in the arabic treebank. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May.

[Maamouri and Bies2004] Mohamed Maamouri and Ann Bies. 2004. Developing an arabic treebank: methods, guidelines, procedures, and tools. In *Semitic '04: Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages*, pages 2–9.

[Maamouri et al.2010] Mohamed Maamouri, Ann Bies, Seth Kulick, Wajdi Zaghouani, David Graff, and Michael Ciul. 2010. From speech to trees: Applying treebank annotation to arabic broadcast news. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May.

[Mona Diab and Jurafsky2007] Kadri Hacioglu Mona Diab and Daniel Jurafsky, 2007. *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, chapter 9, pages 159–179. Springer.

[Smrž2007] Otakar Smrž. 2007. Elixirfm: implementation of functional arabic morphology. In *Semitic '07: Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages*, pages 1–8, Prague, Czech Republic.