AMERICAN UNIVERSITY OF BEIRUT

Morphology-Based Entity and Relational Entity Information Extraction Framework for Arabic

by
AMEEN ALI JABER

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master in Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
January 2014

AMERICAN UNIVERSITY OF BEIRUT

Morphology-Based Entity and Relational Entity Information Extraction Framework for Arabic

by

AMEEN ALI JABER

Approved by:

Dr. Fadi Zaraket, Assistant Professor — Advisor
Electrical and Computer Engineering

Dr. Louay Bazzi, Associate Professor — Member of Committee
Electrical and Computer Engineering

Dr. Mohamad Jaber, Assistant Professor — Member of Committee
Computer Science

Fadi Zaraket on behalf of Mohamad Jaber

Date of thesis defense: January 2nd, 2014

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

I, AMEEN ALI JABER

□ authorize the American University of Beirut to supply copies of my thesis to libraries or individuals upon request.

□ do not authorize the American University of Beirut to supply copies of my thesis to libraries or individuals for a period of two years starting with the date of the thesis deposit.

Signature

Date

To my parents – Ali and Ibtisam

Acknowledgments

بِسْمِ اللهِ الرَّحْمنِ الرَّحِيمِ
وَقُل رَب زِدْنِى عِلْماً
"And say: My Lord increase me in knowledge."
(Qur'an, Ta-Ha 20:114)

Accordingly, I'm heartily thankful to God for His overwhelming blessings in planting the seeds of patience and knowledge within my thriving self to reap this awe-inspiring success.

Words cannot express my solemn appreciation to my family; my father, my mother, my sisters, my brothers, and my fiancé. Your sincere prayers and kind support boosted my ambitions and made this dream come true.

My special appreciation goes to my advisor, Prof. Fadi Zaraket, without whom this achievement would have never been the same. Thank you for being a great mentor and a caring inspirational figure; your priceless advice helped me tackle most of the difficulties with ease and joy.

Finally, I would like to thank all my friends and colleagues for coloring my days at AUB and engraving lovely memories to be remembered forever. Thank you all.

An Abstract of the Thesis of

Ameen Ali Jaber for Masters of Engineering
Major: Electrical and Computer Engineering

Title: Morphology-Based Entity and Relational Entity Information Extraction Framework for Arabic

Natural Language Processing is concerned with automating the understanding of natural language. *Morphological analysis* is key to Arabic natural language processing due to the morphological richness of Arabic. Researchers proposed and evaluated *knowledge-based* and *empirical* techniques to extract *entities* and *relations* from text. Knowledge-based techniques require advanced linguistic and programming expertise. Empirical techniques require large training and reference corpora to learn and evaluate computational models respectively.

In this work, we present a morphology-based entity and relational entity information extraction framework for Arabic text. The framework provides a user-friendly interface where the user defines tag types and associates them with regular expressions defined over Boolean formulae. Boolean formulae are terms, negations of terms, and disjunctions of terms where terms are matches to Arabic morphological features. The framework introduces a semantic feature that relates words based on synonymity. The framework allows the user to associate an action with each regular sub-expression and to define semantic relations. The framework uses an in-house Arabic morphological analyzer to compute morphological matches, computes regular expression matches, and then builds the relations across matches. We evaluated our work with several case studies and compared with existing application-specific techniques.

Contents

Contents **viii**

1 Introduction **1**

2 Preliminaries **6**

2.1 Morphological features 6
2.2 Finite state transducers 7
2.3 Morphological analyzer 8
2.4 Classes, labels, and tag types 11

3 Motivation **12**

4 Overview **17**

5 MERF **21**

5.1 Syn^k 21
5.2 Morphology-based atomic terms (MAT) 23
5.3 Morphology-based Boolean formula (MBF) 24
5.4 Morphology-based regular expression (MRE) 25
5.5 Computational actions 26
5.6 Tag type 28
5.7 MBF evaluation 28
5.8 MRE and action simulation 28
5.9 Semantic relations 30

6 Implementation **32**

6.1 Data Model 32
6.1.1 Tag file 32
6.1.2 Tag type file 33
6.2 Data structures 34
6.2.1 Arabic document 35
6.2.2 MBF tag types 35
6.2.3 MRE tag types 35
6.2.4 MBF tags 36

6.2.5 MRE tags 36
6.3 MBF simulation 37
6.4 MRE simulation 37
6.4.1 NFA generation 37
6.4.2 NFA simulation 38
6.5 Code action execution 38
6.5.1 Action file generation 38
6.5.2 Action file execution 39
6.6 MERF interface 39
6.6.1 Tree and graph visualization 40
6.7 Open source tool 40

7 MERF GUI **41**
7.1 Tag type Boolean formula editor 42
7.2 MERF MBF match visualization 43
7.3 Tag type regular expression editor 44
7.4 MERF MRE match visualization 45
7.5 Semantic relation editor 46
7.6 Code action editor 47
7.7 Analysis 48

8 Case Studies **49**
8.1 Narrator chain 50
8.2 Number normalization 52
8.3 Temporal entity extraction 54

9 Related Work **56**
9.1 Information extraction 56
9.2 WordNet 57
9.3 Tagging 58

10 Conclusion **60**
10.1 Future work 60

List of Figures **61**

List of Tables **63**

Bibliography **64**

Appendices **69**

A **70**

A.1 Economic analysis 70
A.2 Prayer times 71
A.3 Football results 72
A.4 Geographical information 72
A.5 Official notifications 74
A.6 Professions information 76
A.7 Money 77
A.8 Crop information 77

Chapter 1

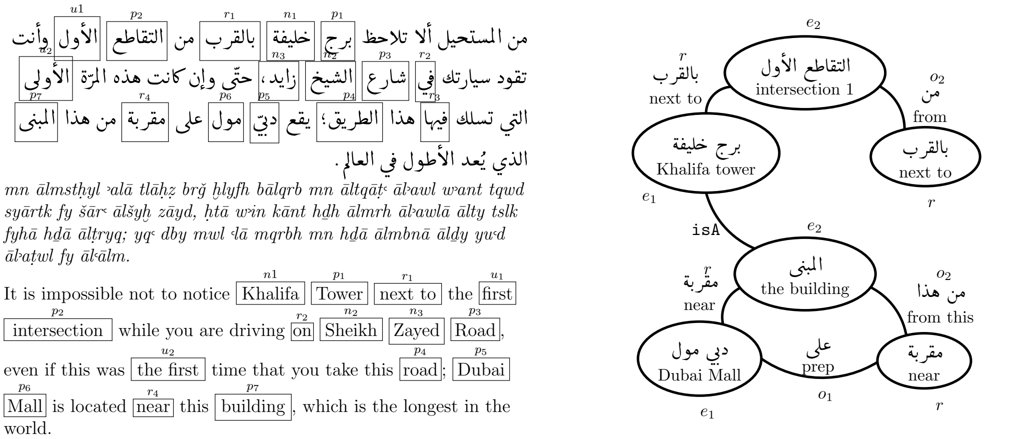
Introduction

Computational Linguistics (CL) is concerned with building accurate linguistic computational models. *Natural Language Processing* (NLP) is concerned with automating the understanding of natural language. CL and NLP tasks range from simple tasks such as spell checking and typo-error correction to more complex tasks including text summarization, *named entity recognition* (NER), *cross-document analysis*, *machine translation*, and *entity-relational information extraction* [1, 2]. These tasks take as input digital text documents from sources including literature, books, news, business reports, chat messages, and emails. Some documents are originally typed in digital format such as emails. Other documents are automatically digitized using techniques such as *optical character recognition* (OCR) and *automatic speech recognition* [3, 4].

Entities are elements of text that are of interest to an NLP task. *Relational entities* are elements of text that connect entities. *Annotations* (referred to as *tags* in the sequel) relate a chunk of text to a *label* (a *tag type*) denoting a semantic value such as an entity or a relational entity.

In this work, we address entity and relational entity extraction from Arabic text using morphological analysis. For example, given the text in Figure 1.1(a), we would like to extract entities and relations that form the graph in Figure 1.1(b). The text in Figure 1.1(a) contains directions to Dubai Mall taken from the mall website [1]. The framed words in the text are target entities referring to names of people, names of places, relative position, and numerical terms. From those entities, we extract the graph shown in Figure 1.1(b) where vertices express entities, and edges represent the relations between those entities. This task is accomplished in four phases that include morphological analysis, entity extraction based on morphological features, relation construction, and entity cross-reference.

[1] http://www.thedubaimall.com/ar/



من المستحيل ألا تلاحظ برج خليفة بالقرب من التقاطع الأول وأنت تقود سيارتك في شارع الشيخ زايد، حتّى وإن كانت هذه المرّة الأولى التي تسلك فيها هذا الطريق؛ يقع دبيّ مول على مقربة من هذا المبنى الذي يُعد الأطول في العالم.

mn ālmstḥyl ʾalā tlāḥẓ brǧ ḫlyfh bālqrb mn āltqāṭʿ ālʾawl wʾant tqwd syārtk fy šārʿ ālšyḫ zāyd, ḥtā wʾin kānt hḏh ālmrh ālʾawlā ālty tslk fyhā hḏā ālṭryq; yqʿ dby mwl ʿlā mqrbh mn hḏā ālmbnā ālḏy yuʿd ālʾaṭwl fy ālʿālm.

It is impossible not to notice Khalifa Tower next to the first intersection while you are driving on Sheikh Zayed Road, even if this was the first time that you take this road; Dubai Mall is located near this building, which is the longest in the world.

(a) Text with directions

(b) Formula, matches, and entity-relation graph

Figure 1.1: Text, formula, and match MERF example

Morphological analysis

The Arabic language is morphologically rich. Short vowels, also known as diacritics, are almost always omitted in Arabic text and inferred by human readers [5]. For example, the word أسد *ʾsd* [2] can be interpreted as أَسَد *ʾsad* `‘‘lion’’` with a *fatha* on the letter س *s* or أُسُدّ *ʾsodd* (I block) with a *damma* on the letter س *s* and *shadda* on the letter د *d* . Consequently, *morphological analysis* is key to Arabic CL and NLP even for simple tasks such as tokenization and stemming[6]. Tokenization requires morphological analysis in Arabic because a subset of the Arabic letters are non-connecting letters and do not require a space after them to provide a visual separation from the next letter. For example, in the sentence ذهب الولدإلى المدرسة *ḏhb ālwldʾilā ālmdrsh* (the kid went to the school) the letter ى is non-connecting and the two words إلى and الولد are visually separable, yet there is no space character between them.

Given an Arabic word delimited by white space and punctuation, a *morphological analyzer* returns the internal structure of the word. The word structure is composed of several morphemes including *affixes* (*prefixes* and *suffixes*), and *stems* [6]. The analyzer returns a set of morphological solution vectors with features such as *prefix, stem, suffix*, and part of speech (POS), gloss, and category tags. For example, for the word فسيلعبون *fsylʿbwn* , the analyzer may return فسيـ *fsy* as a prefix morpheme with the POS tag `fa/CONJ+sa/FUT+ya/IV3MD` and with gloss tag `and + will + they (both)`, لعب *lʿb* as a stem with POS tag

[2] In this document, we use the default ArabTeX transliteration style ZDMG.

loEab/VERB_IMPERFECT and with gloss tag ``play'', and ـون *wn* as a suffix with POS tag uwna/IVSUFF_SUBJ:MP_MOOD:I and with gloss tag [MASC.PL.].

Knowledge-based entity extraction

Researchers proposed and evaluated empirical and knowledge-based techniques to extract entities and relations from text. Knowledge-based techniques target a task based on solid linguistic or structural grounds [7]. Knowledge-based techniques such as [8, 9] propose local grammars with morphological stemming to perform NER. The work in [10] presents a method for extracting entities, events, and relations amongst them from Arabic text using a hierarchy of manually built finite state machines driven by morphological features and graph transformation algorithms. Such techniques require advanced linguistic and programming expertise. Our method provides a user-friendly interface that enables an average user to define target entities and relations.

Empirical entity extraction

Supervised and unsupervised empirical techniques employ machine learning techniques to automatically extract entities without the need to manually encode the requisite knowledge [7]. Supervised learning techniques require training text that is annotated with correct tags to learn a computational model. Supervised and unsupervised techniques require reference text that is annotated with correct tags to evaluate the accuracy of the technique in terms of metrics such as precision and recall [11, 12, 13]. The work in [14] presents a language independent approach for NER extraction using *support vector machines*. The work in [15] integrates a semi-supervised bootstrapping pattern recognition technique, and a supervised classifier based on *conditional random fields* to solve NER problems. Our method enables the user to incrementally create complex annotations for Arabic text based on automatic extraction of morphological tags through a user friendly interactive interface.

MERF

In this work, we present a *morphology-based entity and relational information extraction framework for Arabic text* (MERF). MERF provides a user-friendly interface where the user defines tag types and associates them with MERF formulae that are regular expressions over MERF Boolean formulae. Boolean formulae are terms, negations of terms, and disjunctions of terms. Terms are matches to Arabic morphological features including prefix, stem, suffix, POS tags, gloss tags, and semantic categories. In addition to the morphological features, MERF introduces Syn^k as a feature that relates two words w_1 and w_2 iff w_1 is a synonym of w_2 or w_1 is a Syn^{k-1} with one of the synonyms of w_2. Consider the example shown in Figure 1.2. Given the Arabic words طعام*ṭʿām* , أكل*ʾkl* , and أتعب*ʾtʿb* and their glosses {food}, {food, eat, make tired}, and {make tired, bother, drink} respectively. We can relate طعام*ṭʿām* to أكل*ʾkl* based on the gloss intersection `food`. Moreover, we can relate طعام*ṭʿām* to “أتعب*ʾtʿb* ” since أكل*ʾkl* and أتعب*ʾtʿb* have the gloss intersection `make tired`.

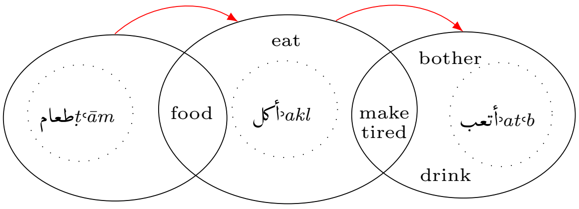


Figure 1.2: Syn^2(طعام*ṭʿām*)

MERF regular expressions support operators such as concatenation, zero or one, zero or more, one or more, up to M, and logical conjunction and disjunction operations. MERF editor allows the user to associate an action with each MERF sub-expression. The user specifies the action with C++ code and uses the MERF API to access information related to the matches such as text, position, length, morphological features, and numerical values.

MERF takes an Arabic text, a set of user-defined MERF Boolean formulae, and a set of user-defined MERF regular expressions. MERF computes the morphological solutions of the words in the input text then computes matches to the Boolean formulae. MERF then generates a *non-deterministic finite state automata* (NDFSA) for each expression and simulates it with the sequence of Boolean formulae matches to compute the regular expression matches. MERF generates, complies, links, and executes the actions of the corresponding regular expressions as shared object libraries. Finally, MERF constructs the semantic relations and cross-reference between entities. MERF also provides visualization tools to present the matches, and estimate their accuracy with respect to reference tags.

MERF has the following advantages:

Advantage 1. MERF provides a novel and intuitive visual interface to build Boolean formulae over morphological features, build regular expressions over the resulting Boolean formulae, and thereafter compute automatic tags.

Advantage 2. Up to our knowledge, MERF is the first morphology-based framework for Arabic entity and relational entity extraction.

Advantage 3. MERF provides the user with the ability to rapidly create annotated Arabic text corpora with sophisticated morphology-based tags.

In MERF, we make the following contributions:

Contribution 1. MERF enables the user to define semantic relations and automatically construct matches of these relations.

Contribution 2. MERF enables the user to associate code actions with subexpressions with API access to match features including text, position, length, numerical value, and morphological features.

Contribution 3. MERF enables the user to tag words based on a synonymic relation using the Syn^k feature.

Related work

Researchers proposed systems for automatic information extraction based on user specifications. CPSL is a common pattern specification language for finite-state grammar [16]. It defines three sections for declaration, rule definition, and macros. MERF is an extension to CPSL with action execution and relation construction. The work in [17] presents SystemT, a system based on an algebraic Approach to Declarative information extraction (IE). It uses a declarative rule language and an optimizer. TEXTMARKER is a rule-based IE system designed to extract structured data from text [18]. The work in [19] presents a user-driven relational model targeting entity-relation extraction. In this model, the user enters a natural language query. QARAB is a question answering system supporting the Arabic language [20]. It takes Arabic natural language query and attempts to provide short answers for it. We discuss related work and compare to it in details in Chapter 9.

Chapter 2

Preliminaries

In this Chapter we define and discuss terms important to our method such as morphological features, finite state transducers, morphological Analyzer, classes, labels, and tag types.

2.1 Morphological features

Arabic is a morphologically rich language. Due to the rich Arabic morphology, a single Arabic word might replace a complete sentence in English. For example, the word يَأْكُله *yaʾakulh* stands for the English statement "he/it is eating him/it".

Form-based morphological analysis decomposes an Arabic word into several morphemes [21]. A *morpheme* is the smallest linguistic unit that has a meaning and fulfills a grammatical function. A morpheme can be a *stem*, or an *affix*. Each morpheme is associated with other morphological features including *POS*, *gloss*, *lemma*, and *category* tags.

A stem can be a *templatic* or *non-templatic* stem. Templetic stems are formed from roots using templetic morphemes such as كاتب *kātb* (writer) which is formed from كتب *ktb* (write). Non-templatic stems tend to be foreign names such as واشنطن *wāšnṭn* (Washington). A root is a sequence of three, four, or rarely five letters which signifies some abstract meaning. We denote the set of all stems by the symbol $\mathcal{S}$.

An affix can be a *prefix*, a *suffix*, or an *infix*. Prefixes attach before the stem and a word can have multiple prefixes. We denote the set of all prefixes by the symbol $\mathcal{P}$. Suffixes attach after the stem and a word can have multiple suffixes. We denote the set of all suffixes by the symbol $\mathcal{X}$.

The part-of-speech tag, referred to as POS, assigns a morpho-syntactic tag for a morpheme. Sample POS tags are `NOUN` and `VERB_PERFECT`. We denote the set of all glosses by the symbol POS.

The gloss is a brief notation of the semantic meaning of a morpheme in English. A word might have multiple glosses attached to it as it could stand for multiple meanings. We denote the set of all glosses by the symbol $GLOSS$.

The lemma is a conventionalized choice of one of the word forms to stand for the set of all words with one same core meaning. For example, the words بيت*byt* (house), للبيت*llbyt* (for the house), and بيوت*bywt* (houses) are represented by the masculine singular form noun بيت*byt* .

The category is a user defined tag that includes several morphemes of one kind. For example, the user can define a temporal category to include the prefix س*s* (will) and the time unit ساعة*sāʿh* (hour). We denote the set of all categories by the symbol CAT.

Consider the following word يَأْكُله*yaʾakulh* as a complete example. The word is composed of the three morphemes يَ*ya* , أْكُل*ʾakul* , and ه*h* . Each morpheme is associated with morphological features as shown in Table 2.1. The `IV3MS` POS tag indicates a third person masculine singular subject pronoun attached to a verb, and the `IVSUFF_DO:3MS` POS tag indicates a third person masculine singular pronoun attached as an object to an action verb. The `VERB_IMPERFECT` tag indicates an imperfect verb. The notation for the gloss and POS tags is taken from the Buckwalter morphological analyzer [22].

Table 2.1: Sample solution vector for the text يَأْكُله*yaʾakulh* .

	Prefix	**Stem**	**Suffix**
Data	يَ*ya*	أْكُل*ʾakul*	ه*h*
POS	IV3MS+	VERB_IMPERFECT	IVSUFF_DO:3MS
Gloss	he/it	eat/consume	him/it

2.2 Finite state transducers

A *finite state transducer* (FST) is a finite state machine with an input and an output tape. FSTs differ from *finite state automata* (FSA) in that they have an output tape while FSAs have accept states instead. Formally, an FST is a tuple $M = (S, S_0, \sigma, \mathit{\Gamma}, \delta)$ where S is the set of states, $S_0 \subset S$ is the set of initial states, σ is the input alphabet, $\mathit{\Gamma}$ is the output alphabet, and $\delta \subseteq$

$S \times (\sigma \times \{\epsilon\}) \times (\Gamma \times \{\epsilon\}) \times S$ is the transition relation. FSTs have been used extensively in text mining applications where the input is the text and the output is the delimiters of a chunk of text with an associated class [23]. FSTs are attractive in NLP tasks due to their efficiency and ease of use.

