On Abstract Finite-State Morphology

Ajit Narayanan & Lama Hashem
Department of Computer Science
University of Exeter
Exeter EX4 4PT
UK

Abstract

Aspects of abstract finite-state morphology are introduced and demonstrated. The use of two-way finite automata for Arabic noun stem and verb root inflection leads to abstractions based on finite-state transition network topology as well as the form and content of network arcs. Nonconcatenative morphology is distinguished from concatenative morphology by its use of movement on the output tape rather than the input tape. The idea of specific automata for classes of inflection inheriting some or all of the nodes, arc form and arc content of the abstract automaton is also introduced. This can lead to novel linguistic generalities and applications, as well as advantages in terms of procedural efficiency and representation.

1 Introduction

Finite-state approaches to morphology provide ways of analyzing surface forms by appealing to the notion of a finite-state transducer which in turn mimics an ordered set of rewrite rules. Instead of intermediate forms being introduced (as would happen if rewrite rules are used (e.g. [Narayanan and Mehdi, 1991] for Arabic morphology)), the finite-state transducer works on two tapes (one representing lexical structure, the other the surface structure) and switches states if the symbols currently being scanned on the two tapes match the conditions of the state transition. Following the distinction expressed by Kay [1987], *two-level* morphology is a specialization of finite-state morphology in that intermediate forms are not required even in the grammatical formalism (e.g. [Koskenniemi, 1983; Koskenniemi, 1984]). The only representations required are those for the lexical and surface forms, together with ways of mapping between the one and the other directly. Surface forms express the result of any spelling-change interactions between dictionary/lexicon primitives. A typical architecture of a two-level morphological system [Karttunen, 1983; Kataja and Koskenniemi, 1988] consists of a dictionary/lexicon component containing roots, stems, affixes and their co-occurrence restrictions, and an automaton component which codes for the mappings between dictionary/lexicon forms and surface realizations.

One of the problems faced by two-level approaches was their handling of *nonconcatenative* morphology. The main difference between Semitic and non-Semitic languages is that inflectional patterns are not straightforwardly concatenative (where morphemes are simply concatenated with roots, stems and each other) but 'interdigitate' or 'intercalate', i.e. the affix pattern is distributed among the constituents of the root morpheme. For example, the Arabic root 'd_r_s' ('study') intercalates with the inflectional pattern '_u_i_' (perfect passive) to form the stem 'duris' ('was studied'), which in turn can be inflected to signify number and gender[1]. This nonconcatenative aspect of Arabic can be problematic for a traditional two-level approach which bypasses intermediate forms. The problem concerns the way roots, stems (roots for Arabic verbs, stems for Arabic nouns) and inflection patterns are represented and stored. It is obviously not practical to store all the possible inflected forms

[1]Modern written Arabic rarely marks the vowels (short vowels are marked by diacritics), in this case the 'u' and 'i' in 'duris', except in beginners' books on Arabic. The (text) realization has the form 'drs'.

of each root. Instead, roots are usually separated from inflections. Morphological analysis of a string then consists of identifying the root and following pointers to inflections which may themselves contain pointers to other inflections [Karttunen, 1983]. The nonconcatenative aspect of Arabic means that, when processing a 'word' from beginning to end, different constituents of different inflections are encounted during root and inflection identification. The traditional idea of identifying a root and then following a pointer to types of inflection depending on immediately contiguous constituents of the inflection cannot be adopted. This forced the ALPNET researchers, for example, to adopt a novel way of storing and identifying inflections [Beesley *et al.*, 1989; Beesley and Newton, 1989; Beesley, 1990]. In their system there are two types of lexicon: the *root* lexicon, and the *pattern* lexicon. The root lexicon stores (three-consonant) roots in the form 'X_Y_Z', and the pattern lexicon stores inflectional patterns in the form '_A_B_', where the underscores '_' are called *detours*. Starting with the pattern lexicon, the analysis routines recursively switch between the two types of lexicon whenever a detour character is found.