2.3 Morphological analyzer

An Arabic morphological analyzer takes a word in Arabic and returns a set of morphological solutions. Each solution splits the word into its morphemes and associates each morpheme with corresponding tags. Multiple solutions can be the result of multiple valid segmentations of the word into morphemes, or the multiple possible tags associated with a morpheme.

MERF is integrated with *Sarf*, an in-house open source Arabic morphological analyzer based on finite state transducers [24]. The contribution of Sarf over previous work is that it considers the Arabic affixes as agglutinative, i.e. composed of one or more morphemes. Hence, it introduces fusional compatibility rules for affix-affix concatenations. This contribution results in a lexicon which is smaller in size, has less redundancy, and resolves a number of inconsistencies. Figure 2.1 shows the finite state machine of Sarf. Boxes denote legal affixes and stems, and circles denote regular nodes. The edges are transitions and the labels correspond to the input letters. ϵ represents an empty string and is the source of non-determinism. The transitions between the prefix $\mathcal{P}$, stem $\mathcal{S}$, and suffix $\mathcal{X}$ sub-machines are non-deterministic to compute all valid morphological analyses.

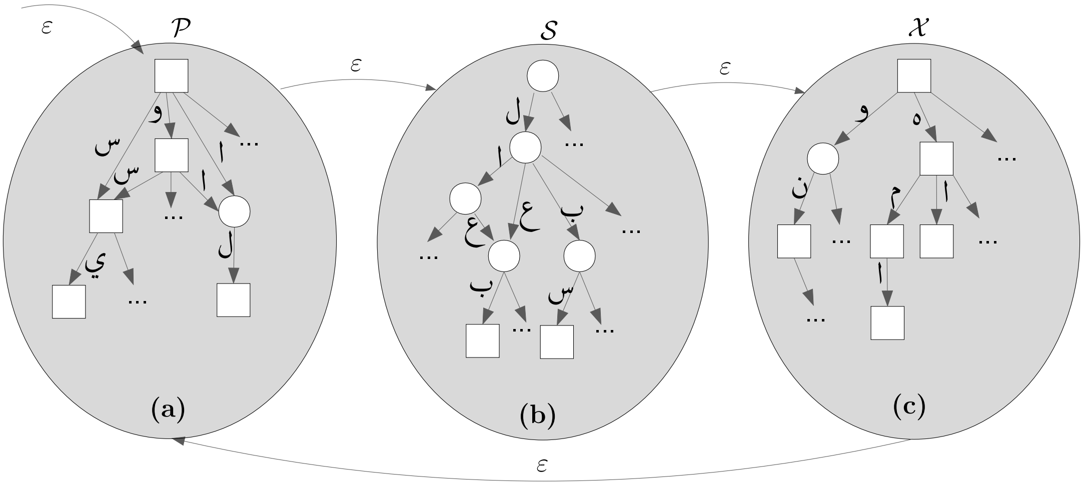


Figure 2.1: FSM of Sarf

We use Sarf to extract the morphological features of words present in a text. Sarf takes an Arabic word as input and returns a set of morphological solutions. Each solution splits the word into its morphemes and associates each morpheme with corresponding tags. We refer to the morphemes and associated tags as features. These features are be used to tag each input word with one or more tags. Sarf processes a word w_i in a text and returns the relevant solution vectors. Multiple solutions are the result of multiple valid segmentations of the word into morphemes, or the multiple possible tags associated with a morpheme. Also, a word might have multiple solution vectors as it can have multiple interpretations and meanings. An example is shown in Table 2.2 below:

Table 2.2: Word with different interpretations

وَلَد*walad*	child/son
وَلَّدَ*walada*	generate
وُلِدَ*wulida*	be born

An output solution vector of Sarf for a word w_i contains the following information:

- w_i: The word that is analyzed.
- w_i-index: The index of the word w_i in the text.
- pref: A prefix of w_i. Note that an input word might have multiple prefixes.
- pref-length: The length of the prefix represented by number of characters.
- pref-gloss: A brief notation of the meaning of the prefix.
- pref-pos: The part of speech of the prefix.
- pref-data: The prefix diacritized.
- stem: The stem of w_i.
- stem-index: The index of the stem in the text.
- stem-length: The length of the stem.
- stem-gloss: A brief notation of the meaning of the stem.
- stem-pos: The part of speech of the stem.
- stem-data: The stem diacritized.

Table 2.3: Sample solution vector

w_i	يأكله *yʾaklh*
$w_i - index$	10
pref	ي *y*
pref-length	1
pref-gloss	he/it
pref-pos	IV3MS+
pref-data	يَ *ya*
stem	أكل *ʾakl*
stem-index	11
stem-length	3
stem-gloss	eat/consume
stem-pos	VERB_IMPERFECT
stem-data	أَكُل *ʾakul*
suf	ه *h*
suf-index	14
suf-length	1
suf-gloss	him/it
suf-pos	IVSUFF_DO:3MS
suf-data	ه *h*

- stem-ac: A list of the abstract categories relevant to this stem. For example, "Country" is an abstract category for "Lebanon".

- suf: A suffix of w_i. Note that an input word might have multiple suffixes.

- suf-index: The index of the suffix in the text.

- suf-length: The length of the suffix.

- suf-gloss: A brief notation of the meaning of the suffix.

- suf-pos: The part of speech of the suffix.

- suf-data: The suffix diacritized.

A sample solution vector is shown in Table 2.3 below:

By the end of the morphological analysis stage, Sarf returns a sequence of solution vectors. These vectors contain all the possible morphological analyses of the words in the input text.

2.4 Classes, labels, and tag types

A *class* is a semantic decision that the NLP of CL task tries to make. Parts of text that belong to the desired class are all assigned the same class *label.* For example, *temporal unit* is a label of a class that encompasses explicit temporal information in the sentence related to time such as دقيقة*dqyqh* (minute) and ساعة*sāˤh* (hour).

The user may wish to define a simple class as an abstract category that contains a set of morphemes. For more sophisticated classes, the user can use the visual interface in MERF to define the class through Boolean formulae with morphology-based atomic terms. The annotation of text with the labels of the classes can later be used in CL and NLP tasks for learning, testing, and validation.

Chapter 3

Motivation

We motivate MERF with an example that extracts directions from a sample Arabic text. Consider the text in Figure 3.1(a) taken from the website of Dubai Mall describing the directions to reach the Mall [1]. Figure 3.1(b) presents an English translation of the Arabic text and Figure 3.1(c) presents a transliteration. The text contains details that are not interesting to the direction extraction task. Interesting entities, such as names, places, relative directions, and numerical terms are highlighted in the text. For example, row 1 in the table shown in Figure 3.1(d) lists the matches of names of persons as n_1, n_2, and n_3. We are also interested in presenting the directions in a relational diagram as shown in Figure 3.1(g).

In the following, we demonstrate how to do that using MERF. In the process the user interacts with the MERF interface to specify Boolean formulae, regular expressions, and semantic relations. MERF will automatically extract formulae matches using an in- house Arabic morphological analyzer (Sarf), extract regular expression matches, and construct the semantic relations.

Boolean formulae definition

Using MERF, the user specifies the N, P, R, and U tag types with Boolean formulae based on morphological features. As described in the table in Figure 3.1(d), the user denotes names of persons with formula N and specifies it with a constraint requiring the category feature in the morphological solution to be `name of person`. Similarly, the user specifies formula P with a `name of place` category. The user specifies formula R to denote relative positions and requires the stem feature to belong to a selected list of stems containing في*fy* and قرب*qrb* . U denotes numerical terms and is specified by a disjunction of con-

[1] `http://www.thedubaimall.com/ar/`.

من المستحيل ألا تلاحظ برج خليفة بالقرب من التقاطع الأول وأنت
تقود سيارتك في شارع الشيخ زايد، حتّى وإن كانت هذه المرّة الأولى
التي تسلك فيها هذا الطريق؛ يقع دبي مول على مقربة من هذا المبنى
الذي يُعد الأطول في العالم.

(a) Text with directions

It is impossible not to notice $\overset{n1}{\boxed{\text{Khalifa}}}$ $\overset{p_1}{\boxed{\text{Tower}}}$ $\overset{r_1}{\boxed{\text{next to}}}$ the $\overset{u_1}{\boxed{\text{first}}}$ $\overset{p_2}{\boxed{\text{intersection}}}$ while you are driving $\overset{r_2}{\boxed{\text{on}}}$ $\overset{n_2}{\boxed{\text{Sheikh}}}$ $\overset{n_3}{\boxed{\text{Zayed}}}$ $\overset{p_3}{\boxed{\text{Road}}}$, even if this was $\overset{u_2}{\boxed{\text{the first}}}$ time that you take this $\overset{p_4}{\boxed{\text{road}}}$; $\overset{p_5}{\boxed{\text{Dubai}}}$ $\overset{p_6}{\boxed{\text{Mall}}}$ is located $\overset{r_4}{\boxed{\text{near}}}$ this $\overset{p_7}{\boxed{\text{building}}}$, which is the longest in the world.

(b) English translation

mn ālmsthyl ʾalā tlāḥẓ brǧ ḫlyfh bālqrb mn āltqāṭʿ ālʾawl wʾant tqwd syārtk fy šārʿ ālšyḫ zāyd, ḥtā wʾin kānt hḏh ālmrh ālʾawlā ālty tslk fyhā hḏā ālṭryq; yqʿ dby mwl ʿlā mqrbh mn hḏā ālmbnā āldy yuʿd ālʾaṭwl fy ālʿālm.

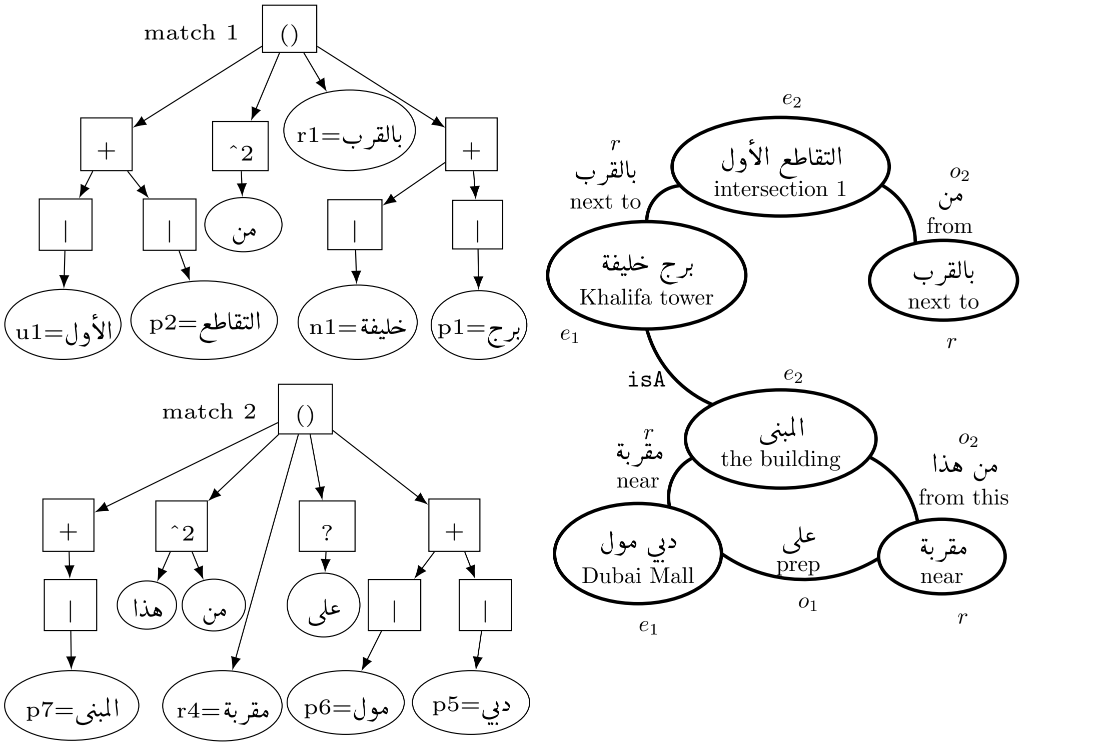
(c) Arabic text transliteration

MBF	description	formula	matches
N	name of person	*category = Name_of_Person*	n_1, n_2, n_3
P	name of place	*category = Name_of_Place*	$p_1, p_2, ..., p_7$
R	relative position	*stem* ∈ {قرب,في,...}	r_1, r_2, r_3, r_4
U	numerical term	*stem* ∈ {أول,ثاني,...}	u_1, u_2

(d) Tag types with Boolean formulae

$$\text{Expression} = (P|N)+\ \ O?\ R\ O\hat{\ }2\ (P|N|U)+$$

(e) Regular expression



(f) MRE match trees

(g) Entity-relation graph

Figure 3.1: Text, formula, and match MERF example

straints requiring the stem feature to belong to a set of stems such as أول*ʾwl* (first), ثاني*ṯāny* (second), ثالث*ṯālṯ* (third), ..., تاسع*tāsʿ* (ninth), and عاشر*ʿāšr* (tenth).

Morphology-based Boolean formulae matches

The existence of a morphological analyzer is crucial to the Boolean formulae match extraction. Stemming is required in the case of بالقرب*bālqrb* in order to detect the relative position tag match based on قرب*qrb* . Thus, MERF calls an in-house morphological analyzer [24] and computes matches of the tag types N, P, R, and U. We refer to all *other* words in the text that do not match a user defined tag type as *null* words and we denote them by O. The resulting matches are illustrated with boxes and superscripts in Figure 3.1(a) and are also listed in the fourth column of the table in Figure 3.1(b). So words with superscripts n_1, n_2, and n_3 are matches of tag type N. Words with superscripts p_1, p_2, $\ldots$, p_7 are matches of tag type P. Words with superscripts r_1, r_2, r_3, and r_4 are matches of tag type R. Words with superscripts u_1, and u_2 are matches of tag type U.

Regular expression definition

The user now interacts with the regular expression editor to specify the direction entities and relations. Intuitively, the directions are names of places (P) related to each other with positional propositions (R). A place name can be a tabulated place name, a street named after a person (N), or a numbered street (U). The text containing the directions might also include words that are not necessary to indicate directions (O) but are necessary to complete the sentence.

The user tries several sequences of the above entities in the editor and checks their matches in the visualizer. Finally, the user is satisfied with an expression such as $(P|N)+\ O?\ R\ O\wedge 2\ (P|N|U)+$ as shown in Figure 3.1(e) where $|, +, ?$, and $\wedge k$ denote disjunction, one or more, zero or one, and up to k matches, respectively. The expression specifies a sequence of places or names of persons, optionally followed by a null word, followed by one relative position, followed by up to two possible null words, followed by one or more match of name of place, name of person, or numerical term. O and $\wedge 2$ are used in the expression to allow for flexible matches.

The user writes the expression by experimenting with the visualizer and the expression editor which does not require knowledge and expertise in regular expressions.

Regular expression matches

MERF computes the matches of the expression in the text. The match trees in Figure 3.1(f) illustrate two of them. The first match tree refers to the text برج خليفة بالقرب من التقاطع الأول *brǧ ḫlyfh bālqrb mn āltqāṭʿ āl-ʾwl* (Khalifa Tower next to the first intersection). The second match tree refers to the text دبي مول على مقربة من هذا المبنى *dby mwl ʿlā mqrbh mn hḏā ālmbnā* (Dubai Mall is located near this building).

The nodes of the trees are entities and the edges and internal nodes are text, morphology-based, and word distance based relational entities. The sequence and structure gives us the text of a parent node from the children nodes. The same sequence can also give us the interesting (matching) morphological features. The word distance is defined and abstracted by the matches of the regular expression operators (internal nodes).

Semantic relations

The user now uses the semantic relation editor to declare semantic relations that relate parts of the matches of the expression with each other. The aim of the semantic relations to be defined is to construct the entity-relation graph shown in Figure 3.1(g). Intuitively, the user wants to create relations between place, name, and numerical entities. A relation between two entities can be a prepositional entity occurring between them. For example, the entities دبي مول *dby mwl* (Dubai Mall) and المبنى *ālmbnā* (the building) are related by the preposition مقربة *mqrbh* (near) as shown in Figure 3.1(g).

Let e_1, o_1, r, o_2, and e_2 be the $(P|N)+$,$O?$,R,$O \wedge 2$, and $(P|N|U)+$ parts of the expression, respectively. The user selected $(P|N)+$ to be an entity after noticing in the visualizer that it happens to capture non-separated sequences of place and name entities denoting a single entity such as `Khalifa tower`.

The declaration of the relation `Relation(`e_1`,`e_2`,`r`)` creates the edge labeled with `next to` between `intersection 1` and `Khalifa tower` nodes in match 1, and the edge labeled with `near` between the `Dubai Mall` and `the building` nodes in match 2. Those relations will be constructed if e_1, e_2, and r exist as match entities.

The relation `Relation(`r`,`e_1`,`o_1`)` creates the edge labeled with `prep` between the `Dubai Mall` and `near` nodes in match 2. This relation will be constructed if r, e_1, and o_1 exist as match entities.

The relation `Relation(`r`,`e_2`,`o_2`)` creates the edge labeled with `from` between the `intersection 1` and `next to` nodes in match 1, and the edge labeled with `from this` between the `near` and `the building` nodes in match 2. This relation will be constructed if r, e_2, and o_2 exist as match entities.

Cross-reference

After constructing the semantic relations, the user is interested to relate the entities across relations to each other. The user aims in this step to relate entities that point to the same concept or thing. In the directions task, the user aims to relate the entities referring to the same place.

MERF provides the `isA` relation as a default cross-reference relation. This relation relates two entities if they are directly synonymous or one entity is a synonym of a synonym of the other entity. As shown in Figure 3.1(g), the cross-reference relation creates the edge between the `Khalifa Tower` and the `The building` nodes.

Chapter 4

Overview

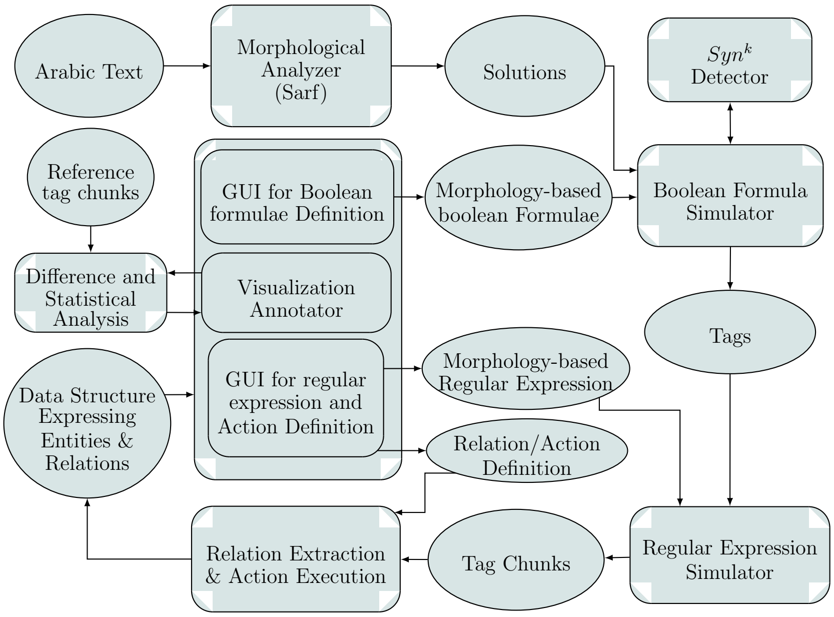

Arabic Text
Morphological Analyzer (Sarf)
Solutions
Syn^k Detector
Reference tag chunks
GUI for Boolean formulae Definition
Morphology-based boolean Formulae
Boolean Formula Simulator
Difference and Statistical Analysis
Visualization Annotator
Tags
Data Structure Expressing Entities & Relations
GUI for regular expression and Action Definition
Morphology-based Regular Expression
Relation/Action Definition
Relation Extraction & Action Execution
Tag Chunks
Regular Expression Simulator

Figure 4.1: MERF flow diagram.