This interesting solution raises the question of what aspect of morphology detouring is meant to reflect or express. If detouring is based simply on implementation and efficiency criteria, it is open to the possible criticism that an alternative, efficient way of handling intercalation which expresses some linguistic generalities whilst being consistent with the two-level approach should be preferred. Also, it is not clear what the implications of detouring are for parallel evaluation. However, one possible advantage is that detouring forces inflectional patterns to be kept together in the dictionary, rather than splitting them up into even smaller fragments, as might be required by a simple two-level approach. For instance, without detouring, patterns of the form '_A_B_' may need to be split up into lexical entries first for the 'A' and then, at a different level, for 'B'. The fact that 'A' and 'B' together represent a certain class of morphological phenomena might be lost.

2 Representing intercalation

An alternative approach to nonconcatenative morphology consists of using the idea of *prosodic templates* [McCarthy, 1981], whereby the underlying patterns of vowels and consonants are described. For instance, Kay [1987] provides a *four-level* account of how the Arabic root 'ktb' ('write') is mapped onto the stem 'aktabib' (imperfective active form) by means of the template 'VCCVCVC' (where 'V' stands for vowel and 'C' for consonant) and eight transitions. The first tape contains the root, the second the template, the third the intercalative vowels (vocalism), and the fourth the surface form. State switches are determined by 'frames' of quadruples which specify what each tape symbol must be. There is an overhead attached to the formulation of individual templates and quadruples (which represent the mapping rules) for even a restricted set of lexical entries. More generally, there is nothing in the templates themselves which allows underlying patterns to emerge or be used. This has led to the examination of ways of making abstractions on and classifying templates. For instance, *inheritance* and default-based approaches, as used in artificial intelligence, can be adopted for template and lexical entry representation [DeSmedt, 1984], so that duplicate and redundant information can be deleted from individual entries if the path they are on already contains this information. Research has focused on unification-based formalisms for inheritance network representation (e.g.[Flickinger *et al.*, 1985; Shieber, 1986; Porter, 1987; Evans and Gazdar, 1990; Bird and Blackburn, 1990; Reinhard and Gibbon, 1991]).

The question arises as to whether it is possible to achieve the generalities obtainable through a prosodic template approach within a multi-level finite-state model. Briefly, we hypothesize, in addition to the lexical and surface levels, an ***abstract level of automaton representation*** at which classes of inflectional phenomena are given an abstract representation. These abstract automata are translated into two-level automata for specific morphological phenomena. Concatenative and nonconcatenative patterns of inflection are represented not via the dictionary but at an abstract automaton component level. Applications of abstract automata to Arabic noun stems and verb roots are described below.

3 Arabic noun structure

A noun stem in Arabic is inflected according to *Case Type* (nominative, accusative, genitive), *Number* (singular, dual, plural), *Gender* (feminine and masculine), and *Definite/Indefinite*. These mainly are suffixes added to the noun stem. The case endings determine the vowelisation of the end letter of the stem.

The Indefinite Noun Endings are:

Singular

Nominative: -/un/ ٌ (double damma) (e.g. waladon وَلَدٌ)

Accusative: -/an/ َ (fatha) (e.g. waladan وَلَداً)

Genitive: -/en/ ِ (kasra) (e.g. waladen وَلَدٍ)

Dual

Nominative: -/ani/ انِ (e.g. waladani وَلَدانِ)

Accusative: -/ayni/ ينِ (e.g. waladyni وَلَدينِ)

Genitive: as for accusative.

Plural

In Arabic there are three types of plural. These are the *Sound Masculine Plural (SMP)*, the *Sound Feminine Plural (SFP)*, and the *Broken Plural (BP)*. The SMP is for male human beings [2]. For example مهندس

[2]Exception: sana - year سنة which can take the SMP.

('engineer') becomes مهندسون or مهندسين depending on the case ending. The SFP is for female human beings, inanimates, and most foreign words that have been incorporated into the language. For example, عالمة ('scientist') becomes عالماتٌ or عالماتٍ, again depending on the case ending. Similarly, 'car' (an inanimate object) (سيارة) becomes سياراتٌ or سياراتٍ. The BP does not follow any regular pattern and is for nouns that do not fall into the above categories. But this is not necessarily the case. For example, إبن ('son' — male human) can be pluralised to أبناءٌ which is a broken plural.

The SMP Ending

Nominative: -/oon/ ون (e.g. muhamiyoon محاميون)
Accusative: -/yyn/ ين (e.g. muhamiyyn محاميين)
Genitive: as for the accusative

The SFP Ending

If there is the feminine ending of ة then it needs to be removed before adding the SFP ending.

Nominative: -/atun/ اتٌ (e.g. maktabatun مكتباتٌ)
Accusative: -/aten/ اتٍ (e.g. maktabaten مكتباتٍ
Genitive: as for the accusative

The definite noun endings are the same as for the indefinite noun, except that **al** (ال) is added to the beginning of the noun. When a noun is made definite, the nunation is lost, so any ending with double fatha, kasra, or damma would be reduced to a single fatha, kasra, or damma. For example, ولدٌ ('boy') becomes الولدُ ('the boy').

4 Network representation

The noun structure system to be described below produces surface forms of lexical representation and so is a generator of inflected nouns. Generation is achieved by the use of *finite-state transition networks* (FSTNs). FSTNs realize *finite-state tables* (FSTs) which can be used for providing the mappings between lexical and surface structure. For instance, consider the FST in Figure 1 and the associated transition network in Figure 2. According to the

		Input		
		h	a	!
States	1.	2	0	0
	2.	0	3	0
	3.	2	0	4
	4:	0	0	0

Figure 1: FST for a Laughing Machine

tabular representation, if we're in state 1 (first row) and an 'h' is the current input character found (first column), then we switch to state 2 and look at the next character. If we're in state 1 and an 'a' or '!' is found, then we switch to an error state (0). If we're in state 2 and an 'a' is found, we switch to state 3 and read the next character, otherwise we

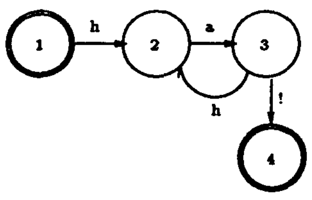

Figure 2: FSTN for the FST in Figure 1

switch to an error state. States 1, 2 and 3 are non-terminal (signified by the full-stops), whereas state 4 is terminal (signified by ':'). This FST specifies the state-switching behaviour of any machine which is to accept strings of the form '$\{ha\}^n$!', i.e. one or more occurrences of 'ha' followed by an exclamation mark. The same FST can be interpreted as a generator of such strings if 'Input' is changed to 'Output' in Figure 1. The 'conditions' on arcs are reinterpreted as characters to be output in this case.

The transition network in Figure 2 is constructed directly from the FST: nodes are labeled with state numbers, and arcs specify the input conditions before a state switch can occur. Double-circled nodes in the transition network signify start and terminal nodes. Given such FSTs and equivalent transition networks for Arabic noun and verb structures, Prolog was used to implement the automata. Start and end states are declared with the predicates **start_state(X)** and **end_state(Y)** where X and Y represent state numbers, and arc declarations have the form: **arc (CurrentState, NextState, [InputString], [OutputString])**. The third argument consists of the parameters **Input Character, Direction, Offset,** and the fourth refers (for nouns) to the characters for the output word. The direction indicates how to move the scanning head across the input. It can be one of two values: *r* for right, and *l* for left. The offset indicates by how much to move left or right along the input tape. (Right or left zero is the same as not moving.) The use of directions and offsets (a non-zero offset of *n* can be regarded as *n* separate state transitions of one move in the required direction) means that the automata used here are examples of *two-way finite automata* [Rabin and Scott, 1959; Sheperdson, 1959; Hopcroft and Ullman, 1979].