Figure 4.1 shows the flow diagram of MERF. An ellipse node represents input and output data and a box node represents a process inside MERF. The Arabic text and reference tag chunks are input data to MERF. Solutions, morphology-based Boolean formulae, tags, morphology-based regular expression, tag chunks, relation and action definition, and data structure expressing entities and relations are input and output data of processes. Morphological analyzer (Sarf), Syn^k detector, GUI for Boolean formulae definition, visualization annotator, GUI for regular expression and action definition, Boolean formula simulator, regular expression simulator, relation extraction and action execution, and difference and statistical analyzer are MERF processes.

Morphological analyzer (Sarf)

Using the interface of MERF, the user specifies an input Arabic text document. The user aims to extract entities and relations from this text. MERF analyzes the text using an in-house Arabic morphological anaylzer [24] and computes morphological solutions for each word in the Arabic text. A word might have more than one solution due to multiple segmentations or multiple tags associated with each word. Sarf's morphological solution vector contains the different morphemes (stem and affixes), POS and gloss tags, and categories. These morphological features will be used in a later stage for tag type matching. We present a brief description of Sarf in Chapter 2.

Syn^k detector

Syn^k detector is a Boolean function that takes as input two words and returns whether they are related in terms of synonymity in K steps. So given the words w_1 and w_2, w_1 is a synonym of word w_2 or w_1 is a Syn^{k-1} with one of the synonyms of w_2. This unit is used by MERF to evaluate the Boolean formulae that contain this semantic feature. We introduce this feature in detail in Chapter 5.

Boolean formula simulator

The user interacts with MERF through a user-friendly interface to specify tag types with Boolean formulae. Each tag type contains description, visualization legend, and a morphology-based Boolean formula. The visualization legend includes foreground color, background color, font size, and font weight. The Boolean formulae are built by the user using fixed values for morphological features subject to disjunction and negation operations. MERF passes the morphological solutions and the user-defined tag types to the Boolean formula simulator. The simulator interacts with the Syn^k detector, computes the tag type matches, and produces a tag set for each word. A word might have multiple tags as its morphological solutions could match multiple Boolean formulae. Each tag contains information about the matching word and the relevant tag type name. Word information include word index relative to the text, character position, and text. We formally define the MBF and explain its simulation in Chapter 5.

Regular expression simulator

The user interacts with MERF through a user-friendly interface to specify tag types with Morphology-based regular expressions. Each tag type contains description, legend for visualization, and a morphology-based regular expression.

The regular expressions are based on user-defined tag types with Boolean formulae subject to operations such as disjunction, conjunction, zero or more, zero or one, and one or more. MERF then passes the sequence of tag sets and the user-defined tag types with morphology-based regular expressions to the regular expression simulator. The simulator computes tag chunks; i.e. sequences of words, whose tag sets match the expressions. Each tag chunk contains information about the matching sequence of words and the tag type with the matching regular expression. We formally define the MRE and explain its simulation in Chapter 5.

Relation extraction and action execution

MERF enables the user to associate code actions to sub-expressions in the regular expression. In the code actions, MERF provides the user with an API access to match information. The information includes text, position, length, and morphological features.

The user declares the semantic relations using MERF's relation editor. Each relation is defined by two entities and a relational entity representing the edge. The user specifies the entities and relational entities to be the matches of sub-expressions of the regular expressions.

MERF then executes the user-defined actions corresponding to each sub-expression match in a tag chunk. It also evaluates the semantic relation declarations against the tag chunks to compute the semantic relations. Finally, MERF uses a cross-reference relation such as the `isA` relation to create relations across tag chunks. The output of this process is a user-defined data structure that expresses entities and relations among them. We formally define the semantic relation and explain its construction in Chapter 5.

Visualization annotator

MERF presents the resulting tags to the user incrementally in the form of text annotated with style and color legends, match trees, and graphs. The annotation can be edited by the user in a user-friendly interface. Match trees present the match text associated with the relevant tags and regular expression structure. The graphs are the result of the semantic relations defined by the user. We present MERF's interface in Chapter 7.

Difference and statistical analysis

MERF provides statistical analysis tools that help compare sets of tags to reference tags and compute standard accuracy measures. MERF provides criteria for comparison including exact match and intersection. This comparison tool can be used to edit the automatically generated annotations. MERF provides the user with an interactive interface to build the reference tags. We explain the analysis unit and its interface in Chapter 7.

Chapter 5

MERF

MERF takes a sequence of Arabic words $T = \langle t_1, t_2, \ldots, t_M \rangle$ as input text, a set of tag types $\mathcal{T}$ each with its Boolean or sequential formulae. MERF is integrated with *Sarf*, an open source Arabic morphological analyzer based on finite state transducers [24]. MERF uses Sarf to compute a set of morphological solutions $M(t_i) = \{m_1, m_2, \ldots, m_N\}$ for each word $t_i, 1 \leq i \leq M$.

Recall from Chapter 2 that $\mathcal{P}$, $\mathcal{S}$, $\mathcal{X}$, POS, $GLOSS$, and CAT denote the set of prefixes, stems, suffixes, POS tags, gloss tags, and abstract category tags respectively. Each morphological solution m is of the form $\langle p, s, x, P, G, C \rangle$ where $p \in \mathcal{P}, s \in \mathcal{S}, x \in \mathcal{X}, P \in POS, G \in GLOSS$, and $C \in CAT$.

In what follows we formally present MERF and its components.

5.1 Syn^k

Let E and A be the sets of all English and Arabic words respectively. Let G $\subset$ E be the set of English glosses. Let L $\subset$ A be the set of Arabic words in a given lexicon. Let S $\subset$ L be the set of stems in the given Arabic lexicon. We define the following functions.
Let function α: S $\rightarrow 2^G$, map input Arabic Lexicon stems to subsets of related English glosses; e.g. g$_s = \alpha(s) \subset 2^G$.
Let function γ: L $\rightarrow 2^S$, map input Arabic Lexicon words to subsets of relevant Arabic stems; e.g. s$_l = \gamma(l) \subset 2^S$.

Given an Arabic word w$\in$L, we define Sy(w) to be the set of Arabic words directly related to w under gloss map.
$Sy(w) = \{u \mid u \in S \wedge \exists s \in \gamma(w) \wedge\ \alpha(u) \cap \alpha(s) \neq \phi\}$

Example:1
Consider the following example:
L = $\{l_1,l_2,l_3,l_4,l_5,l_6\}$
S = $\{l_1,l_2,l_3,l_4\}$

E = {e$_1$,e$_2$,e$_3$,e$_4$,e$_5$,e$_6$,e$_7$}
G = {e$_1$,e$_2$,e$_3$,e$_4$}
α = **{(l$_1$,{e$_1$,e$_2$}),(l$_2$,{e$_2$})**,(l$_3$,{e$_2$,e$_3$}),(l$_4$,{e$_3$,e$_4$})}
γ = {(l$_1$,{l$_1$}),(l$_2$,{l$_2$}),(l$_3$,{l$_3$}),(l$_4$,{l$_4$}),**(l$_5$,{l$_1$,l$_2$})**,(l$_6$,{l$_4$})}
Given an Arabic word w = l$_5$, the stems of w are given by:
γ(w) = {l$_1$,l$_2$}.

Table 5.1: Detailed example of computing Sy(l$_5$)

s	α(s)	$\alpha(l_1) \cap \alpha(s)$	$\alpha(l_2) \cap \alpha(s)$
l$_1$	e$_1$,e$_2$	e$_1$,e$_2$	e$_2$
l$_2$	e$_2$	e$_2$	e$_2$
l$_3$	e$_2$,e$_3$	e$_2$	e$_2$
l$_4$	e$_3$,e$_4$	$\emptyset$	$\emptyset$

After computing the stems of the input word w, we compute α of its stems as shown in Table 5.1 above where:
α(l$_1$) = {e$_1$,e$_2$} ;gloss map of l$_1$, and
α(l$_2$) = {e$_2$} ;gloss map of l$_2$.

The next step is to compute α for each s $\in$ S. Then, we get the intersection between $\alpha(s)$ and $\alpha(l_{5i})$, where l$_{5i}$ $\in \gamma(l_5)$, as shown in the last two columns of Table 5.1. if the intersection set is not empty, add s to Sy(w).

For s = l_3, $\alpha(l_3)$ = {e$_2$,e$_3$}. Since $\alpha(l_3) \cap \alpha(l_1) = \{e_2\}$, l_3 is included in Sy(w).

For s = l_4, $\alpha(l_4)$ = {e$_3$,e$_4$}. Since the intersection set $\alpha(l_4) \cap \alpha(l_{5i}) = \emptyset$, l_4 is not included in Sy(w). Finally, Sy(w) = {l$_1$,l$_2$,l$_3$}.

We define Sy^i(w) to denote stems related to w using gloss map of order i recursively.
$Sy^1(w) = Sy(w)$.
$Sy^{i+1}(w) = \{u \mid u \in S \wedge \exists s \in Sy^i(w) \wedge \ \alpha(u) \cap \alpha(s) \neq \phi\}$.

Table 5.2: Example of computing Sy^2(l$_5$)

s	α(s)	$\alpha(l_3) \cap \alpha(s)$
l$_1$	e$_1$,e$_2$	e$_2$
l$_2$	e$_2$	e$_2$
l$_3$	e$_2$,e$_3$	e$_2$,e$_3$
l$_4$	e$_3$,e$_4$	e$_3$

Example:2

Table 5.2 shows that Sy^2(l$_5$) adds l$_4$ to the set of words related to l_5, since $(\alpha(l_3) \cap \alpha(l_4)) \neq \emptyset$. Hence, Sy^2(l$_5$) = {l$_1$,l$_2$,l$_3$,l$_4$}.

We define $Syn^k(w)$ to be the union of $\mathrm{Sy}^i(w)$ for i=1 … k. Formally, $Syn^k(w) = \bigcup_{i=1}^{k} Sy^i(w)$.

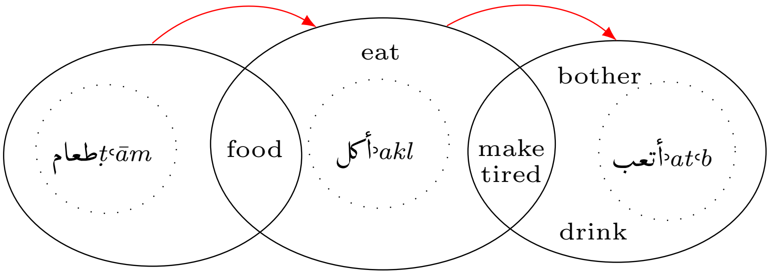


Figure 5.1: *Syn*^{2}(طعام *ṭʿām*)

Consider the example shown in Figure 5.1. The example presents the Arabic words طعام *ṭʿām* , أكل *ʾkl* , and أتعب *ʾtʿb* with the glosses {food}, {food, eat, make tired}, and {make tired, bother, drink} respectively. In Sy(طعام *ṭʿām*), we relate طعام *ṭʿām* to أكل *ʾkl* through the gloss intersection `food`. In Sy(أكل *ʾkl*), we relate أكل *ʾkl* to أتعب *ʾtʿb* through the gloss intersection `make tired`. Hence, we relate طعام *ṭʿām* to أتعب *ʾtʿb* in *Sy*^{2}(طعام *ṭʿām*).

Syn^2(طعام *ṭʿām*) $= \bigcup_{i=1}^{2}$ *Sy*^{i}(طعام *ṭʿām*) $=$ ⟨أكل *ʾkl* , أتعب *ʾtʿb* ⟩.

5.2 Morphology-based atomic terms (MAT)

Let $\mathcal{O} = \{isA, contains\}$ be the set of atomic term predicates, and let $\mathcal{F} = \{\mathcal{P}, \mathcal{S}, \mathcal{X}, POS, GLOSS, CAT\}$ be the set of morphological features.

A MAT $a(w)$ in a Boolean tag type formula takes a word w and a constant value of a morphological features CF as input and is of the form.

$$a(w) := \exists m \in M(w).m = \langle p, s, x, P, G, C\rangle.r \circ CF$$

where $\circ \in \mathcal{O}$, $r \in \{p, s, x, P, G, C\}$, $\exists A \in \mathcal{F}.r \in A, CF \in A$.

Informally, a MAT indicates that a solution vector exists where a feature from the solution contains or exactly matches a constant value for the feature specified by the user.

Another form of an MAT is based on the Syn^k feature. It takes a word w, a constant stem value CS and a constant integer k and checks whether a stem of w belongs to Syn^k(CS). Formally,

$$a(w) := \exists s \in \gamma(w).s \in Syn^k(CS)$$

where $k \in \{1, \dots, 7\}$

- t_1: $\exists m \in M(w)$ such that $m.P$ $contains$ $VERB$
- t_2: $\exists m \in M(w)$ such that $m.C$ isA $PROPER\_NAME$
- t_3: $\exists m \in M(w)$ such that $m.P$ $contains$ $POSS\_PRON$
- Underline marks words with multiple tags

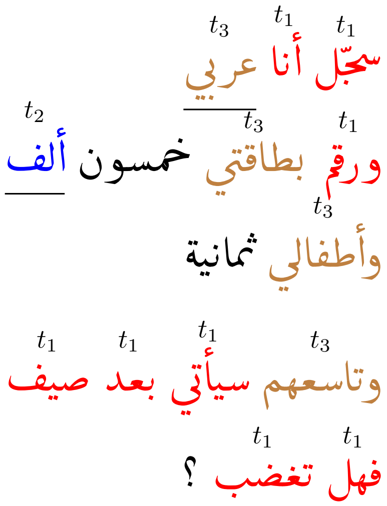


Figure 5.2: Morphology-based atomic term examples

Consider the example shown in Figure 5.2. We define three morphology-based atomic terms $t_1, t_2,$ and t_3. t_1 checks if there is a morphological solution that belongs to the set of morphological solutions of the word w such that the POS tag contains VERB. Similarly, t_2 checks if the category tag is a proper name, and t_3 checks if the POS tag contains possessive pronoun.

The matches of these MATs are shown in the text. Matches of t_1 are colored in red, matches of t_2 are colored in blue, and matches of t_3 are colored in brown. Words that are underlined are tagged with more than one MAT.

5.3 Morphology-based Boolean formula (MBF)

The MERF Boolean formula is of the following form.

- a is an MBF where a is an MAT.
- $\neg f$ is an MBF where f is an MAT. This is interpreted as the negation (complement) of words matching f.

- $f \vee g$ where f and g are MBF. This is interpreted as the disjunction (union) of words matching f with the words matching g.

- t_1 : $\exists m \in M(w)$ such that $m.P$ *contains VERB*
- t_2 : $\exists m \in M(w)$ such that $m.C$ *isA PROPER_NAME*
- t_3 : $\exists m \in M(w)$ such that $m.P$ *contains POSS_PRON*
- Examples:
  - F_1: $\neg t_1$
  - F_2: t_2 or t_3

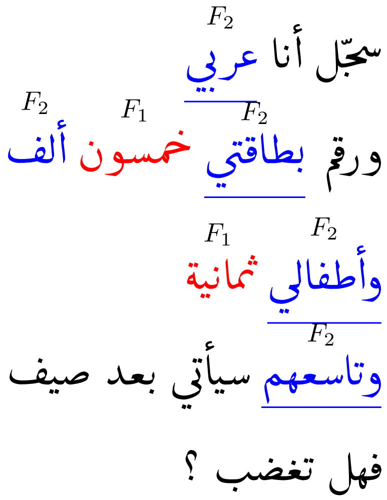


Figure 5.3: Morphology-based Boolean formula examples

Consider the example shown in Figure 5.3. We use the same text and MATs defined previously in Figure 5.2. We define the two Boolean formulae $F_1, and F_2$. F_1 is the negation of t_1 and targets the words that don't have a morphological solution with a POS tag that contains verb. F_2 is a disjunction between t_2 and t_3 and targets the words that have a category that is a proper name or POS tag that contains possessive pronoun.

The matches of these MBFs are shown in the text. Matches of F_1 are colored in red and matches of F_2 are colored in blue. The underlined words are matches of both formulae.

5.4 Morphology-based regular expression (MRE)

The MERF sequential formula is of the following form.

- m is an MRE where m is an MBF.
- fg is an MRE where f and g is an MRE. This is interpreted as the sequence MRE f followed by MRE g.
- $f*$ is an MRE where f is an MRE. This operation refers to the Kleene star interpreted as zero or more occurrences of MRE f.
- $f+$ is an MRE where f is an MRE. This operation is interpreted as one or more occurrences of MRE f.

- $f\hat{}x$ is an MRE where f is an MRE and $x \in \mathbb{N}^*$. This is interpreted as zero or more occurrences of MRE f up to x.
- $f?$ is an MRE where f is an MRE. This is interpreted as zero or one occurrence of MRE f.
- $f\&g$ is an MRE where f is an MRE and g is an MRE. This is interpreted as the occurrence of MRE f and MRE g.
- $f|g$ is an MRE where f is an MRE and g is an MRE. This is interpreted as the occurrence of MRE f or MRE g.

- MATs:
  - $t_1 : \exists m \in M(w)$ such that $m.P$ *contains VERB*
  - $t_2 : \exists m \in M(w)$ such that $m.C$ *isA PROPER_NAME*
  - $t_3 : \exists m \in M(w)$ such that $m.P$ *contains POSS_PRON*
- MBFs:
  - F_1: $\neg t_1$
  - F_2: t_2 or t_3
- MRE:
  - E_1: $F_1\&F_2$

سجّل أنا E_1 عربي

ورقم E_1 بطاقتي خمسون ألف

E_1 وأطفالي ثمانية

E_1 وتاسعهم سيأتي بعد صيف

فهل تغضب ؟

Figure 5.4: Morphology-based regular expression example

Consider the example shown in Figure 5.4. We use the same text, MATs, and MBFs defined previously in Figure 5.3. We define the regular expression E_1. E_1 is a conjunction between F_1 and F_2 and targets text chunks (one word in this expression) that match F_1 and F_2. In other words, it targets words that don't have a POS tag that contains a verb and either the category is a proper name or part of speech tag is a possessive pronoun. The matches of E_1 are shown in the text and colored in red.

5.5 Computational actions

MERF allows the user to specify computational actions to be executed when a match for a sub-expression is found. These actions can be used by the user to

```
cout << $s2.text;
isHundred = true;
if(current == 0)  {
           currentH=$s2.number;
}
else {
           if(!isKey) {
              currentH= current * $s2.number;
              current = 0;
           }
           else {
              currentH = $s2.number;
           }
}
isKey = false;
```

Figure 5.5: Sample code for an on-match action

store the results in separate files to use later, perform statistical analysis, and apply algorithms such as the number normalization task explained in Chapter 8.

The computational actions are C++ snippets of code that may contain MERF API calls. The MERF API calls allow the code to access solution features referring to sub-expression matches. The features include text, position in text, length of word (count of characters), the equivalent numerical value if applicable, and morphological features.

Each MRE is associated with two computational actions. Once all matches are computed, MERF computes the sequence of actions as follows. The prematch action of a match m of a sub-expression e gets executed. The subexpression e may include other smaller sub-expressions with their own actions. MERF executes those actions. Then, MERF executes the on-match action of m.

In addition the user can also specify code headers to associate the expressions with proprietary code such as declaring global variables and including user libraries.

Consider the sample code action shown in Figure 5.5. This code action is taken from the number normalization case study which we explain in detail in Chapter 8. The variables `isHundred`, `isKey`, `current`, and `currentH` are global variables declared by the user. The use of `cout` requires the user to include the `iostream` library. As shown by the example, the user can access the word features by using the key character $ followed by the sub-expression name and target word feature. In this example, the user prints the sub-expression text and processes its numerical value.

As for the morphological features, the user accesses them using the command `$sub_expression_name.matches`. The API returns a vector of solutions where each solution contains the stem, arrays of affixes (prefixes and suffixes), stem gloss, arrays for affix glosses, stem POS, and arrays for affix POSs.

5.6 Tag type

The set of tag types $\mathcal{T}$ contains tuples of the form $\langle l, f, d \rangle$ where l is a text label with a descriptive name of the tag type, f is a MERF MBF or MRE, and d is a visualization legend. The visualization legend describes how the tags should be displayed to the user and contains information such as the foreground and background color, the font size, weight, and style.

5.7 MBF evaluation

For each word $t_i \in T$, MERF computes a Boolean value ($\{\mathit{true}, \mathit{false}\}$) for all atomic terms. Then MERF computes Boolean values for all Boolean formulae. Then MERF computes the set of tags $R_i \subseteq T \times \mathcal{T}$ such that $(t_i, tt_j) \in R_i$ if and only if the Boolean formula F_j associated with tag type tt_j is true for t_i.