The system works in the following way for Singular Nominatives (and similarly for all the other noun inflections). A request for 'bnt' ('girl') to be inflected with Singular Nominative produces the list **[b,n,t,+,o,n]** which is then fed to the appropriate automaton. The FSTN for the Singular Nominative automaton can be seen in Figure 3 and its associated FST in Figure 4. The first character, 'b', is identified. The current arc statement is matched against

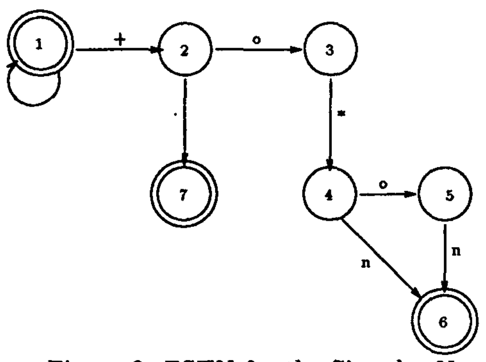
Figure 3: FSTN for the Singular Nominative

		Lexical level				
		*	+	.	o	n
	1:	1	2	0	0	0
	2.	0	0	7	3	0
	3.	4	0	0	0	0
States	4.	0	0	0	5	6
	5.	0	0	0	0	6
	6:	0	0	0	0	0
	7:	0	0	0	0	0

Figure 4: FST for the Singular Nominative

the arc facts of the automaton. For the first letter we have: **arc(1,?,[b,?,?],[?])**, i.e. what is the state to be moved to from state 1, and what is to be produced at this stage? This will match against the stored **arc(1,1,[Anychr,r,1],[Anychr])**, i.e. if in state 1 and any character found, then stay in state 1 and move one position to the right (offset) after copying the character ('b') to the output. The next character is then scanned. This matching process is repeated until the whole of the input word has been read.

Figure 5 shows how the output string is built up for input **[b,n,t,+,o,n]**. For the first four steps the procedure is straightforward: the input is echoed to the output list. The boundary sign (+) is replaced with a null value ("). When the first of the case ending letters is met, nothing is produced until a check is made whether the previous output character needs changing. The automaton therefore moves back to the end of the stem to check the end character (line 7). For this particular example, the character remains the same, and the automaton moves forward again to the first case ending (line 8). The offsets for movement backwards and forwards leaves the automaton at the same position as in line 6. The bottom line shows the output list at the end of the traversal of the automaton. (The 'O' in the output list refers to the double damma.) Null values are deleted, and the output list sent to the Arabic output routines. Narayanan and Hashem [1992] provide example runs and more detail about the implementation.

Input Character	Output List	Current State
b		1
b	[b]	1
b	[b,n]	1
t	[b,n,t]	1
+	[b,n,t,”]	2
o	[b,n,t,”,”]	3
t	[b,n,t,”,”]	4
o	[b,n,t,”,”,”]	5
n	[b,n,t,”,”,”,O]	6

Figure 5: Building The Output String

5 Inheritance-based derivation

Two-way automata for all nine types of inflection (three Case by three Number) can be constructed from abstract ones. For instance, the noun system used two abstractions on number. Figure 6 represents the abstract automaton form for all three cases (nominative, accusative and genitive) of singular and plural, and Figure 7 of dual.