The MBF evaluation results in a sequence of tag sets $\langle R_0, R_1, \ldots, R_{n-1} \rangle$ where R_i is the tag set for word $t_i \in T, 0 \leq i < n$. If a word t_o doesn't have any tag type match, it is tagged by a default tag type called O, referred to as *NONE*. We refer to these words as *null* words.

5.8 MRE and action simulation

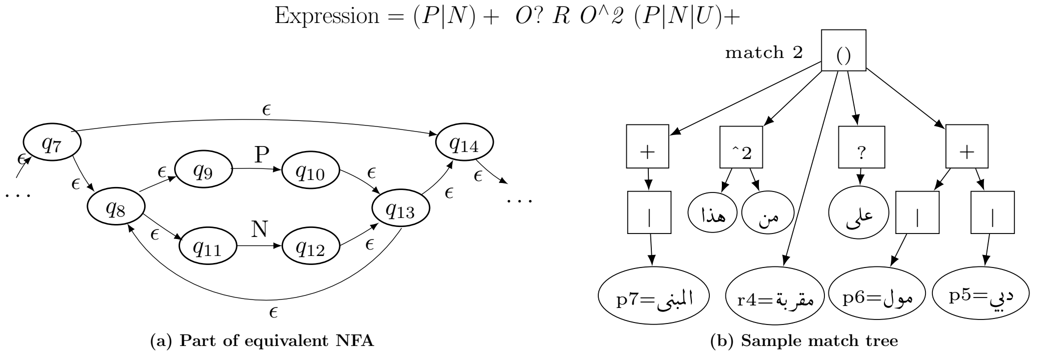
For each MRE, MERF generates its equivalent non-deterministic finite automaton (NFA) in the typical manner [25]. Each MRE operation has its equivalent representation in an NFA. As for the upto operation ($f\hat{}x$), which is not directly supported in [25], we can expand it into a standard regular expression form, for example f^5 is equivalent to $f?|ff|fff|ffff|fffff$.

MERF simulates the generated NFA over the sequence of tag sets generated in the MBF evaluation stage. A simulation match m is a vector of the form $\langle r_k, r_{k+1}, \ldots, r_j \rangle$ where $r_k \in R_k, r_j \in R_j, 0 \leq k < j < n$. This tag match corresponds to the text sequence $\langle t_k, t_{k+1}, \ldots, t_j \rangle$ where $t_k, t_j \in T$.

If the simulation has a match $\langle r_m, r_{m+1}, \ldots, r_n \rangle$ where $0 \leq m \leq n$, the next simulation starts at R_{n+1}. This disallows overlap of matches for the same MRE.

In case the NFA simulation has no match, the next simulation starts at R_{m+1}. If we have more than one match starting at R_k where $0 \leq k \leq n$, MERF currently returns the longest one.

MERF maintains a function $\phi \subset Q \times \Phi$, where Q is the set of states in the NFA and Φ is the set of sub-expressions in the MRE. So $(q, f) \in \phi$ iff state $q \in Q$ was generated by MERF to correspond to sub-expression $f \in \Phi$. MERF uses ϕ to compute a match tree with respect to the MRE regular expression. It also uses ϕ and the match sequence to compute the sequence of computational actions of an MRE match.



(a) Part of equivalent NFA

(b) Sample match tree

Figure 5.6: MERF expression, NFA, and match example

Figure 5.6 illustrates the MRE simulation process. We consider the direction extraction task discussed in Chapter 3. The user defines the MRE shown at the top of Figure 5.6. The MRE simulation unit generates the equivalent NFA of the regular expression. Part of the NFA is shown in Figure 5.6(a). States $q_7, q_8, \ldots, q_{14}$ represent NFA states and the edges are transitions based on input labels. P and N are labels referring to MBF tag type names and ϵ is an empty string. ϵ is the source of non-determinism in the simulation.

MERF simulates the NFA with the MBF-based tag type matches and calculates MRE matches. Figure 5.6(b) shows a sample match of the MRE. Leaf nodes are matching MBF tags represented in the Figure with their original text, and internal nodes represent matches to MRE operations. دبي*dby* and مول*mwl* are sample leaf nodes referring to name of place tags (P). $+, |$, and ? are internal nodes referring to MRE one or more, disjunction, and zero or one operations, respectively.

5.9 Semantic relations

A Semantic relation is a directed labeled binary relation of the form $\langle e_1, e_2, r \rangle$

- e_1: identifier associated with an MRE sub-expression denoting the source
- e_2: identifier associated with an MRE sub-expression denoting the destination
- r: identifier associated with an MRE sub-expression or a user-defined constant, denoting the label of the relation between e_1 and e_2.

We refer to a tuple of the form $\langle e_1, e_2, r$ as a relational entity.

Moreover, MERF defines the cross-reference relation between two entities e_1 and e_2 as:

$$cr(e_1, e_2) := e_1.text \in Syn^2(e_2.text)$$

- Annotated Expression
  - $(P|\overset{e_1}{N})+\ \overset{o_1}{O}?\ \overset{r}{R}\ \overset{o_2}{O}\text{\textasciicircum}2\ (P|\overset{e_2}{N}|U)+$
- User defined semantic relations
  - $\langle e_1, e_2, r \rangle$
  - $\langle r, e_1, o_1 \rangle$
  - $\langle r, e_2, o_2 \rangle$

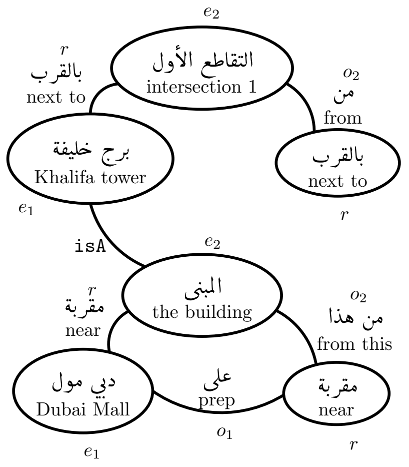


Figure 5.7: Semantic relation example

Consider the example shown in Figure 5.7. We illustrate the semantic relation and cross-reference construction using the match trees in Figure 3.1(f). MERF assigns the notations e_1, e_2, o_1, o_2, and r to $(P|N)+$, $(P|N|U)+$, $O?$, $O \wedge 2$, and R, respectively, as shown in the top of Figure 5.7. We use these notations to define the semantic relations. Hence, we define the relations $\langle e_1, e_2, r \rangle$, $\langle r, e_1, o_1 \rangle$, and $\langle r, e_2, o_2 \rangle$.

The matches of e_1, e_2, o_1, o_2, and r from the match tree in Figure 5.6 are دبي مول*dby mwl* (Dubai Mall), المبنى*ālmbnā* (the building), على*ʿlā* (prep), من هذا*mn hḏā* (from this), and مقربة*mqrbh* (near), respectively. MERF constructs the semantic relation matches and builds the entity-relation graph shown in the lower part of Figure 5.7.

Similarly, we construct the relations of the match برج خليفة بالقرب من التقاطع الأول*brǧ ẖlyfh bālqrb mn āltqāṭʿ āl-ʾ-wl* . The matches of e_1, e_2, o_2, and r are برج خليفة*brǧ ẖlyfh* (Khalifa tower), التقاطع الأول*āltqāṭʿ āl-ʾ-wl* (intersection 1), من*mn* (from), and بالقرب*bālqrb* (next to), respectively. MERF doesn't construct the relation $\langle r, e_1, o_1\rangle$ since o_1 has no match. Therefore, we get the entity-relation graph shown in the upper part of Figure 5.7.

After constructing the semantic relations, MERF constructs the cross-reference relations between the extracted entities. MERF uses the Syn^k feature of second order to find cross relations (Syn^2). The graph in Figure 5.7 shows the cross-reference relation constructed between برج خليفة*brǧ ẖlyfh* (Khalifa tower) of the first match with المبنى*ālmbnā* (the building) of the second match. The edge is labeled with `isA`.

Chapter 6

Implementation

In this Chapter, we describe the different implementation aspects of MERF. We cover the data model used to store the MBF and MRE based tag types, the MBF and MRE tags, and the generated actions. We explain the data structures used to hold the data from the files including MBF and MRE tag types, MBF and MRE matches, code action, and relations. Then, we describe the simulation of the MBFs and MREs to compute the matches, the construction of relations matches, and the execution of actions. We also describe how the interface is written with Qt and how the match tree and entity-relation graphs are visualized.

6.1 Data Model

MERF saves the tag types and the tags in a user friendly format using the JavaScript Object Notation (JSON) data-interchange format [26]. The JSON interface is built on two structures. The first structure is a collection of name and value pairs. An instance of the structure is referred to as an object. The second structure defines ordered lists of values as arrays. The name in a name value pair is an identifier string and the value can be a string, a number, a Boolean (true or false), an object, or an array.

6.1.1 Tag file

The tag file keeps the paths of the separate text and tag type files. It also contains a list of tags with their tag type identifiers as well as their position in text in terms of number of characters from the beginning of text, length of the tag, and word index. In addition to the previous features, the MRE matches contain an object representing the match tree with the identifier “match”.

A sample tag file is shown in Figure 6.1. The tag file includes the tag type file pointer(TagTypeFile), text file pointer (file), the list of MBF tags (TagArray), the list of MRE tags (simulationTags), and *textchecksum* which is used to ensure

```
{ "TagArray":
  [{"length": 3,"pos": 152,"source": 1,"type": "P","wordIndex": 29},...],
  "simulationTags":
  [{"formula": "direction","length": 33,"match": {...},"pos": 22,"source": 1},...],
  "TagTypeFile":"directions.stt.json",
  "file":"directions.txt",
  "textchecksum":211
}
```

Figure 6.1: Tag file format

that the text is not corrupt. An MBF tag is defined by its position in the text (pos), length of the tag (length), tag type (type), word index (wordIndex), and the source of the tag (source). A simulation tag, referring to an MRE match, is defined by the formula (MRE), length, position and the match tree (match).

6.1.2 Tag type file

The tag type file contains two objects referring to MBF and MRE based tag types. Each object has an array value containing the user-defined tag types. Each tag type is stored as an object and contains the name, description, formula or expression, and the visualization legends.

```
{ "TagTypeSet":
  [{"Description":"numerical term",
   "Features":[{"Negation": "",
              "Relation": "isA",
              "Stem-Gloss": "first"},
              ...
            ],
   "Tag": "U",
   "background_color": "grey",
   "bold": false,
   "font": 12,
   "foreground_color": "orangered",
   "italic": false,
   "underline": false
   },...
  ],
```

```
 "MSFs": [{
     "name": "direction","description": "",
    "MSF": {... {"MBF": "U","actions": "",
                "init": "","name": "s7",
                "parent": "s13","type": "mbf"}
                ...
          },
    "Relations": [...
            { "entity1": "s9","e1Label": "text",
              "entity2": "s14","e2Label": "text",
              "edge": "s3","edgeLabel": "text",
              "name": "r1"}
              ... ],
     "bgcolor": "#9acd32","fgcolor": "#f0f8ff",
     "includes": "","members": "",
     "delimiter": true
    }]
}
```

Figure 6.2: Tag type file format

Figure 6.2 shows a sample tag type file. The file includes a list of defined MBF tag types (TagTypeSet). Each tag type is defined by the fields name

(Tag), description, morphological features (Features), foreground color, background color, bold, italic, underline, font size (font), and id. The morphological features are stored in an array of objects where each object refers to an MAT. An MAT object contains the fields negation, relation (isA or contains), and feature name and value pair.

The file also includes a list of defined MRE tag types and refers to them with the notation Morphology-based Sequential Formulae (MSFs); the tool name of MRE. Each tag type is defined by the fields name, description, expression, semantic relations (Relations), code action information (includes and members), delimiter, and foreground and background color. The includes and members contain the libraries and global variables added to the actions by the user respectively.

The expression is a tree of objects and arrays whose depends on the definition of the user. For example, a sequence of sub-expressions is expressed as an array of objects. A sample MBF added to the expression is shown in the Figure defined by the name (MBF), pre-match actions (init), on-match actions (actions), unique name assigned by MERF (name), name of parent expression (parent), and expression type (type) which is specified based on operation applied. The delimiter entry specifies whether the simulation should stop on a full stop or any punctuation mark.

Relations are saved in an array where each relation is defined by its name, the first entity (entity1) and its label (e1Label), second entity (entity2) and its label (e2Label), and the edge and its label (edgeLabel).

The user-defined actions (pre-match and on-match) written into a cpp output file. The file includes the libraries included, global members declared, pre-match and on-match functions, and the function calls. A sample of the generated file is shown in Figure 6.3. The Figure shows a sample on-match function for a sub-expression named `s2` (s2_Match). The function takes an integer as a parameter. The integer refers to the numerical value of the match of `s2` (s2_number). The second function named `number_actions` contains the list of function calls. A sample call of the s2_Match is shown with the parameter value 200.

6.2 Data structures

In this section, we explain the data structures we used to store the text, MBF-based tag types, MRE-based tag types, relations, MBF matches, MRE match trees, and relations constructed.

```
extern "C" void s2_onMatch(int s2_number) {
   isHundred = true;
   if(current == 0)  {
      currentH=s2_number;
   }
   else {
      if(!isKey) {
         currentH= current * s2_number;
         current = 0;
      }
      else {
         currentH = s2_number;
      }
   }
   isKey = false;
}
```

```
extern "C" void number_actions() {
   s5_preMatch();
   s4_preMatch();
   s2_preMatch();
   s2_onMatch(200);
   s4_onMatch();
   s4_preMatch();
   s3_preMatch();
   s0_preMatch();
   s0_onMatch(9);
   .
   .
   .
}
```

Figure 6.3: sample generated actions file

6.2.1 Arabic document

MERF supports Arabic documents with UTF-8 encoding only. The input Arabic text is saved as a string. We process the text to build a word position to word index hash, a set of words that end by punctuation, and a set of words that end with full stop. The two sets are used in MRE simulation.

6.2.2 MBF tag types

We store the MBF tag types in a vector of `TagType` class. Each tag type contains strings to store the name, description, foreground color, and background color. Boolean variables are used to track the bold and italic properties. The Boolean formula is stored as a vector of quadruples where each quadruple contains the feature, feature value, negation option, and predicate (isA or contains).

6.2.3 MRE tag types

The MRE tag types are stored in a vector of `MSFormula` class. The regular expression is stored in a tree structure where each node refers to a sub-expression. Each sub-expression is represented by a class derived from a base class called `MSF`. `MSF` defines the common attributes of all sub-expressions including name, pre-match code actions, and on-match code actions. It also contains the common functions required to be implemented by all derived classes.

An MRE tag type is stored in a class called MSFormula. This class contains strings for description of the tag type, foreground color, background color, included libraries by user, and action global variables. It also contains a vector of MSF pointers referring to a sequence of sub-expressions. MSFormula contains a formula that maps the name of each sub-expression to a pointer of the relevant structure instance.

The expressions with the operations zero or one (?), zero or more ($*$), one or more (+), and zero up to x ($f\hat{}x$) are stored in a class called `UNARYF`. This class holds the operation and an MSF pointer referring to the sub-expression subject to the operation. The expressions with the disjnuction (|) or conjunction (&) operations are stored in a class called `BINARYF`. This class holds the operation along with two MSF pointers referring to the two sub-expressions subject to conjunction or disjunction. The sequential expression is stored in a class called `SequentialF`. It contains a vector of MSF pointers referring to a sequence of sub-expressions. An MBF-based sub-expression is stored in a class called `MBF` which contains the name of the MBF tag type. All the classes introduced above are derived from the base class `MSF`.

Moreover, the MRE tag type stores the relevant semantic relations defined by the user. The relations are stored in a vector and each relation is represented by a class `Relation`. This class holds MSF pointers that refer to the three entity sub-expressions, and strings to identify the name of the relation and the labels of the three entities.

6.2.4 MBF tags

The MBF tags are stored in a multi-hash based on wordindex. We use a multi-hash because a single word can have multiple tags as stated before. Each tag is represented in an instance of the `Tag` class. This class holds a pointer to the tag type, and integers for word index, position, and length. It also contains a flag to differentiate the automatic tag from a user one.

6.2.5 MRE tags

We store the MRE tag type matches in a vector of `Match` class. In order to preserve the relation between the expression and the match tags, we use similar classes to store the match in a tree structure as well. Hence, we define the classes `KeyM`, `UnaryM`, `BinaryM`, and `SequentialM` referring to MBF, UNARYF, BINARYF, and SequentialF matches, respectively. All those classes inherit from base class `Match`.

The base class contains common data including pointer to relevant MSF, operation if present, and a flag to differentiate the automatic tag from a user one. KeyM holds the matching word, tag name (key), position, and length. UnaryM contains a vector of `Match` pointers and an integer to the limit in UPTO operation if applicable. BinaryM contains two `Match` pointers. Both pointers are required in the conjunction case, but only one is used in the disjunction. The SequentialM class holds a vector of `Match` pointers.

6.3 MBF simulation

The Arabic morphological analyzer, Sarf, provides a pure virtual function called *on_match*. This function is triggered by the analyzer for every morphological solution computed for an input word. Through this function, we can access the morphological features of the solution including the stem, affixs, POS and gloss tags, and categories.

We derive the class `SarfTag` from `Stemmer`, the main class of Sarf, and implement the *on_match* function. For each solution found, we iterate over the MBFs and check whether the solution matches any of the MATs defined by the user. In case there is a match, we add a tag to the multi-hash referring to the word index. The tag contains a pointer to the tag type that refers to the matching MBF.

In order to minimize the computational cost of the Syn^k feature, we compute the sets prior to the tag computation process. For each CS (constant stem) and k (order) pair, we compute the set of Arabic stems that are synonyms of the stem CS of order k. When we find a Syn^k based MAT in the tag computation process, we detect a match if a solution stem belongs to the relevant set.

6.4 MRE simulation

In this Section, we describe how we generate the NFA equivalent to the MRE. Then we describe how we simulate the NFA and construct the match trees.

6.4.1 NFA generation

In order to simulate the regular expression, we first generate the equivalent non-deterministic finite automata (NFA). We represent the generated NFA with the `NFA` class which contains strings representing start and accept states, a multi-map for transitions, and a map from NFA state to relevant MSF. The state name is represented by a string of the form q_0, q_1, ... The transition multi-hash takes a concatenation of current state and label (epsilon or tag type name). It the transition exists, the hash returns one or more transition states. A sample transition input is $q_1|NONE$ and a sample value would be q_2 if the transition exists. The MSF map is used to track the source sub-expression of each state and whether the state was generated with a pre-match or on-match property. This information is important in the simulation of the NFA, and computation of the match tree. Thus, the map takes the state name as input and returns a pair of MSF pointer and string indicating the pre-match or on-match property.

Figure 6.4 shows a sample NFA generated for a zero or more expression ($*$). The dotted block represents the sub-expression subject to the zero or more operation. q_i is the start state of this expression and q_j is the end state. The

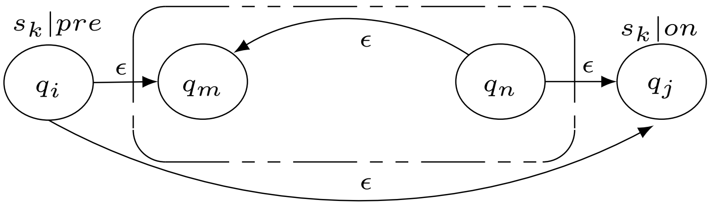


Figure 6.4: Star expression NFA

map entry of the start state is s_k concatenated with *pre*. s_k is the name of the stared expression and *pre* refers to pre-match. Similarly, *on* in the q_j entry refers to on-match.

6.4.2 NFA simulation

We simulate the generated NFA in a recursive function. The function takes as parameters a pointer to the NFA, current state, and current word index and returns a `Match` instance. At each stage, we check for possible transitions from current state based on empty strings (ϵ) or MBF tags. Note that we can get the tags for current word index using the multi-hash used to store the MBF tag based on word index.

When we reach the accept state, we build the match tree backwards. At each stage, we check if the current state refers to a sub-expression pre-match or on-match property. On an on-match property, we generate the relevant match node compatible with the sub-expression type (Subsection 6.2.5) and add it as a child node to returned `Match` structure. In case of a pre-match property, we return the parent node of the `Match` structure.

6.5 Code action execution

In this Section, we describe how we generate the code action file and then compile, link, and run it.

6.5.1 Action file generation

In order to execute the user-defined code actions, we first generate a C++ file. This file contains the includes, global declared variables, pre-match and on-match functions, and the sequence of function calls.

The libraries and global variables are directly inserted at the beginning of the file as entered by the user. Then, we generate the pre-match and on-match functions referring to each sub-expression in the MRE tree. In case the user uses the feature access API, we process the feature API and pass the relevant variable as a parameter to the function. For example, `$s0.number` allows the user to access the numerical value of the match of the sub-expression s_0. This call is processed and transformed into the form `s0_number`, and it is added as an integer parameter of the calling function.

After the MRE simulation stage, we traverse the match trees and generate the sequence of pre-match and on-match calls. The function calls are listed in a function named after the MRE tag type name such as `number_actions() {...}`. At each node in a match tree, we generate the pre-match call of the current node, call the generation method of children `Match`, then generate the on-match call of the current node.

6.5.2 Action file execution

In order to execute the generated C++ file, we use shared object libraries and dynamic loading. The shared object library allows us to compile, load, and run the action file at run-time using dynamic linking loader system functions. The process requires us to create the shared library, load it, then use it by calling the target functions.

We create the library using a system call of the form:
`/usr/bin/g++ -fPIC -shared cppFile -o cppFile_Path lib_MREName.so`.
The `cppFile` is the name of the action file generated, `cppFile_Path` is the path of the action file, and `lib_MRENAME.so` is the name of the output shared library.

We load the library using the `dlopen` function provided by the dynamic library. The function takes the library name as input and returns a library pointer. As previously noted, the function calls are listed in a single function. In order to call this function, we use `dlsym` which is a content extraction function. This function takes the library pointer and the name of the target method as parameters.

6.6 MERF interface

We implemented MERF and its GUI as a C++ desktop application with the Qt GUI toolkit. The MERF GUI uses standard intuitive tagging facilities such as the application menus, the context menus, the mouse selection mechanism, and the drag and drop mechanism to perform tagging and editing operation. The MERF GUI allows the user to create, manipulate, and analyze a tagging project.

In order to provide a user-friendly interface, we use `QDockWidget` in MERF main window. This class provides the concept of dock widgets which can be moved into new areas and removed by the user. The main window contains 4 widgets referring to text visualization, tag list, tag description, and match tree and entity-relation graph. The main window displays the text with rich colors in a text browser pane. A companion pane shows a list of all the tags. A secondary companion pane shows details about the selected tag. A tabbed widget visualizes the match tree and entity-relation graph.

As for the MBF and MRE tag type editors, we use grids to organize the layout. The MBF and MRE are represented using the tree widget provided by Qt. We use combo box, label, edit text, and push button classes provided by Qt to add the other objects in the editor.

6.6.1 Tree and graph visualization

We use graphics scene and graphics view in order to visualize the match tree and entity-relation graph. The graphics scene provides a surface to manage 2D graphical items such as lines, text, and shapes. The graphics view provides a widget to display the content of the graphics scene. Our visualization method is based on an example provided by Qt showing how to implement nodes, and edges between nodes in a graph [1]. However, Qt doesn't calculate the layout automatically but requires the user to position the elements.

In order to generate the tree or graph layout, we use the graphviz library. This library provides a variety of software for drawing attributed graphs and computes the layout using a set of common graph layout algorithms such as dot. First, we define our match tree or entity-relation graph using this library. We use commands such as `agopen` to create a graph, `agsafeset` to set graph attributes, `agnode` to create a graph node, and `agedge` to create a graph edge. After creating the graph, we compute a layout using the *dot* layout engine. This task is performed by calling the method `gvLayoutJobs`. We use the generated coordinates of the nodes in the graphviz graph to position the nodes in MERF's main window graphics scene.

6.7 Open source tool

MERF is an open source tool present on google code under `atmine` repository (`https://code.google.com/p/atmine/`). In addition to MERF, the repository contains entity extraction tasks developed by our research group. People are welcome to download and use the tools. We appreciate any feedback that we get and we try to improve the tools accordingly.

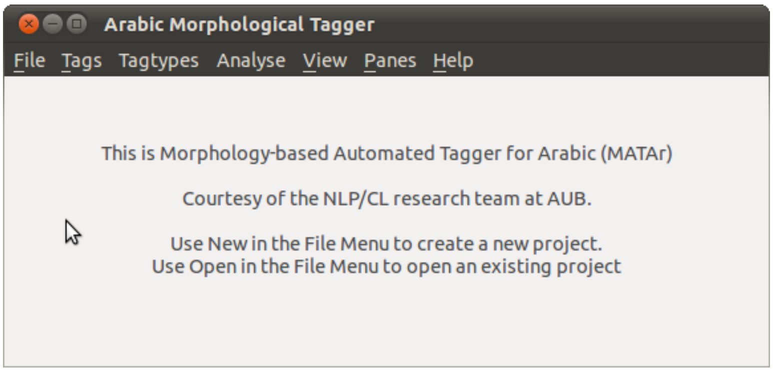
[1] `http://qt-project.org/doc/qt-4.8/graphicsview-elasticnodes.html`

Chapter 7

MERF GUI

In this Chapter, we present the user friendly interface of MERF. We show how to build the direction extraction task shown in Chapter 3. The task requires the user to define Boolean formulae, define regular expression, declare the semantic relations, and call MERF's simulators.

MERF provides a user friendly interface to specify the atomic terms, the MERF Boolean formulae, the MERF regular expressions, the tag types, and the legends. The MERF GUI also allows the user to modify and correct the resulting tag set R. The MERF GUI allows the user also to compute accuracy results that compare different tag sets. The accuracy results serve well as inter annotation agreement results when the tag sets come from two human annotators, or as evaluation results when comparing MERF output with reference tag sets.



This is Morphology-based Automated Tagger for Arabic (MATAr)

Courtesy of the NLP/CL research team at AUB.

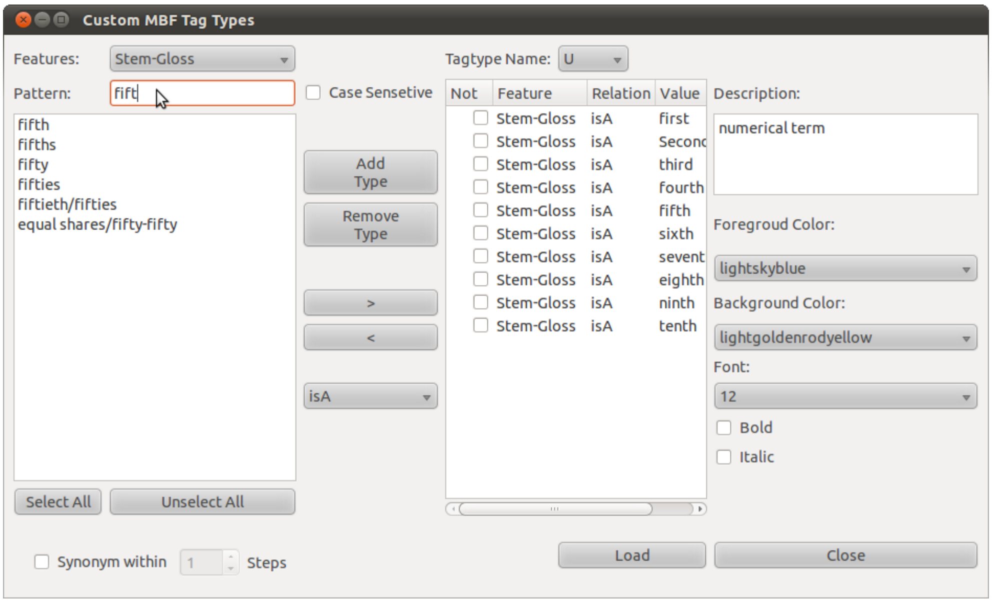
Use New in the File Menu to create a new project.
Use Open in the File Menu to open an existing project

Figure 7.1: Initial main window of MERF

The snapshot in Figure 7.1 shows the MERF GUI. MERF's GUI allows the user to create, manipulate, and analyze a project. The *File* menu is used to create, close, or open an existing project. The *Tags* menu enables the user to manually define and edit general purpose tag types with no formulae or expressions. The user can define and edit the MBF and MRE based tag types and trigger the relevant automatic morphology-based simulators from the *Tag-*

types menu. The user performs comparison and analysis of two tag sets using the *Analyse* menu. *View* menu enables the user to switch between visualizing the MBF-based tag type matches or MRE-based tag type matches. To start our direction extraction project, we create a new project.

7.1 Tag type Boolean formula editor



Custom MBF Tag Types

Features: Stem-Gloss

Pattern: fift

□ Case Sensetive

fifth
fifths
fifty
fifties
fiftieth/fifties
equal shares/fifty-fifty

Tagtype Name: U

Not	Feature	Relation	Value
□	Stem-Gloss	isA	first
□	Stem-Gloss	isA	Seconc
□	Stem-Gloss	isA	third
□	Stem-Gloss	isA	fourth
□	Stem-Gloss	isA	fifth
□	Stem-Gloss	isA	sixth
□	Stem-Gloss	isA	sevent
□	Stem-Gloss	isA	eighth
□	Stem-Gloss	isA	ninth
□	Stem-Gloss	isA	tenth

Description: numerical term

Foregroud Color: lightskyblue

Background Color: lightgoldenrodyellow

Font: 12

□ Bold

□ Italic

□ Synonym within 1 Steps

Figure 7.2: MERF tag type Boolean formula editor.

The morphology-based tag type editor shown in Figure 7.2 allows the user to write a tag type Boolean formula in a user friendly manner. The user first specifies atomic terms by selecting a feature from $\mathcal{F}$. The pattern filters the feature values. MERF editor provides the user with a combo box to specify the predicate of the match. The predicate can be an `isA` predicate requiring an exact match of the morphological feature value, or a `contains` predicate requiring a substring match.

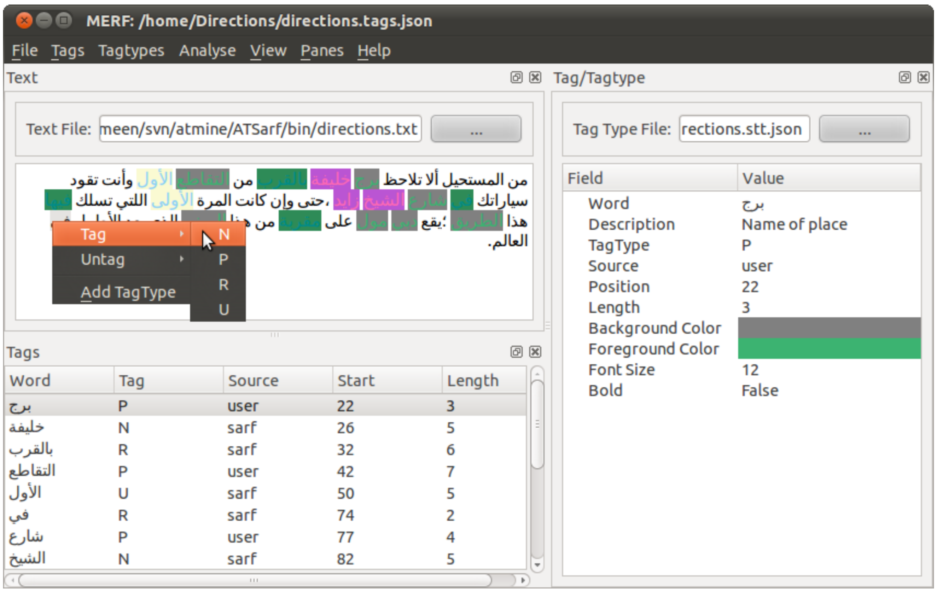
Then the user can add and remove the selected feature values to the atomic terms under the tag type name using the push buttons. The feature column has a check box that allows negating the term. The Relation column has a context sensitive menu that can switch the operation between the values in $\mathcal{O} = \{isA, contains\}$. Multiple feature/value pairs can be included in a single tag type definition with a disjunction semantics. The right pane shows a description of the tag type and a set of legend descriptors. When the stem or gloss features are selected, the user has the option to use the Syn^k feature. The user selects the check box in the lower left of the editor and sets the order in the spin box.

In the direction extraction task example, the user specifies four MBF-based tag types with labels N, P, R, and U with descriptions names of person, name of place, relative position, and numerical term, respectively. For each MBF, the user selects the morphological features, specifies the constant value CF, and adds it to the Boolean formula editor. The user also assigns legend descriptors to each defined tag type.

Figure 7.2 shows the definition of the numerical term (U) MBF. The MBF is based on an MBF formula that inspects the gloss tag of the stem of the word and checks whether it is `first`, `second`, . . . , or `tenth`. The user selects the legend descriptors by setting the foreground color to `lightskyblue`, background color to `lightgoldenrodyellow`, font size to `12`, normal font weight, and unitalicized.

7.2 MERF MBF match visualization

After defining the MBFs of the direction extraction task, we call the MBF simulator from the `Tagtypes` menu. The snapshot in Figure 7.3 shows the MERF GUI with the tag type color sensitive text view, the tag list view, and the tag description view. The color-sensitive text view shows the text with visualization of the MBF tag type matches. The tag list view shows all the tag matches that are automatically or manually applied to the text. The tag description view presents the details of the tag along with the relevant tag type information.



Text File: meen/svn/atmine/ATSarf/bin/directions.txt

من المستحيل ألا تلاحظ برج خليفة بالقرب من التقاطع الأول وأنت تقود
سياراتك في شارع الشيخ زايد، حتى وإن كانت المرة الأولى اللتي تسلك فيها
هذا الطريق؛ يقع دبي مول على مقربة من هنا
العالم.

Word	Tag	Source	Start	Length
برج	P	user	22	3
خليفة	N	sarf	26	5
بالقرب	R	sarf	32	6
التقاطع	P	user	42	7
الأول	U	sarf	50	5
في	R	sarf	74	2
شارع	P	user	77	4
الشيخ	N	sarf	82	5

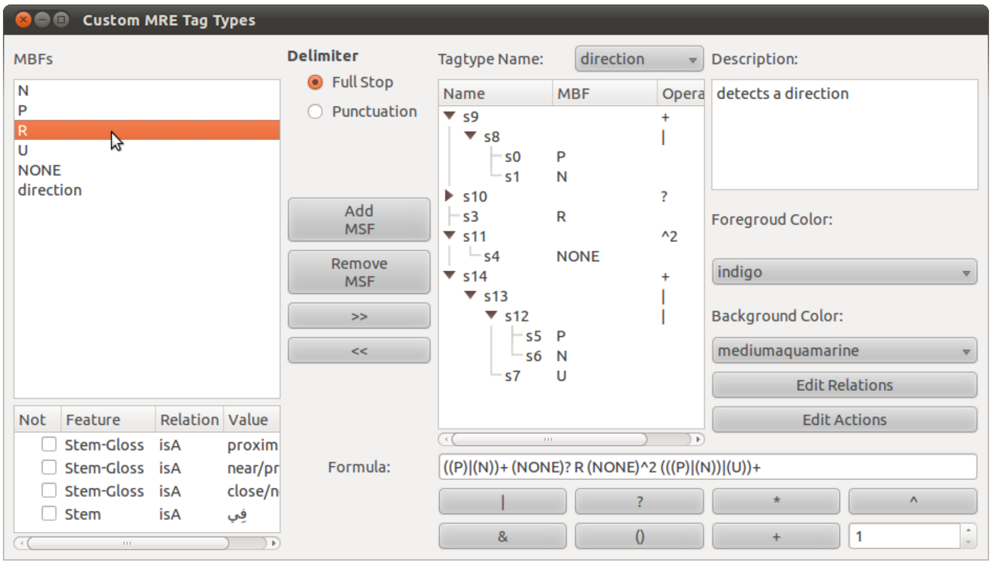
Tag Type File: rections.stt.json

Field	Value
Word	برج
Description	Name of place
TagType	P
Source	user
Position	22
Length	3
Background Color	⬛
Foreground Color	🟩
Font Size	12
Bold	False

Figure 7.3: MERF main window with MBF match annotated text, tag descriptions, tag type legend properties, and manual tag edition menus.

The context sensitive menus in Figure 7.3 allow the user to tag a selected word or chunk differently or to entirely remove the tag. MERF GUI also allows manual tag types that are not based on morphological features. These tags enable the users to build their own reference corpora without help from the morphological analyzer.

7.3 Tag type regular expression editor



MBFs: N, P, R, U, NONE, direction

Delimiter: ● Full Stop ○ Punctuation

Tagtype Name: direction

Description: detects a direction

Name	MBF	Opera
s9		+
s8		|
s0	P	
s1	N	
s10		?
s3	R	
s11		^2
s4	NONE	
s14		+
s13		|
s12		|
s5	P	
s6	N	
s7	U	

Foregroud Color: indigo

Background Color: mediumaquamarine

Not	Feature	Relation	Value
□	Stem-Gloss	isA	proxim
□	Stem-Gloss	isA	near/pr
□	Stem-Gloss	isA	close/n
□	Stem	isA	في

Formula: ((P)|(N))+ (NONE)? R (NONE)^2 (((P)|(N))|(U))+

1

Figure 7.4: MERF tag type regular expression editor.

After interacting with the tool and getting satisfied with the results, the user moves to specify the regular expressions. The morphology-based tag type editor shown in Figure 7.4 allows the user to define a tag type regular expression in a user friendly manner. The user first adds the required MBFs to the formula under the tag type name by selecting a label from $\mathcal{T}$ under MBFs. The Boolean formula of a highlighted tag type is shown in the table on the lower left pane. Each selected MBF is associated with an automatic name. The regular expression tree features the name, MBF, and operation for each sub-expression.

`NONE` is a special tag type with no Boolean formula. It tags all the words that are not tagged with any of the morphology-based Boolean tag types. `NONE` is defined by default and can be used to introduce flexibility and noise tolerance into the formula. We previously defined this tag type in Chapter **??** and referred to it by O (other). Moreover, a defined MRE is directly added to the MBF list enabling substitution and recursion in formula definition.

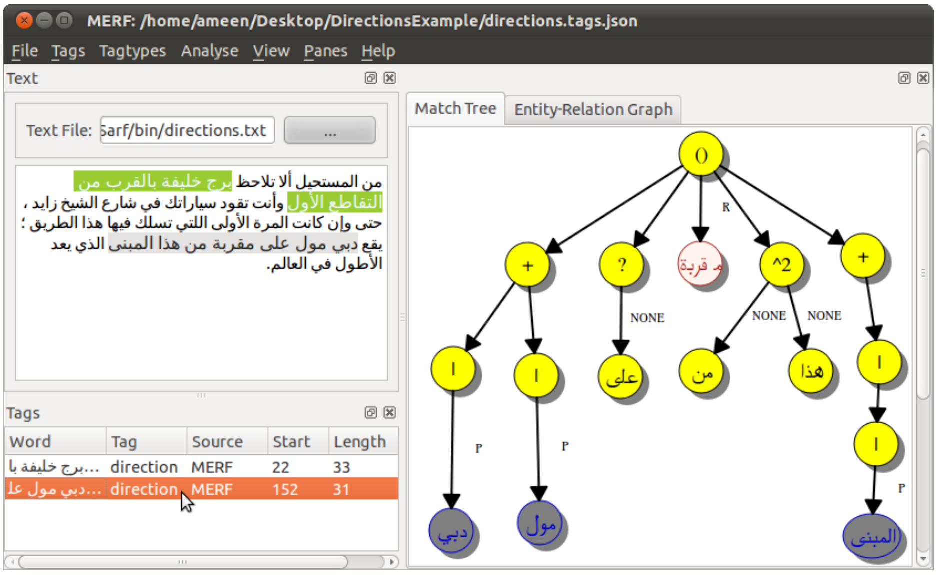
The regular expression editor enables the user to apply operations to the selected expressions. To do so, the user selects one or two expressions then selects an operation from the ones shown in the lower left of the view. The operations include disjunction, conjunction, zero or one, sequence, zero or more, one or more, and zero up to some constant integer specified by the user. The right pane shows a description of the tag type and a set of legend descriptors.

$$(P|N)+\ \ O?\ R\ O{}^\wedge 2\ (P|N|U)+$$

In the direction task, we aim to build the expression shown above. Thus we start by adding the MBFs P, N, $NONE$, R, $NONE$, P, N, and U. We apply the required operations incrementally to build the final expression. For example, we apply the disjunction (|) operation to the first P and N added, then we apply the one or more (+) operation to the resulting sub-expression. We proceed as such to build the regular expression shown in Figure 7.4. Similar to the MBF tag type editor, the user specifies a set of legend descriptors for each MRE tag type.