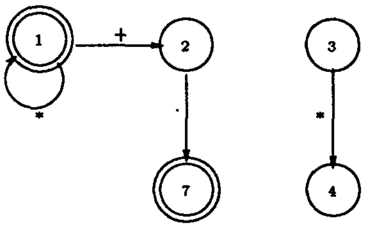
Figure 6: The abstract automaton for the Singular and Plural

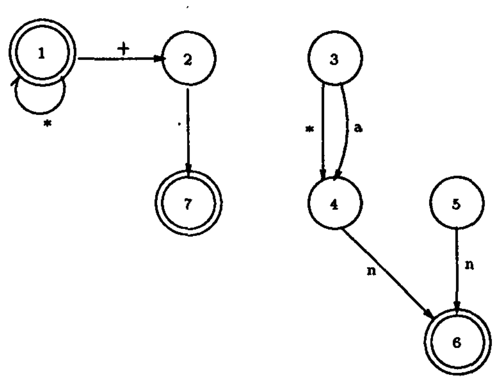
Figure 7: The abstract automaton for the Dual

Specific automata, for example for Dual Accusative and Genitive (Figure 8), can be derived from the abstract dual automaton by means of the specific automaton inheriting the basic form of the abstract automaton and adding specific arcs and nodes (specialization), as will be described later.

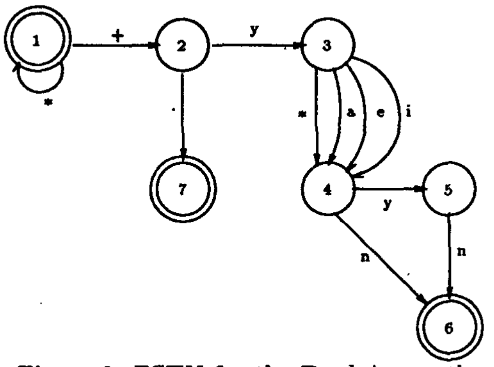

Figure 8: FSTN for the Dual Accusative/Genitive

6 Verb structure

The major difference between concatenative and nonconcatenative two-way automata for Arabic is that, for nonconcatenation, movement in both directions is required within the output tape rather than the input tape, so that affix information can be inserted between root characters. For concatenative two-way automata (as for the nouns), any moves are to the beginning or ending of the stem on the input tape, and if the last character of the stem needs changing this happens before the affix output is added.

Arabic verb structure is well-documented (e.g. [McCarthy, 1981; Hudson, 1986]). The following table gives the perfect active and perfect passive stems of the first three forms of 'ktb' only, but these are adequate to demonstrate the abstraction principles involved here.

Form	Active	Passive
I	katab	kutib
II	kattab	kuttib
III	kaatab	kuutib

The input representation is of the form [<root> + <vowels>], e.g. [k,t,b,+,a,a] with a request for Form II results in 'kattab', and [k,t,b,+,u,i] results in 'kuutib' if Form III passive is requested.

The following six statements describe an automaton (Figure 9) for generating Form I stems.

```
(1) arc(1,2,[C,r,1],[C_,r,0])
(2) arc(2,3,[C,r,1],[C_,r,1])
(3) arc(3,4,[C,r,1],[C,r,1])
(4) arc(4,5,[+,r,1],['',r,1])
(5) arc(5,6,[V,r,1],[[V,l,4],['',r,4]])
(6) arc(6,7,[V,r,1],[[V,l,2],['',r,2]])
```

The *output* argument of the arc statement is more complex than for nouns. The output argument [X, D, N] means 'After moving N steps in direction D, write X', where X can be a consonant C or vowel V. Also, the output argument can consist of one or two lists, the first for moving in one direction, the other to return the head to an appropriate location on the output tape for the next state. For instance, given the input [k,t,b,+,a,a] with a request for Form I, arc (1) would produce 'C_' (i.e. the first consonant is output together with a blank space to its right). The same would happen for the second consonant by arc (2). Arc (3) produces only a consonant, so in state 4 the output tape contains 'C_C_C', with the head of the output tape resting on the last C. Arc (4) acts as a check that exactly three consonants have been found. Arc (5) makes the output head move left four positions (to the first blank between two Cs) and inserts the V before moving back to its original position (and writing a null value again over the existing null value). Arc 6 works similarly, except that the offset is only two. The *input* has been scanned sequentially, one character at a time.

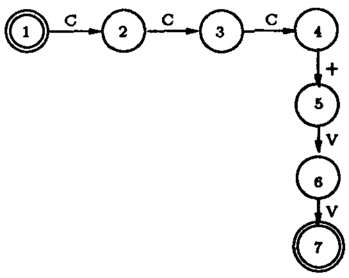

Figure 9: Automaton for Form I

This automaton also works for perfect passive Form I stems: 'a' and 'a' are replaced by 'u' and 'i'. Also, Form II can *inherit* the Form I automaton and add two specializations. First, arc (2) is changed so that instead of one C being written two copies of the C are made (i.e. (2a)), and arc (5) has offset 5 and not 4 (i.e. (5a)):

```
(2a) arc(2,3,[C,r,1],[CC_,r,1])
(5a) arc(5,6,[V,r,1],[[V,l,5],['',r,5]])
```

Form III can inherit from Form I and add its two specializations, namely, arc (1) is changed so that two blanks are introduced (i.e. (1b)), and arc (5) so that two Vs are written (i.e. (5b)). The offset when moving left is 5, and when returning 4.

```
(1b) arc(1,2,[C,r,1],[C__,r,0])
(5b) arc(5,6,[V,r,1],[[VV,l,5],['',r,4]])
```

7 Abstract automata and inheritance

The abstract automaton underlying Forms I, II and III is given in Figure 10. The solid lines specify those arcs which are *core* to all specific automata, and the dashed lines signify arcs which will be specialized. In

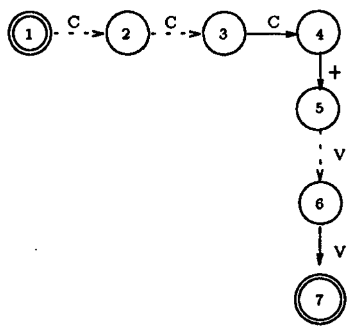

Figure 10: Abstract automaton for Forms I, II and III

the arcs of the automata for Forms I, II and III the pattern of output Cs and Vs has specialized (as in (1b), (2a) and (5b)) and so have offsets (as in 5(a) and 5(b)). Inheritance is multiple since the automaton for Form III inherits (2) from Form I as well as

1. the right return offset of 4 from (5) of Form I, i.e. arc(5,6, [V,r,1], [[V,l,4], ['', r,4]]), and
2. the move left (before writing) offset from (5a) of Form II, i.e. arc(5,6, [V,r,1], [[V, l,5], ['',r,5]]).

Form III also specializes its V pattern, i.e. arc(5, 6, [V,r,1], [[VV, l, 5], ['',r,4]]). In all cases, there are seven states and fixed length stems depending on their form. The inheritance structure for these three Forms is given in Figure 11. Form 0 specifies the core arcs which are inherited by all specific automata and cannot be specialized, and subsequent automata can further specialize their behaviour by adding their own arcs or changing contents of arcs inherited from other automata.

The inheritance status of an arc is given by another argument in the arc representation. Arcs therefore have the following form in the implemented system:

```
arc(S1,S2,IP,OP,status)
```

where S1 and S2 are state numbers, IP and OP are the sets of input and output parameters, respectively, and 'status' is 0 for core and non-zero for non-core. In the case of representing the inheritance relationships between the different Forms, any non-zero status value refers to the Form for which the arc is a specialization. The Form I automaton is therefore fully described by:

```
(1) arc(1,2,[C,r,1],[C_,r,0],1)
(2) arc(2,3,[C,r,1],[C_,r,1],1)
(3) arc(3,4,[C,r,1],[C,r,1],0)
(4) arc(4,5,[+,r,1],['',r,1],0)
(5) arc(5,6,[V,r,1],[[V,1,4],['',r,4]],1)
(6) arc(6,7,[V,r,1],[[V,1,2],['',r,2]] 0)
```

where status 1 refers to Form I specialization. Form II automata are fully described by:

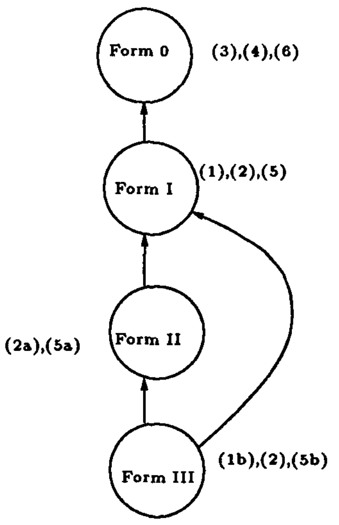

Figure 11: Inheritance structure for Forms I, II and III

```
(1) arc(1,2,[C,r,1],[C_,r,0],1)
(2a) arc(2,3,[C,r,1],[CC_,r,1],2)
(3) arc(3,4,[C,r,1],[C,r,1],0)
(4) arc(4,5,[+,r,1],['',r,1],0)
(5a) arc(5,6,[V,r,1],[[V,1,5],['',r,5]],2)
(6) arc(6,7,[V,r,1],[[V,1,2],['',r,2]] 0)
```

where status 2 refers to Form II specialization. Similarly for Form III:

```
(1b) arc(1,2,[C,r,1],[C__,r,0],3)
(2) arc(2,3,[C,r,1],[C_,r,1],1)
(3) arc(3,4,[C,r,1],[C,r,1],0)
(4) arc(4,5,[+,r,1],['',r,1],0)
(5b) arc(5,6,[V,r,1],[[VV,1,5],['',r,4]],3)
(6) arc(6,7,[V,r,1],[[V,1,2],['',r,2]],0)
```

where (5b) has been constructed out of (5) and 5(a), i.e. the state numbers, input argument and right return offset of 5, and the move left offset of 5, respectively. Ideally, these changes to (5) and (5a) will be carried out within the Form III object.

8 Discussion

The work reported here demonstrates the feasibility of adopting an abstract automaton, three-level approach to Arabic. Of particular importance is the distinction between abstract and particular FSA, where abstract automata represent classes of inflectional phenomena at an abstract level. They also represent algorithmic (processing) generalities. For instance, *crossing sequences*, i.e. movement across cells on the input (for nouns) and output (for verbs) tapes, cannot have repeated states with the head moving in the same direction (otherwise we may be in a loop). The first time movement left takes place, the state number must be odd (3 for nouns, 5 for verbs). Subsequent crossings must be in opposite directions.

The examples presented deal with significant fragments of Arabic, and potentially useful ways of representing Arabic verb Forms in inheritance networks have been identified. Other advantages to the three-level model involve the applicability of parallelism and the general way that the model is faithful to the two-level approach. There is a clear separation between the top level of abstract automata dealing with classes of inflection, on the one hand, and the knowledge expressed in the dictionary component, on the other. Also, the abstract automata express general inflectional processes: particular automata derived from these abstract automata handle individual inflectional variations.

Another advantage is that the three-level model may actually be intuitively more plausible as a general model of how native speakers acquire morphologically rich languages such as Arabic. The child may construct the abstract automata for classes of inflectional variations after exposure to individual words and sentences, and then use these abstract automata to make sense of the remaining inflectional variations not so far encountered. And with regard to the teaching of Arabic, the abstract automata may represent a teaching strategy whereby the overall structure of Arabic inflection types can be taught before specific ones are introduced.

There are implications for grammatical descriptions of inflectionally-rich languages. Most Arabic grammar books introduce inflectional variations in the form of complete tables which need to be memorized. Abstract automata may provide a more structured description of morphological phenomena. And finally, and perhaps most interestingly, the abstract level of automata description makes possible the comparison and contrasting of morphological phenomena across different but related morphologically rich languages. Analysis of inflections in different languages can be based on automata topology and arc form and content. This can lead to language-independent morphological theories of inflectional types. Research is continuing on all these aspects, as well as on relationships with structured Markov models [Kornai, 1991] and multi-tape autosegmental phonology [Wiebe, 1992].