7.4 MERF MRE match visualization

After defining the MRE of the direction extraction task, we call the MRE simulator from the `Tagtypes` menu. The snapshot in Figure 7.5 shows the MERF GUI with the tag type color sensitive text view, the tag list view, the tag description view, and the match tree view. The color-sensitive text view shows the text with visualization of the MRE tag type matches. The tag list view shows all the tag matches that are automatically or manually applied to the text. The tag description view presents the details of the tag along with the relevant tag type information. The match tree is visualized when the user selects a tag from tag list.



MERF: /home/ameen/Desktop/DirectionsExample/directions.tags.json
File Tags Tagtypes Analyse View Panes Help
Text
Match Tree
Entity-Relation Graph
Tags
()
+
?
R
ـه قربة
^2
+
NONE
NONE
NONE
|
|
على
من
هذا
|
|
P
P
P
دبي
مول
المبنى

Text File: ʂarf/bin/directions.txt ...

من المستحيل ألا تلاحظ برج خليفة بالقرب من
التقاطع الأول وأنت تقود سياراتك في شارع الشيخ زايد ،
حتى وإن كانت المرة الأولى اللتي تسلك فيها هذا الطريق ؛
يقع دبي مول على مقربة من هذا المبنى الذي يعد
الأطول في العالم.

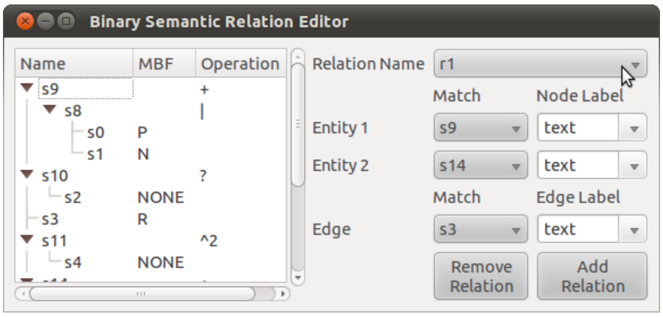
Word	Tag	Source	Start	Length
...برج خليفة با	direction	MERF	22	33
...دبي مول عل	direction	MERF	152	31

Figure 7.5: MERF main window with match tree view

Figure 7.5 shows the match tree of the direction task regular expression match دبي مول على مقربة من هذا المبنى *dby mwl ˁlā mqrbh mn hḏā ālmbnā* (Dubai Mall is located near this building). Leaf nodes show the match text, internal nodes show the operations, and edge labels shows the MBF tag type name.

7.5 Semantic relation editor

After the user is satisfied with the MRE matches, the user moves to define the semantic relations and the code actions. The semantic relation editor of MERF shown in Figure 7.6 allows the user to define relations in a user-friendly manner. The relation is defined by the tuple $\langle e_1, e_2, r \rangle$, where e_1 and e_2 are entities and r is the relational entity. The editor provides the user with the regular expression tree shown in the left pane to ease the relation construction. The user first adds a relation, then assigns the values for the entities and relational entity (edge). For each element in the tuple, the user specifies the sub-expression match and relevant label. The label can be a match text, position, length, or numerical value, or user-defined label.



Name	MBF	Operation
s9		+
s8		|
s0	P	
s1	N	
s10		?
s2	NONE	
s3	R	
s11		^2
s4	NONE	

Relation Name: r1

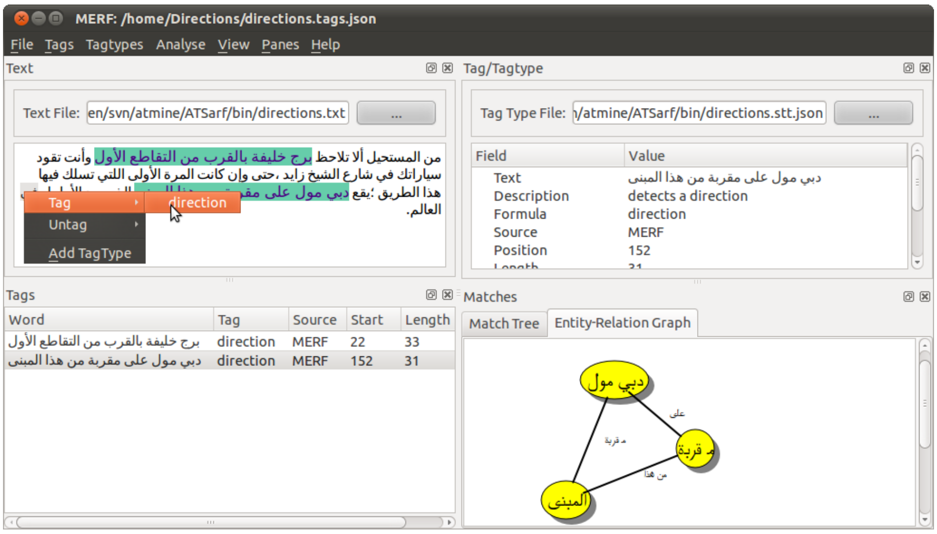
	Match	Node Label
Entity 1	s9	text
Entity 2	s14	text

	Match	Edge Label
Edge	s3	text

Figure 7.6: MERF relation editor

Recall the semantic relations we defined in the direction extraction task; namely relations $\langle e_1, e_2, r \rangle$, $\langle r, e_1, o_1 \rangle$, and $\langle r, e_2, o_2 \rangle$. Figure 7.6 shows the declaration of the first relation where s_9, s_{14}, and s_3 correspond to e_1, e_2, and r, respectively. We choose the text of the sub-expression match to be the entity node and relational entity edge labels. Similarly, we define the other relations $\langle s_3, s_9, s_{10} \rangle$, and $\langle s_3, s_{14}, s_{11} \rangle$ corresponding to $\langle r, e_1, o_1 \rangle$, and $\langle r, e_2, o_2 \rangle$, respectively.

After specifying the semantic relations, we call the semantic relation constructor. The snapshot in Figure 7.7 shows the MERF GUI with the entity-relation graph. The entity-relation graph is visualized when the user selects an MRE match tag from the tag list. Figure 7.7 shows the entity-relation graph of the match shown in Figure 7.5. The interactive graph view enables the user to move the nodes.



Text File: en/svn/atmine/ATSarf/bin/directions.txt

Tag Type File: n/atmine/ATSarf/bin/directions.stt.json

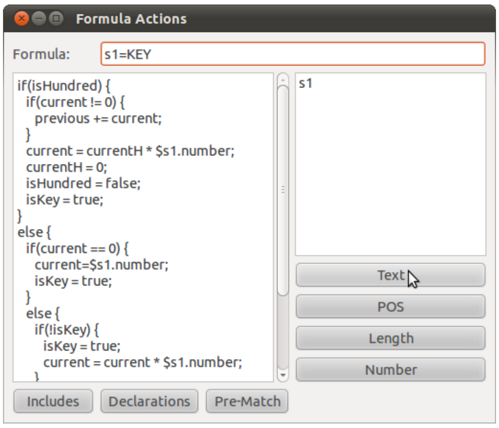
من المستحيل ألا تلاحظ برج خليفة بالقرب من التقاطع الأول وأنت تقود
سياراتك في شارع الشيخ زايد ،حتى وإن كانت المرة الأولى اللتي تسلك فيها
هذا الطريق ؛يقع دبي مول على
العالم.

Field	Value
Text	دبي مول على مقربة من هذا المبنى
Description	detects a direction
Formula	direction
Source	MERF
Position	152
Length	31

Word	Tag	Source	Start	Length
برج خليفة بالقرب من التقاطع الأول	direction	MERF	22	33
دبي مول على مقربة من هذا المبنى	direction	MERF	152	31

Figure 7.7: MERF main window with entity-relation view

7.6 Code action editor



Formula: s1=KEY

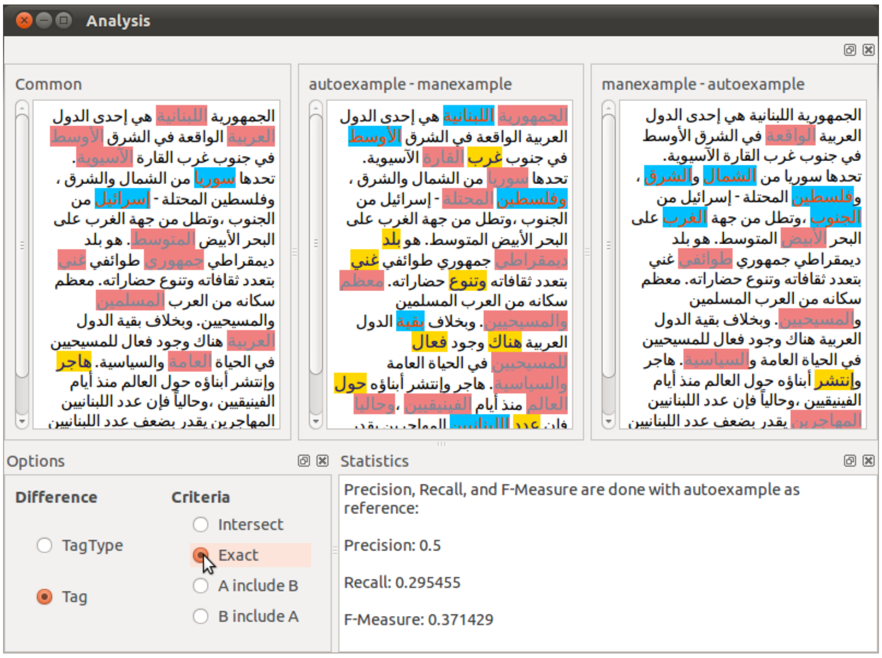
```
if(isHundred) {
  if(current != 0) {
    previous += current;
  }
  current = currentH * $s1.number;
  currentH = 0;
  isHundred = false;
  isKey = true;
}
else {
  if(current == 0) {
    current=$s1.number;
    isKey = true;
  }
  else {
    if(!isKey) {
      isKey = true;
      current = current * $s1.number;
    }
```

Figure 7.8: MERF MRE action editor

In order to add computational actions to an MRE sub-expression, the user selects to edit the actions in the MRE tag type editor. The view shown in Figure 7.8 allows the user to specify the actions in a user friendly manner. MERF provides the user with an API to access the sub-expression match features easily and define the pre-match actions. The match features include the text, position, length, and numerical value if applicable, and morphological features for MBF matches. MERF also enables the user to add C++ declarations and library includes.

7.7 Analysis

In addition to automatic and manual tagging, MERF allows comparing tag sets and tag types applied to the same input text. The MERF comparator takes as input two tag sets R_1 and R_2 and two tag type sets $\mathcal{T}_1$ and $\mathcal{T}_2$. It produces a difference view for the tag types and a difference view for the tag sets. The tag type difference view shows the common tag types $\mathcal{T}_1 \cap \mathcal{T}_2$, the tag types in $\mathcal{T}_1$ and not in $\mathcal{T}_2$, and the tag types in $\mathcal{T}_2$ and not in $\mathcal{T}_1$.



Common

الجمهورية اللبنانية هي إحدى الدول
العربية الواقعة في الشرق الأوسط
في جنوب غرب القارة الآسيوية.
تحدها سوريا من الشمال والشرق ،
وفلسطين المحتلة - إسرائيل من
الجنوب ،وتطل من جهة الغرب على
البحر الأبيض المتوسط. هو بلد
ديمقراطي جمهوري طوائفي غني
بتعدد ثقافاته وتنوع حضاراته. معظم
سكانه من العرب المسلمين
والمسيحيين. وبخلاف بقية الدول
العربية هناك وجود فعال للمسيحيين
في الحياة العامة والسياسية. هاجر
وإنتشر أبناؤه حول العالم منذ أيام
الفينيقيين ،وحالياً فإن عدد اللبنانيين
المهاجرين يقدر بضعف عدد اللبنانيين

autoexample - manexample

الجمهورية اللبنانية هي إحدى الدول
العربية الواقعة في الشرق الأوسط
في جنوب غرب القارة الآسيوية.
تحدها سوريا من الشمال والشرق ،
وفلسطين المحتلة - إسرائيل من
الجنوب ،وتطل من جهة الغرب على
البحر الأبيض المتوسط. هو بلد
ديمقراطي جمهوري طوائفي غني
بتعدد ثقافاته وتنوع حضاراته. معظم
سكانه من العرب المسلمين
والمسيحيين. وبخلاف بقية الدول
العربية هناك وجود فعال
للمسيحيين في الحياة العامة
والسياسية. هاجر وإنتشر أبناؤه حول
العالم منذ أيام الفينيقيين ،وحاليا
فان عدد اللبنانيين المهاجرين يقدر

manexample - autoexample

الجمهورية اللبنانية هي إحدى الدول
العربية الواقعة في الشرق الأوسط
في جنوب غرب القارة الآسيوية.
تحدها سوريا من الشمال والشرق ،
وفلسطين المحتلة - إسرائيل من
الجنوب ،وتطل من جهة الغرب على
البحر الأبيض المتوسط. هو بلد
ديمقراطي جمهوري طوائفي غني
بتعدد ثقافاته وتنوع حضاراته. معظم
سكانه من العرب المسلمين
والمسيحيين. وبخلاف بقية الدول
العربية هناك وجود فعال للمسيحيين
في الحياة العامة والسياسية. هاجر
وإنتشر أبناؤه حول العالم منذ أيام
الفينيقيين ،وحالياً فإن عدد اللبنانيين
المهاجرين يقدر بضعف عدد اللبنانيين

Difference

○ TagType

● Tag

Criteria

○ Intersect

● Exact

○ A include B

○ B include A

Precision, Recall, and F-Measure are done with autoexample as reference:

Precision: 0.5

Recall: 0.295455

F-Measure: 0.371429

Figure 7.9: MERF comparison and accuracy results view.

Similarly, the tag set difference view shows $R_1 \cap R_2$, R_1/R_2 and R_2/R_1. The tag set difference view, as shown in Figure 7.9 also shows the precision, recall and F-measure between the two sets. The metrics can be computed based on several predicates. The "Intersection" predicate returns true if a tag from R_1 intersects in text T with a tag in R_2. The "Exact" predicate returns true if a tag from R_1 exactly matches a tag in R_2. The "A includes B" predicate returns true if a tag from R_1 contains a tag from R_2. Finally, the "B includes A" predicate returns true if a tag from R_2 contains a tag from R_1.

In the difference view panes, the user can select a difference tag and accept it, or reject it to build a corrected corpora.

Chapter 8

Case Studies

In order to evaluate MERF, we developed a set of twelve case studies with an information extraction task for each. We compare the results of MERF with existing application specific techniques for three of the case studies. We discuss the three case studies and the manually encoded techniques used in them in the following. We discuss the rest of the case studies in Appendix A.

We consider the narrator chain, number normalization, and temporal entity extraction tasks that were addressed in ANGE [27], NUMNORM, and ATEEMA [28], respectively. We compare the results of MERF with the results from the entity extraction tasks and we report the time required to implement the application, the runtime to extract the target entities, the accuracy (recall and precision) of the output tags, and the ease of composition as a complexity metric (lines of code).

Recall refers to the fraction of the entities correctly detected against the total number of entities available. Precision refers to the fraction of correctly detected entities against the total number of extracted entities. Intuitively, the precision measure denotes whether the system generated false positives.

We used newspaper articles to evaluate number normalization, and temporal entity extraction, and a hadith book to evaluate narrator chain extraction. For the temporal and number normalization cases, we evaluated the techniques against text chosen arbitrary from issues of the Lebanese Assafir newspaper[1] and the Lebanese Al-Akhbar newspaper[2]. For the narrator chain case, we evaluate the techniques against part of a hadith book (Musnad Ahmad). We present our results in Table 8.1.

[1] available online at `http://www.assafir.com`.

[2] available online at `http://www.al-akhbar.com`.

Table 8.1: MERF against manually-coded applications: Time, accuracy, and ease of composition

Task		Build Time	Run Time(s)	Accuracy		Ease of Composition
				Recall	Precision	
Narrator Chain	ANGE	1-2 month	1.79	1	1	3000+ lines of code
	framework	3 hours	7.24	1	0.93	8 MBFs and 4 MREs
Number Normalization	NUMNORM	1 week	0.32	0.91	0.93	500 lines of code
	framework	1 hour	1.53	0.91	0.90	3 MBFs/1 MRE/57 lines
Temporal Entity	ATEEMA	1-2 month	2.53	0.88	0.89	1000+ lines of code
	framework	3 hours	3.14	0.91	0.81	3 MBFs and 2 MREs

8.1 Narrator chain

In this application, our target is to detect the narrator chains. A narrator chain is a sequence of narrators referencing each other. A sample narrator chain is shown in Table 8.2. The chain includes proper nouns (names) and connectors expressing paternal relations and referencing. This chain is usually found in hadith books and is referred to as *sanad*. ANGE uses Arabic morphological analysis, finite state machines, and graph transformations to extract named entities and relations including the narrator chains [27]. Accordingly, we design the following MERF MBFs and MRE to extract the narrator chains.

First, we define the MBF-based tag types. *PN* stands for "proper noun" and is defined by the abstract category `Name of Person`. *FAM* stands for "family connector" and it is defined by the stem gloss "son". The MBF *TOLD* is used to detect the referencing between the narrators and is defined the disjunction formula of the stems حدث*ḥdṯ* , عن*ʿn* , سمع*smʿ* , أخبر*ʾḫbr* , and أنبأ*ʾnb-ʾ* . Also, we define the MBF *MEAN* to tag the stem عني*ʿny* which exists sometimes as part of a name such as عاصم يعني ابن محمّد*ʿāṣm yʿny ābn mḥmmd* (Asim meaning son of mohammad). *BLESS*, *GOD*, *UPONHIM*, and *GREET* MBFs are defined by the stems صلّى*ṣllā* , اللّه*āl-lāh* , علي*ʿly* , and سلّم*sllm* , respectively. These MBFs are used to detect the text segment صلّى اللّه عليه وسلّم*ṣllā āl-lāh ʿlyh wsllm* (peace be upon him) which refers to prophet Mohammad. We use NONE to introduce flexibility to the expression.

We illustrate the MREs defined to detect a narrator chain. The expression *name* is defined as one or more *PN* with optional *MEAN*.

```
name : PN ((MEAN)? PN)*;
```

nar, stands for narrator, is defined as *name* followed by zero or more sequences of *FAM* followed by *name* with optional *NONE* for flexibility.

```
nar : name ((NONE)^3 FAM (NONE)^3 name)*;
```

The expression *pbuh* stands for "peace be upon him" and is defined by the sequence of *BLESS*, *GOD*, *UPONHIM*, and *GREET*.

```
pbuh : BLESS GOD UPONHIM GREET;
```

We define *nchain*, denoting narrator chain, as *TOLD* followed by *nar* repeated one or more times, followed by an optional sequence of up to eight *PN*, *FAM*, or *NONE* tags followed by a match of the *pbuh* expression. This optional part is used to detect a possible chain of the narrators to prophet Mohammad.

```
nchain : (TOLD nar)+ ((((PN)|(FAM))|(NONE))^8 pbuh)?
```

Table 8.2: Narrator chain example

القعقاع *ālqʿqāʿ*	بن *bn*	عمارة *ʿmārh*	عن *ʿn*	جرير *ǧryr*	حدثنا *ḥdṯnā*	سعيد *sʿyd*	بن *bn*	قتيبة *qtybh*	حدثنا *ḥdṯnā*
PN	FAM	PN	TOLD	PN	TOLD	PN	FAM	PN	TOLD
name		name		name		name		name	
nar				nar		nar			
nchain									

Table 8.2 shows an example match for the *nchain* regular expression. The first row in the table is the input Arabic text. In the second row, each word is tagged with MBFs based on its morphological features. The names are extract using the *name* MRE. Then, the narrators are detected using the *nar* expression. Last, the *nchain* match is detected as a combination of narrators (nar) and referencing (TOLD).