References

[Beesley and Newton, 1989] K. Beesley and S. Newton. Computer analysis of Aymara morphology: A two-level, finite state approach. In S. Cox, editor, *Proceedings of the 15th Annual Deseret Language and Linguistics Symposium*, pages 126–144. Deseret Language and Linguistics Society, Brigham Young University, 1989.

[Beesley *et al.*, 1989] K. Beesley, T. Buckwalter, and S. Newton. Two-level finite state analysis of Arabic. In *Proceedings of the First Conference on Bilingual Computing in Arabic and English.* Literary and Linguistic Computing Centre, Cambridge University, 1989.

[Beesley, 1990] K. Beesley. Finite-state descriptions of Arabic morphology. In *Proceedings of the Second Conference on Bilingual Computing in Arabic and English.* Literary and Linguistic Computing Centre, Cambridge University, 1990.

[Bird and Blackburn, 1990] S. Bird and P. Blackburn. A logical approach to Arabic phonology. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 89–94, 1990.

[DeSmedt, 1984] W. M. DeSmedt. Using object-oriented knowledge representation techniques in morphology and syntax programming. In *Proceedings of the 1984 European Conference on Artificial Intelligence*, pages 181–184, 1984.

[Evans and Gazdar, 1990] R. Evans and G. Gazdar, editors. *The DATR Papers, Volume 1.* School of Cognitive and Computing Sciences, University of Sussex, 1990.

[Flickinger *et al.*, 1985] D. P. Flickinger, C. J. Pollard, and T. Wasow. Structure-sharing in lexical representation. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 262–267, 1985.

[Hopcroft and Ullman, 1979] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[Hudson, 1986] G. Hudson. Arabic root and pattern morphology without tiers. *Journal of Linguistics*, 22:85–122, 1986.

[Karttunen, 1983] L. Karttunen. KIMMO: A two-level morphological analyzer. *Texas Linguistic Forum*, 22:163–186, 1983.

[Kataja and Koskenniemi, 1988] L. Kataja and K. Koskenniemi. Finite-state description of Semitic morphology: a case study in Ancient Akkadian. In *Proceedings of the International Conference on Computational Linguistics (COLING88)*, pages 313–315, 1988.

[Kornai, 1991] A. Kornai. *Formal Phonology.* PhD thesis, Stanford University, 1991.

[Koskenniemi, 1983] K. Koskenniemi. Two-level model for morphological analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 683–685, 1983.

[Koskenniemi, 1984] K. Koskenniemi. A general computational model for word-form recognition and production. In *Proceedings of the International Conference on Computational Linguistics (COLING84)*, pages 178–181, 1984.

[McCarthy, 1981] J. J. McCarthy. A prosodic theory of nonconcatenative morphology. *Linguistic Inquiry*, 12:373–418, 1981.

[Narayanan and Mehdi, 1991] A. Narayanan and S. Mehdi. A computer model for transliterated Arabic. *Applied Computer Translation*, 1(3):5–28, 1991.

[Porter, 1987] H. H. Porter. Incorporating inheritance and feature structures into logic grammar formalism. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 228–234, 1987.

[Rabin and Scott, 1959] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.

[Reinhard and Gibbon, 1991] S. Reinhard and D. Gibbon. Prosodic inheritance and morphological generalisations. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 131–136, 1991.

[Sheperdson, 1959] J. C. Sheperdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

[Shieber, 1986] S. M. Shieber. *An Introduction to Unification-Based Approaches to Grammar*. CSLI, Stanford, 1986.

[Wiebe, 1992] B. Wiebe. *Modelling Autosegmental Phonology with Multi-Tape Finite State Transducers*. PhD thesis, Simon Fraser University, 1992.