We evaluated the narrator chain extraction in ANGE and MERF using part of a hadith book (Musnad Ahmad) that we obtained from online sources. We report time to implement the model, runtime to extract the narrator chains, accuracy (recall and precision), and ease of composition (lines of code) as our evaluation metrics for both techniques. The first row in Table 8.1 shows the evaluation metrics for both manual coded work in ANGE and MERF. The build time metric presents a great advantage for MERF where the implementation of the model requires hours in comparison to one to two month for manually coded approach. This advantage for MERF also appears in the ease of composition metric where more than 3000 lines of code are required in ANGE compared to eight MBFs and 4 MREs using a user-friendly GUI. However, ANGE has a lower runtime with 1.79 seconds compared to 7.24 seconds for MERF. The recall in both techniques is 100% which means that all the narrator chains present in the text were detected. MERF scored 93% for precision while ANGE showed 100% precision. This value means that ANGE's matches were all correct while MERF had some incorrect matches for narrator chains. A sample mismatch in MERF was عن الوصال*ʿn ālwṣāl* where الوصال*ālwṣāl* was wrongly detected as a *PN*. The precision can be increased by filtering the name matches.

8.2 Number normalization

In this application, we are interested in number normalization in Arabic text. Our target is to extract text chunks that express numbers and normalize them. Number normalization is the process of converting numbers in text into real numbers. For example the normalization of عشرون الف*ʿšrwn alf* (twenty thousand) would be 20,000. We implemented a manually coded application for number normalization called *NUMNORM*. We also implemented the same application using MERF and compared the results.

NUMNORM reads an Arabic text as input, extracts the text chunks with numerical information, and normalizes them. NUMNORM uses the inhouse morphological analyzer, *Sarf*, to identify the words referring to numbers. Sarf provides a `Number` category that tags all words with numerical information. In the normalization algorithm, NUMNORM identifies three different categories for numbers; each with different behavior in normalization. The categories are:

- DT, denoting digits and tens
- H, denoting hundred
- TMB, denoting thousand, million, and billion

NUMNORM retrieves the number referring to a word through a fixed map. This map takes the word gloss as input and returns the matching number if found. For example, the map returns 100 for the input gloss `hundred`. As for the normalization of more than one word, such as ألفين وأربعة عشر*ʾlfyn w-ʾrbʿh ʿšr* (2014), the normalization algorithm used for each category is shown in

```
if(isHundred) {
   if(current != 0) {
      previous += current;
   }
   current = currentH * matchNumber;
   currentH = 0;
   isHundred = false;
   isKey = true;
}
else {
   if(current == 0) {
      current = matchNumber;
      isKey = true;
   }
   else {
      if(!isKey) {
         isKey = true;
         current = current * matchNumber;
      }
      else {
         previous += current;
         current = matchNumber;
      }
   }
}

TMB Algorithm
```

```
if(isHundred) {
   currentH += matchNumber;
}
else {
   if(current == 0) {
      current = matchNumber;
   }
   else {
      if(isKey) {
         previous += current;
         current = matchNumber;
      }
      else {
         current += matchNumber;
      }
   }
}
isKey = false;

DT Algorithm
```

```
isHundred = true;
if(current == 0)  {
   currentH = matchNumber;
}
else {
   if(!isKey) {
      currentH= current * matchNumber;
      current = 0;
   }
   else {
      currentH = matchNumber;
   }
}
isKey = false;

H Algorithm
```

Figure 8.1: NUMNORM algorithm for TMB, DT, and H.

Figure 8.1. The algorithm uses Boolean variables and three integer variables. The three integer variables are previous, current, and currentH for hundred. The main steps in the algorithm are as follows. Hundred is multiplied to previous digit if found, else saved. Digits and tens are added to previous hundred if found, else added to current. TMB match is multiplied by the previous hundred or DT if found, else saved in current.

Using MERF, we designed a morphology-based regular expression to detect the numerical chunks in the text. Based on the previously introduced algorithm, we added actions to this MRE to normalize each match. First, we defined the MBFs *DT*, *H*, and *TMB*. The MBF *DT* denotes digits and tens, and is defined by a disjunction formula of the stem glosses `one`, `two`, ..., `ten`, `twenty`, ..., and `ninety`. The MBF *H* is defined by the stem gloss `hundred`. The MBF *TMB* is defined by a disjunction formula of the stem glosses `thousand`, `million`, and `billion`.

The *number* MRE is defined as one or more matches of *DT*, *TMB*, or *H*. The actions associated with each MBF are shown in Figure 8.1.

```
number : (((DT)|(TMB))|(H))+;
```

We evaluated NUMNORM and MERF against an article chosen arbitrarily from the economical section of Assafir newspaper 25/12/2013 issues. We report the build time, runtime, accuracy, and ease of composition as our evaluation metrics for NUMNORM and MERF. The second row in Table 8.1 shows the metrics for both techniques. Using MERF to perform normalization takes 1 hour while NUMNORM implementation required one to two month. NUMNORM takes 0.32 seconds to perform the task while the MERF model needs

1.53 seconds. As for ease of composition, MERF requires the construction of 3 MBFs and 1 MRE with code actions of 57 lines while NUMNORM requires more than 500 lines of code. Both techniques have the same recall (91%) which means that both detected the same correct number of matches. However, NUMNORM has a higher precision (93%) than MERF (90%) which means that NUMNORM has less false matches.

8.3 Temporal entity extraction

In this case study, we are interested in extracting temporal entities. Temporal entities are text chunks that express or infer temporal information. Some of these entities represent absolute time and dates such as الخامس من آب ٢٠١٠ *ālḫāms mn ʾāb 2010* . Others represent relative time such as بعد خمسة أيام *bʿd ḫmsh ʾayām* and others represent quantities such as ١٤ يوما *14 ywmā* . *ATEEMA* presents a temporal entity detection technique for the Arabic language using morphological analysis and a finite state transducer [28]. Hence, we design our expressions and compare the tag results with ATEEMA.

The MBFs defined for this purpose are introduced in Table 8.3. The Details and Examples columns show the type of the atomic terms included in each MBF. *TIME* defines explicit temporal words. *NUM* denotes numerals and includes digits or words referring to numbers. As for *TIMEPREP*, it denotes temporal prepositions which precedes or follows a time expression.

Table 8.3: Temporal MBFs

MBF	Details	Examples
TIME	unit	دقيقة*dqyqh* (minute)
	relative	غد*ġd* (tomorrow)
	range	ربيع*rbāʿ* (spring)
	nominal	احد*āḥd* (Sunday)
	events	هجرى*hğrā*
NUM	digit	٣*3* (3)
	word	ثمانن*ṯmānān* (eighty)
	range	ثمانينيات*ṯmānynyāt* (eighties)
TIMEPREP	point	في*fy* (in)
	relative	قبل*qbl* (before)
	approximate	نحو *nḥw* (about)
	range	خلال*ḫlāl* (during)

The MERF MREs constructed are shown below. The expression *mts*, stands for `maybe time start`, and detects words which are either tagged by *NUM* or *TIMEPREP*. The expression *definitetime* is defined by zero or more sequence of *mts* and two optional *NONE*, *TIME*, then a zero or more sequence of optional two *NONE* followed by *mts* or *TIME*.

```
mts : NUM | TIMEPREP;
definitetime : (mts (NONE)^2)* TIME ((NONE)^2 (mts)|(TIME))*;
```

Table 8.4: Temporal entity example

السنوات*ālsnwāt*	أصعب*ʾṣʿb*	من*mn*	وخمس*wẖms*	عجاف*ʿǧāf*	أشهر*ʾšhr*	أربعة*ʾrbʿh*	بعد*bʿd*
TIME	NONE	NONE	NUM	NONE	TIME	NUM	TIMEPREP
			mts			mts	mts
definitetime							

Table 8.4 shows a sample temporal entity match. The Arabic words are tagged by the relevant MBFs if matching any. In case there was no match, a word is tagged by *NONE*. Then, matches of the defined expressions are detected as shown in the table.

We evaluated ATEEMA and MERF against articles chosen arbitrarily from the political section of Al-Akhbar 17/6/2011 and 27/12/2013 issues and sports section of Al-Akhbar 24/12/2013 issue. We report the build time, runtime, accuracy, and ease of composition as our evaluation metrics for ATEEMA and MERF. The third row in Table 8.1 shows the metrics for both techniques. The time metric shows an advantage of MERF over ATEEMA with three hours compared to one to two month. The same advantage for MERF approach appears in the ease of composition. ATEEMA contains more than one thousand lines of code while MERF the definition of three MBFs and two MREs. However, ATEEMA has a lower runtime with 2.53 seconds compared to 3.14 seconds for MERF. MERF shows higher recall (91%) compared to ATEEMA (88%) which means that MERF detected more temporal entities than ATEEMA. However, ATEEMA presents higher precision (89%) than MERF (81%). This indicates than the MRE in MERF is underfitting which leads to incorrect temporal matches. An incorrect temporal match that MERF detects is عقد*ʿqd* which has has the two meanings `contract` and `decade`.

Chapter 9

Related Work

9.1 Information extraction

The work in [16] presents a common pattern specification language called CPSL. CPSL grammar consists of three parts. The declaration part specifies the name of the grammar and the annotations to be used. The rule definition part defines the patterns and relevant actions. The third part is concerned with defining macros which are pure text substitution. Each macro consists of pattern part and rule part same as a rule definition. The target of CPSL is to specify IE rules in a relatively system-independent way. The grammar we propose in MERF is an extension to CPSL with support for code action execution and semantic relation construction.

SystemT is a system based on an algebraic Approach to Declarative information extraction (IE)[17]. It uses declarative rule language, Annotation Query Language (AQL), and an optimizer to generate high performance execution plans for the rules. The AQL query is translated into an algebraic expression and an optimizer selects an execution plan. The target of this system is to overcome the expressivity and performance limitations of CPSL. MERF overcomes the limitations described by SystemT. Simulating multiple tags for each word overcomes the lossy sequencing caused by random annotation dropping. MERF introduces the conjunction operation in the MRE to overcome the expressivity limitations of expressing rules with overlapping annotations. MERF has the advantage of providing a user-friendly interactive interface, code action execution, and semantic relations construction.

The work in [18] presents a semi-automatic approach for structured data acquisition using TEXTMARKER, a rule-based IE system. This system takes user-specified simple rules that consider features of the text. This work requires the manual work of a domain specialist in order to construct the rules. MERF provides a user-friendly interface for the user who is not expected to be an expert. Moreover, MERF extracts semantic relations based on user defined relations.

The work in [19] presents a user-driven relational model targeting entity-relation extraction. In this model, the user enters a natural language query. An NLP engine parses the query and extracts possible entities and relations of which a query model is generated. This tool extracts entities and relations automatically based on the user natural language query while MERF require the user to define the target entities and relations and provides cross-reference relations based on an `isA` relation. MERF provides the user with code actions with API access to match features.

QARAB is a question answering system supporting Arabic [20]. It takes an Arabic natural language query and attempts to provide short answers for it. QARAB tackles the problem using traditional information retrieval techniques and a sophisticated NLP approach. A question is classified based on a set of known question types. However, QARAB is limited to a set of predefined tag types as compared with MERF. It also lacks support for relational entity extraction.

9.2 WordNet

WordNet is a lexical reference system that mimics human lexical memory inspired by psycholinguistic theories. The Syn^k feature in our framework is inspired by this system. The WordNet lexicon is composed of five categories: nouns, verbs, adjectives, adverbs, and function words; each category with its distinct organization to better represent the psychological complexity of lexical knowledge [29, 30, 31].

By convention, a word associates a lexicalized concept with an utterance, plays a syntactic role in a sentence. Word form, i.e. sequence of alphabet symbols that represent the word, denotes the utterance. The word meaning denotes the lexicalized concept. Researchers addressed the representation of the word meaning in the absence of a psychological theory.

Lexical semantics considers the word meaning representation through definitions. The constructive theory states that a definition must contain sufficient information to support accurate construction of a word meaning. The differential theory only requires the distinction between different concepts using some symbols. WordNet adopted the differential theory to represent the word meanings over the constructive theory to avoid its constraints. Therefore, different word meanings are represented by sets of word forms. For instance, {board,plank} and {board,committee} are two word meanings referring to two different senses of the word board. This set of word forms representing a unique meaning is referred to as “synset”.

Sematic relations refer to relations between word meanings/synsets. Lexical relations refer to relations between word forms. The organization of the categories in WordNet are based on those two types of relations. Some of the relations are:

Synonymy Two expressions are synonymous in a context C if the truth value doesn't change upon the substitution of one for the other in that context. For example, "plank" and "board" are synonyms since they are substitutable in the context referring to a piece of wood for some specific purpose. This definition of synonymity necessitates the partition of WordNet into different categories. Words from different categories can't be joined into a single synset since they are not interchangeable.

Antonymy The antonym of a word X can sometimes be not-X, however this is not always the case. This relation is defined between word forms not meanings. We can't consider synsets A and B to be antonyms even if a word x in A is the antonym of y in B. Antonymy provides the central organizing principle for adjectives and adverbs in WordNet. For example, "rise" is the antonym of "fall".

Hyponymy (ISA) A meaning represented by synset X={x,x',...} is said to be a hyponym of meaning Y={y,y',...} if we accept sentences constructed as "An x is a y". We refer to Y as the superordinate of X, and hyponyms of Y as coordinate terms. For example, "tree" and "flower" are hyponyms of "plant". Also, they represent coordinate terms since they share the same superordinate Y. This relation is transitive and asymmetrical and with a single superordinate generates a hierarchical semantic structure.

Meronymy (HASA) Meronymy is a semantic part-whole relation. Synset X={x,x',...} is a meronym of synset Y={y,y',...} if we accept sentences constructed as "y has an x" or "x is part of y". For example, apple is a meronym of an apple tree.

Morphological Relations WordNet had to deal with inflectional morphology in order for the system to be of any practical use. Given the word "trees", WordNet should recognize it as "tree" in plural form. However, WordNet doesn't deal with derivational morphology but considers its addition to be a great contribution to the lexical system.

9.3 Tagging

The work in [32] presents a collaborative effort towards morphological and syntactic annotation of the Quran. The task is held through online supervised collaboration using a multi-stage approach that includes automatic POS tagging, manual verification, and online supervised collaborative proofreading. Moreover, the work in [33] presents a framework for interlingual annotation of parallel text corpora with multi-level representations. This corpora is important for NLP tasks including text summarization, information retrieval, and machine translation. An overview of annotation tools and their Arabic-English word alignment issues concludes with a set of rules and guidelines needed in an Arabic annotation alignment tool [34]. The work in [35] presents the integration of the Standard Arabic Morphological Analyzer (SAMA) into the annotation workflow of the Arabic Treebank. Such tasks motivated us to build MERF, a morphology-based open source annotation tool for the Arabic language.

MMAX2 is a manual multi-level linguistic annotation tool with an XML based data model [36]. It enables the user to create, browse, visualize, and query annotations and may be able to resolve coreference tags. BRAT is a multi-lingual user friendly manual web-based annotator that allows the construction of entity and relation annotation corpora [37]. BRAT provides an API for automated annotators to provide annotations. WordFreak is similar to BRAT. It supports Arabic text and can be extended through a plug-in architecture to integrate with NLP and CL tasks. The plug-in API may enable the use of automatic annotators along with customized visualization and annotation specifications [38]. AGTK is a toolkit for the development of text and speech annotation tools [39]. It provides import APIs from other data and graphical user interface (GUI) components. The work in [40] presents the extension of TrEd, a customizable general purpose tree editor, with the Arabic MorphoTrees annotation. The MorphoTrees present the morphological analyses in a hierarchical organization based on common features.

MERF differs from MMAX2, BRAT, WordFreak, AGTK, and TrEd in that it allows the user to specify sophisticated tag types using Boolean formulae of Arabic morphological features. These are key in CL and NLP for Arabic text. Fassieh is a commercial Arabic text annotation tool that enables the production of large Arabic text corpora [41]. The tool supports Arabic text factorizations including morphological analysis, POS tagging, full phonetic transcription, and lexical semantics analysis in an automatic mode. Fassieh is not directly accessible to the research community and requires commercial licensing. MERF is open source and differs in that it allows the user to build tag types using Boolean formulae of several atomic terms and define regular expressions based on the Boolean formulae.

Task specific annotation tools such as [42] uses enunciation semantic maps to automatically annotate directly reported Arabic and French speech. AraTation is another task specific tool for semantic annotation of Arabic news using web ontology based semantic maps [43]. We differ in that MERF is general, and not task specific, and it uses morphology-based features as atomic terms.

Chapter 10

Conclusion

In this work, we presented a morphology-based entity and relational entity information extraction framework for Arabic text; MERF. MERF provides a user-friendly interface where the user defines tag types and associates them with regular expressions defined over Boolean formulae. The Boolean formulae are terms, negations of terms, and disjunctions of terms where terms are matches to Arabic morphological features. In addition to the morphological features, MERF introduces Syn^k; a semantic feature that relates words based on synonymity. The editor allows the user to associate code actions with each regular sub-expression and to define semantic relations between sub-expressions. MERF uses an in house Arabic morphological analyzer to compute morphological matches. MERF then computes regular expression matches, and then builds the relations. We evaluated our work with several case studies comparing the results with existing application-specific techniques. The results show good accuracy for MERF compared to manually-coded techniques.

10.1 Future work

As future work, we plan the following.

- Currently, the MERF supports one built in cross-reference predicate based on the Syn^2 feature. In the future, MERF will support user defined cross-reference predicates.

- In the future, we will provide cross-document analysis in MERF to support applications similar to [27].

List of Figures

1.1 Text, formula, and match MERF example 2

1.2 Syn^2(طعام*ṭʿām*) 4

2.1 FSM of Sarf 8

3.1 Text, formula, and match MERF example 13

4.1 MERF flow diagram. 17

5.1 Syn^2(طعام*ṭʿām*) 23

5.2 Morphology-based atomic term examples 24

5.3 Morphology-based Boolean formula examples 25

5.4 Morphology-based regular expression example 26

5.5 Sample code for an on-match action 27

5.6 MERF expression, NFA, and match example 29

5.7 Semantic relation example 30

6.1 Tag file format 33

6.2 Tag type file format 33

6.3 sample generated actions file 35

6.4 Star expression NFA 38

7.1 Initial main window of MERF 41

7.2 MERF tag type Boolean formula editor. 42

7.3 MERF main window with MBF match annotated text, tag descriptions, tag type legend properties, and manual tag edition menus. 43

7.4 MERF tag type regular expression editor. 44

7.5 MERF main window with match tree view 45
7.6 MERF relation editor 46
7.7 MERF main window with entity-relation view 47
7.8 MERF MRE action editor 47
7.9 MERF comparison and accuracy results view. 48

8.1 NUMNORM algorithm for TMB, DT, and H. 53

List of Tables

2.1 Sample solution vector for the text يَأْكُلُه*yaʾakulh* 7

2.2 Word with different interpretations 9

2.3 Sample solution vector 10

5.1 Detailed example of computing Sy(l_5) 22

5.2 Example of computing Sy^2(l_5) 22

8.1 MERF against manually-coded applications: Time, accuracy, and ease of composition 50

8.2 Narrator chain example 51

8.3 Temporal MBFs 54

8.4 Temporal entity example 55

A.1 Economic Analysis Example 71

A.2 Salat Example 71

A.3 Football Result Example 73

A.4 Geographical Information Example 74

A.5 Notification Example 75

A.6 Professions Example 76

A.7 Money Example 77

A.8 Crop Info Example 78

Bibliography

[1] S. Linckels and C. Meinel, "Natural language processing," in *E-Librarian Service*, pp. 61–79, Springer, 2011.

[2] S. Ferilli, "Natural language processing," in *Automatic Digital Document Processing and Management*, pp. 199–222, Springer, 2011.

[3] V. Märgner and H. El Abed, *Guide to OCR for arabic scripts.* Springer, 2012.

[4] A. M. Rashwan, M. A. Rashwan, A. Abdel-Hameed, S. Abdou, and A. H. Khalil, "A robust omnifont open-vocabulary arabic ocr system using pseudo-2d-hmm," in *IS&T/SPIE Electronic Imaging*, pp. 829707–829707, International Society for Optics and Photonics, 2012.

[5] N. Habash and F. Sadat, "Arabic preprocessing schemes for statistical machine translation," in *Human Language Technology Conference of the NAACL*, NAACL-Short '06, pp. 49–52, 2006.

[6] I. A. Al-Sughaiyer and I. A. Al-Kharashi, "Arabic morphological analysis techniques: A comprehensive survey," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 3, pp. 189–213, 2003.

[7] A. Soudi, G. Neumann, and A. van den Bosch, *Arabic computational morphology: knowledge-based and empirical methods.* Springer, 2007.

[8] W. Zaghouani, B. Pouliquen, M. Ebrahim, and R. Steinberger, "Adapting a resource-light highly multilingual named entity recognition system to arabic," in *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC10)*, pp. 563–567, 2010.

[9] H. Traboulsi, "Arabic named entity extraction: A local grammar-based approach," in *Computer Science and Information Technology, 2009. IMCSIT'09. International Multiconference on*, pp. 139–143, IEEE, 2009.

[10] J. Makhlouta, F. A. Zaraket, and H. Harkous, "Arabic entity graph extraction using morphology, finite state machines, and graph transformations," in *Computational Linguistics and Intelligent Text Processing, CICLing*, pp. 297–310, 2012.

[11] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[12] M. Maamouri, A. Bies, T. Buckwalter, and W. Mekki, "The penn arabic treebank: Building a large-scale annotated arabic corpus," in *NEMLAR Conference on Arabic Language Resources and Tools*, pp. 102–109, 2004.

[13] N. Xue, F. Xia, F.-D. Chiou, and M. Palmer, "The penn chinese treebank: Phrase structure annotation of a large corpus," *Natural Language Engineering*, vol. 11, no. 2, p. 207, 2005.

[14] A. Ekbal and S. Bandyopadhyay, "Named entity recognition using support vector machine: A language independent approach," *International Journal of Electrical, Computer, and Systems Engineering*, vol. 4, no. 2, pp. 155–170, 2010.

[15] S. AbdelRahman, M. Elarnaoty, M. Magdy, and A. Fahmy, "Integrated machine learning techniques for arabic named entity recognition," *IJCSI*, vol. 7, pp. 27–36, 2010.

[16] D. E. Appelt and B. Onyshkevych, "The common pattern specification language," in *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998*, pp. 23–30, Association for Computational Linguistics, 1998.

[17] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan, "Systemt: an algebraic approach to declarative information extraction," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 128–137, Association for Computational Linguistics, 2010.

[18] M. Atzmueller, P. Kluegl, and F. Puppe, "Rule-based information extraction for structured data acquisition using textmarker," in *Proceedings of LWA*, Citeseer, 2008.

[19] J. Urbain, "User-driven relational models for entity-relation search and extraction," in *Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search*, p. 5, ACM, 2012.

[20] B. Hammo, H. Abu-Salem, and S. Lytinen, "Qarab: A question answering system to support the arabic language," in *Proceedings of the ACL-02 workshop on Computational approaches to semitic languages*, pp. 1–11, Association for Computational Linguistics, 2002.

[21] N. Y. Habash, "Introduction to arabic natural language processing," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–187, 2010.

[22] T. Buckwalter, "Buckwalter Arabic morphological analyzer version 1.0," tech. rep., LDC catalog number LDC2002L49, 2002.

[23] K. R. Beesley, "Finite-state morphological analysis and generation of arabic at xerox research: Status and plans in 2001," in *ACL Workshop on Arabic Language Processing: Status and Perspective*, vol. 1, pp. 1–8, 2001.

[24] F. Zaraket and J. Makhlouta, "Arabic morphological analyzer with agglutinative affix morphemes and fusional concatenation rules," in *Proceedings of COLING 2012: Demonstration Papers*, (Mumbai, India), pp. 517–526, December 2012.

[25] M. Sipser, *Introduction to the Theory of Computation*, vol. 2. Thomson Course Technology Boston, 2006.

[26] D. Nolan and D. T. Lang, "Javascript object notation," in *XML and Web Technologies for Data Sciences with R*, pp. 227–253, Springer, 2014.

[27] F. A. Zaraket and J. Makhlouta, "Arabic cross-document NLP for the hadith and biography literature," in *Florida Artificial Intelligence Research Society Conference (FLAIRS)*, (Marco Island, Florida), May 2012.

[28] F. A. Zaraket and J. Makhlouta, "Arabic temporal entity extraction using morphological analysis," *International Journal of Computational Linguistics and Applications (IJCLA)*, vol. 3, pp. 121–136, 2012.

[29] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller, "Introduction to wordnet: An on-line lexical database*," *International journal of lexicography*, vol. 3, no. 4, pp. 235–244, 1990.

[30] G. Miller, "Nouns in wordnet: a lexical inheritance system," *International journal of Lexicography*, vol. 3, no. 4, pp. 245–264, 1990.

[31] D. Gross and K. Miller, "Adjectives in wordnet," *International Journal of Lexicography*, vol. 3, no. 4, pp. 265–277, 1990.

[32] K. Dukes, E. Atwell, and N. Habash, “Supervised collaboration for syntactic annotation of quranic Arabic,” *Language Resources and Evaluation*, pp. 1–30, 2011.

[33] B. J. Dorr, R. J. Passonneau, D. Farwell, R. Green, N. Habash, S. Helmreich, E. Hovy, L. Levin, K. J. Miller, T. Mitamura, *et al.*, “Interlingual annotation of parallel text corpora: a new framework for annotation and evaluation,” *Natural Language Engineering*, vol. 16, no. 3, p. 197, 2010.

[34] H. A. Kholidy and N. Chatterjee, “Towards developing an Arabic word alignment annotation tool with some Arabic alignment guidelines,” in *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pp. 778–783, IEEE, 2010.

[35] S. Kulick, A. Bies, and M. Maamouri, “Consistent and flexible integration of morphological annotation in the arabic treebank,” *Language Resources and Evaluation (LREC)*, 2010.

[36] C. Müller and M. Strube, “Multi-level annotation of linguistic data with MMAX2,” *Corpus technology and language pedagogy: New resources, new tools, new methods*, vol. 3, pp. 197–214, 2006.

[37] P. Stenetorp, S. Pyysalo, G. Topic, T. Ohta, S. Ananiadou, and J. Tsujii, “Brat: a web-based tool for nlp-assisted text annotation,” *EACL 2012*, p. 102, 2012.

[38] T. Morton and J. LaCivita, “Wordfreak: an open tool for linguistic annotation,” in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Demonstrations-Volume 4*, pp. 17–18, Association for Computational Linguistics, 2003.

[39] K. Maeda and S. Strassel, “Annotation tools for large-scale corpus development: Using agtk at the linguistic data consortium,” in *Proceedings of the Fourth International Conference on Language Resources and Evaluation*, 2004.

[40] O. Smrz and P. Pajas, “Morphotrees of arabic and their annotation in the tred environment,” in *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pp. 38–41, 2004.

[41] M. Attia, M. Rashwan, and M. Al-Badrashiny, “Fassieh^{-}, a semi-automatic visual interactive tool for morphological, pos-tags, phonetic, and semantic annotation of Arabic text corpora,” *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 17, no. 5, pp. 916–925, 2009.

[42] M. Alrahabi, A. H. Ibrahim, and J.-P. Desclés, "Semantic annotation of reported information in Arabic," *FLAIRS 2006, Floride, 11-13 Mai*, 2006.

[43] L. M. B. Saleh and H. S. Al-Khalifa, "AraTation: an Arabic semantic annotation tool," in *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, pp. 447–451, ACM, 2009.

Appendices

Appendix A

In this appendix, we present a list of case studies. For each, we design an extraction model using MERF. We also provide a sample match on an Arabic text for each case.

A.1 Economic analysis

In this example, we will define a task that detects the economic information in an Arabic text. This information includes the variation in the price of different commodities such as gold, oil, dollar, etc.

First, we define the MBFs for variation and the names of the commodities as follows:

- CHANGE : stems include {إرتفع (increase),إنخفض (decrease),حافظ (maintain)}.

- COMM : stems include {ذهب (gold),دولار (dollar),نفط (oil),فضّة (silver), ...}.

The MRE defined to extract economic information is shown below. *CHANGE* tags words related to change in prices. *COMM* denotes commodity and it tags commodities. *econinfo* finds matches for CHANGE followed by COMM with tolerance up to two NONE tags.

```
econinfo : CHANGE NONE^2 COMM;
```

A sample match for the MRE is shown in Table A.1. The MBF matches are shown in the second row in the table. Then, econinfo matches are detected in rows three and four.

Table A.1: Economic Analysis Example

الذهب	سعر	وارتفع	مستوياته	على	الّبنانيّة	السوق	في	الدولار	حافظ
COMM	NONE	CHANGE	NONE	NONE	NONE	NONE	NONE	COMM	CHANGE
econinfo								econinfo	

A.2 Prayer times

Salat times refer to times when Muslims perform their prayers. The term is used to refer to the five daily prayers. In this example, we will design the task to detect the prayer times.

The MBFs DB, SR, NOON, AN, SS, and ISHA denote daybreak, sunrise, noon, afternoon, sunset, and isha, respectively. These MBFs are defined by the stems فجر, شروق , ظهر , عصر , مغرب or غروب , عشاء respectively.

The MRE to detect the prayer times is shown below. The MBF NUM denotes numbers and it detects the numbers. *time* is defined as two NUM delimited by a colon. *prayer* is a prayer key word followed by time. An optional NONE tag is added for error tolerance.

```
time : NUM NUM;
prayer : ( DB | SR | NOON | AN | SS | ISHA ) NONE? time;
```

Table A.2 shows a sample match for the prayer case. Same as the previous examples, we first find the MBF matches for the Arabic words. Then, we find the matches for the MRE taking into consideration the dependency of an MRE on another such as prayer on time.

Table A.2: Salat Example

١٧	:	٣٣	الساعة	المغرب	صلاة	...	٠٥	:	١٨	الفجر
NUM		NUM	NONE	SS	NONE		NUM		NUM	DB
time							time			
prayer							prayer			

A.3 Football results

In this example, we extract information from Arabic articles that reports the football results. We write an MRE to detect the teams playing against each other along with the result of the games. For that purpose, we need to identify the names of the teams. We also have to identify Arabic words that express win, loss, and tie results.

Based on the previous explanation of the information to be extracted, we define the MBFs:

- NUM : Tags digits.
- GOAL : Tags the stems هدف (one goal), هدفين (two goals), and أهداف (three or more goals).
- FC : Tags the names of the football clubs. For simplicity, we will assume that the names are of one word only. Some of the club names are برشلونة (Barcelona), أوساسونا (Osasuna), and تشلسي (Chelsea) ...
- WLT : Denotes “win, lose, or tie”. This MBF is defined by a disjunction fromula between the glosses win, tie, lose, victory, loss, seizure, and seize.

We define the MRE to detect the football results as shown below. We have two MREs for the team detection. In *caseone*, WLT is detected first followed by the names of the teams (FC). In *secondcase*, the first team name is detected, then WLT followed by the name of the second team. We insert NONE between the basic tags to enhance the detection and make it error tolerable. *result* is defined as caseone or casetwo followed by the *score*. The score can be detected as two consecutive NUM tags such as ٥-١. Another form for the score is an optional NUM followed by GOAL such as هدفين (two goals) or ثلاثة أهداف (three goals).

```
caseone : WLT NONE^2 FC NONE^5 FC;
casetwo : FC NONE^5 WLT NONE^5 FC;
score : ( NUM? GOAL ) | ( NUM NUM );
result : ( caseone | casetwo ) NONE^2 score;
```

Table A.3 shows an example match for the football case. The result match is a casetwo match followed by two consecutive NUM tokens.

A.4 Geographical information

In this example, we will extract geographic information from Arabic text. This information includes the borders of countries from different directions. Thus, we

Table A.3: Football Result Example

١	-	٥	أوساسونا	على	بفوزه	الإسباني	الدوري	لترتيب	صدارته	برشلونة	عزّز
NUM		NUM	FC	NONE	WLT	NONE	NONE	NONE	NONE	FC	NONE
score			casetwo								
result											

need to identify the names of countries and places, directions, and key words that detect a relation between the position and the place.

Based on studying the target information, we define the following MBFs:

- POSITION : This MBF detects key words for directions. It is defined by the stems شمال, جنوب, شرق, and غرب.

- SC : This MBF denotes sea and continent. It is defined by the stems بحر (sea) and قارة(continent).

- PNAME : This MBF denotes place name. It is defined by the names of the countries, continents, and seas. These can be detected using the abstract category feature *Name of Place*. For enhanced detection, we include words such as أبيض(white) to detect البحر الأبيض(Mediterranean Sea). This is achieved by an MRE of SC followed by PNAME which will be illustrated in the example later.

- KEY : This token detects words that relate a position to a place. It is defined by the stems حدّ(delimit), طل(be viewed), and قع(take place).

The MREs to extract the geographic information is shown below. *place* is defined by an optional *SC* followed by *PNAME*. *geoinfo*, denotes geographic information, detects a *KEY* tag followed by *caseone* or *casetwo*. *caseone* detects one or more *place* tags followed by one or more *POSITION* tags. As for *casetwo*, it detects the reverse of *caseone*. We always introduce NONE for error tolerance and undesired information.

```
place : SC? PNAME;
caseone : place+ NONE^4 POSITION+;
```

```
casetwo : POSITION+ NONE^4 place+;
geoinfo : KEY ( NONE^2 ( caseone | casetwo ) )+;
```

Table A.4 shows a sample match for the MREs introduced before. *place* has two matches where *SC* wasn't present in one of them. Also, we have two matches of *geoinfo*. One match contains a *caseone* match while the other contains a *casetwo* match.

Table A.4: Geographical Information Example

الأبيض	البحر	على	الغرب	من	وتطلّ	والشرق	الشمال	من	سوريا	تحدّها
PNAME	SC	NONE	POSITION	NONE	KEY	POSITION	POSITION	NONE	PNAME	KEY
place									place	
casetwo						caseone				
geoinfo						geoinfo				

A.5 Official notifications

In this example, we will extract information from official notifications. These notifications are related to property policy requests by people. Our MRE will extract the name of the requester, the property number, the section number, and the region in which the property is present.

Based on the target information, we will define the following MBFs:

- NAME : This MBF detects a proper name. It is defined by the abstract category "Name of Person".

- NUM : This MBF detects numbers written in digit format.

- PROP: This MBF name denotes property. It is defined by the stem عقار.

- SECS, NOUN : In order to detect the section number, we have to detect the key word stem قسم with POS NOUN. Thus, we need the tokens SECS and NOUN which define the stem and the POS tag respectively.

- REQS, VERBP : The following two MBFs define the stem طلب and POS tag VERB_PERFECT respectively. We need them to define the pattern required to detect the starting keyword of an official notification.

- REGION : This MBF includes a list of stems representing places in Lebanon. Sample entries are بيروت and مزرعة.

The MRE to detect the notification information is shown below. A *person* is a sequence of *NAME* tokens. An optional NONE can be present between the NAME tags. *req* is defined by the conjunction between the MBFs *REQS* and *VERB*. Similarly, *sec* is defined by the conjunction between *SECS* and *NOUN*. *property* and *section* are defined by *PROP* and sec MRE respectively followed by *NUM*. *propsec* is defined by *property* and optional *section* separated by optional two *NONE*. Similarly, *secprop* is defined by *section* followed by *property* with two optional *NONE*. A *notification* token is a sequence of *req person*, *propsec* or *secprop*, then *REGION*. The MBFs are separated by zero or more *NONE* tags.

```
req : REQS & VERBP;
sec : SECS & NOUN;
person : NAME ( NONE? NAME)+;
property : PROP NUM;
section : sec NUM;
propsec : property NONE^2 section?;
secprop : section NONE^2 property;
notification : req NONE? person NONE^8
        ( propsec | secprop ) NONE^3 REGION;
```

Table A.5 shows a sample match of the notification MREs explained above. The first row in the table presents the Arabic text. The second raw shows the MBF matches of each word. The words طلب and للقسم are tagged by the MREs *req* and *sec* respectively in row three. The rows four, five, and six show the other MRE matches detected.

Table A.5: Notification Example

المزرعة	منطقة	٤٣٩	العقار	من	٢٨	للقسم	تمليك	سند	البلوز	علي	منير	طلب
REGION	NOUN	NUM	PROP	NONE	NUM	SECS NOUN	NONE	NOUN	NOUN	NAME	NAME	REQS VERBP
						sec						req
		property			section					person		
		secprop										
notification												

A.6 Professions information

In this example, our target is to detect the names of people and professions along with workplace in an Arabic text. Thus, we define MBFs that detect the name of a person, professions, and workplaces. Below is a detailed list of the defined MBFs for this problem.

- NAME : Tags the possible names of persons (male and female). It is defined by the category "Name of Person".

- PROF : It denotes profession and it is defined by stems such as دكتور (doctor), أستاذ(teacher), نجّار(carpenter), قاضي(judge) ...

- WPLACE : This MBF denotes workplace and it is defined by the stems مستشفى(hospital), مركة(company), مدرسة(school), جامعة(university) ...

The MREs to detect the profession information is shown below. *person* is a sequence of NAME. *profinfo* detects the name of a person and his/her profession and it can be found in two forms. The first is PROF followed by person such as الدكتور فادي (Doctor Fadi). The other form is person followed by an optional NONE and PROF such as فادي دكتور (Fadi is a doctor). *profdetails* detects the profession type and workplace. It is defined by profinfo followed by four optional NONE, then WPLACE.

```
person : NAME (NONE? NAME)*;
profinfo : (PROF person) | (person NONE? PROF);
profdetails : profinfo NONE^4 WPLACE;
```

Table A.6 shows a sample match of the profession information MREs introduced above.

Table A.6: Professions Example

الأشبال	مدرسة	في	الرياضيات	مادّة	نادر	شادي	الأستاذ	يدرّس
NONE	WPLACE	NONE	NONE	NONE	NAME	NAME	PROF	NONE
					person			
					profinfo			
	profdetails							

A.7 Money

In this example, our aim is to detect Arabic phrases referring to money. Those phrases contain numbers expressing the amount of money along with the currency. For this purpose, we define the MBFs shown below.

- NUM : This MBF is defined by the category *number*. This category includes words expressing numbers along with digits. Some of the words under this category are ألف (thousand), خمسة (five), ١٦ (16) ...

- CURRENCY : This token includes all the possible currencies. Sample currencies are ليرة (Lebanese pound), دولار (dollar), ين (yen) ...

The MREs to detect money phrases is shown below. *number* MRE is a sequence of NUM with possible NONE in between them. *money* MRE is defined as number followed by CURRENCY.

```
number : NUM (NONE? NUM)*;
money : number CURRENCY;
```

Table A.7 shows a sample match of the money MREs introduced above.

Table A.7: Money Example

البقالة	محلّ	في	ليرة	ألف	وخمسون	وستّة	مئة	شادي	دفع
NONE	NONE	NONE	CURRENCY	NUM	NUM	NUM	NUM	NONE	NONE
				number					
			Money						

A.8 Crop information

In this example, our aim is to extract information related to crop amount and type from an Arabic text. To do this task, we define MBFs that detect numbers, units, and different types of crops. Those MBFs are explained in detail below.

- NUM : This MBF is defined by the category *number*.

- UNIT : This MBF contains the Arabic stems referring to units of mass. Some of those units are طن (tonne), كيلو (Kilo) ...

- CROP : This MBF contains the different types of crops. Thus, it is defined by stems such as حبوب (grains), ذرة (corn), تفّاح (apples) . . .

The MREs designed to detect the crop information is shown below. *number* is defined as a sequence of NUM same as in previous examples. *amount* is number followed by UNIT which indicates an amount of mass. *cropinfo* is defined as amount followed by optional three NONE, then CROP which indicates the type of the crop.

```
number : NUM (NONE? NUM)*;
amount : number UNIT;
cropinfo : amount NONE^3 CROP;
```

Table A.8 shows an example match of the crop information pattern explained above.

Table A.8: Crop Info Example

<table>
<tr><td>الحبوب</td><td>من</td><td>طن</td><td>مليون</td><td>١٩</td><td>أوكرانيا</td><td>صدّرت</td></tr>
<tr><td>CROP</td><td>NONE</td><td>UNIT</td><td>NUM</td><td>NUM</td><td>NONE</td><td>NONE</td></tr>
<tr><td></td><td></td><td></td><td colspan="2">number</td><td></td><td></td></tr>
<tr><td></td><td></td><td colspan="3">amount</td><td></td><td></td></tr>
<tr><td colspan="5">cropinfo</td><td></td><td></td></tr>
</table>