Computational Linguistics Group
Department of Computer Science
University of Haifa

בלשנות חישובית
החוג למדעי המחשב
אוניברסיטת חיפה

Laboratory in Natural Language Processing

Shuly Wintner

Semester B, 2008: Wednesday, 10:00–13:00

`http://cs.haifa.ac.il/~shuly/teaching/08/lab/`

1 Objectives

The Lab offers a number of practical projects in Natural Language Processing (NLP), focusing on (but not limited to) processing of Hebrew. Some projects require previous knowledge of computational linguistics but some assume no previous background. All projects involve programming: the end result is a relatively large-scale, well-documented and efficient software package. Some of the projects may involve also some research (e.g., reading a research paper and implementing its ideas).

2 List of projects

2.1 Morphology-aware search engine for Hebrew

No background in NLP is required

Existing search engines are text-oriented: they index documents by *tokens*, which are usually defined as space-delimited strings. However, natural language texts are not just collections of tokens, since words in natural languages are formed through certain well-understood *morphological* processes, and are hence related to each other in ways which can contribute to better search results.

In this project you will augment a search engine with morphological capabilities: given query terms, you will generate all their possible inflections and submit those as a disjunctive query. In addition, you will use a bilingual dictionary to translate query terms from Hebrew to English and retrieve also relevant documents in English.

The extension will be implemented as a browser plug-in. Users will be able to enter search queries as usual; but those queries will be interpreted as lexemes (base forms). Query terms will

then be inflected. For example, if the user entered the query term מקלדת, the term will be expanded and the inflected forms מקלדות, המקלדת, המקלדות etc. will be generated. All those terms will be presented to the search engine as a disjunction. The user will also have the ability to specify that translated terms should be generated. In this case, the term *keyboard* will be generated and presented to the search engine.

The morphological generator (which produces all the inflected forms of a given lexeme) and the bilingual dictionary are given, although you may have to modify them slightly to adapt them for this task. Your main task, however, will be to implement a plug-in for the search engine that will interface with these existing resources to provide the additional capability and to evaluate the contribution of your work.

2.2 T9 for Hebrew

No background in NLP is required

T9 (Text on 9 keys) is a predictive text technology for mobile phones. T9's objective is to make it easier to type text messages on mobile devices with 9 keys. It allows words to be entered by a single keypress for each letter. It combines the groups of letters on each phone key with a fast-access dictionary of words, and looks up in the dictionary all words corresponding to the sequence of keypresses.

In this project you will design and implement a T9 algorithm for Hebrew. Your main task will be to create a dictionary of Hebrew word forms and experiment with various ways of compacting it. Several resources, including an existing list of inflected Hebrew words, will be provided to you, but you will have to adapt them to the task. Your solution should be easily adaptable: it should extend the dictionary as new word forms are entered, and update the frequencies of existing forms.

2.3 Collecting a bilingual corpus

No background in NLP is required

Text corpora are among the most important resource for a variety of NLP applications. They are used to provide word frequency counts for statistical NLP and information retrieval applications such as part-of-speech taggers, shallow parsers, categorization and summarization, to list just a few. Collecting corpora, representing and maintaining them are non-trivial tasks. The objective of this project is to build a parallel corpus of Hebrew and English documents by crawling the web. The documents in the corpus will then be sentence- and word-aligned.

You will develop software for collecting Hebrew-English corpora. The main technique is web-crawling: a program which crawls the web and searches for relevant documents. The main task is determining whether two documents are indeed possible translations, and you will be able to use some of the techniques reported in the literature (Resnik and Smith, 2003). Search will be limited to a number of dynamic web sites which are known to have similar documents in the two languages (e.g., some newspapers).

You will have to develop a storage solution for the collected corpora, such that new documents can be easily added at any time. In particular, the Hebrew documents you will collect will be mor-

phologically analyzed (using an existing system), and then stored in an SQL database of Hebrew texts, and will be accessible via an existing GUI for searching and viewing linguistic information.

2.4 Aligning a bilingual corpus

No background in NLP is required

Once a parallel corpus is available, it is useful to align it such that each sentence in L1 corresponds to zero or more sentences in L2 which represent its translation. In this project you will implement sentence- and word-level alignment algorithms (Gale and Church, 1993; Kay and Röscheisen, 1993) and apply them to the texts in the corpus. You will also compare their performance with that of an existing tool (GIZA++, Och and Ney (2000)). The expected outcome of the project is a tool for inducing a bilingual Hebrew-English dictionary.

2.5 Hebrew to English Named entity transliteration

No background in NLP is required

Transliteration is the process of replacing words and phrases in one language with their approximate spelling or phonetic equivalents in some other language. We distinguish between two types of transliteration:

Forward transliteration: When a Hebrew name is transliterated into English. For example, אריאל שרון is transliterated to *Ariel Sharon* and חיפה, ישראל to *Haifa, Israel.*

Backward transliteration: This is the reverse transliteration process where an English term which was transliterated to Hebrew has to be recovered. For example, ביל קלינטון to *Bill Clinton*, הוליווד to *Hollywood.*

When translating text from one language to another, proper names are sometimes translated, sometimes transliterated and sometimes a mixed approach is used. For example, when translating from Hebrew to English, names of people are always transliterated: אריאל שרון is transliterated to *Ariel Sharon* and ביל קלינטון to *Bill Clinton.* Other proper names, especially of organizations, are translated: הבית הלבן to *The White House*, האומות המאוחדות to *The United Nations.* Sometimes, however, proper names are partly translated and partly transliterated, as in הר חרמון *Mount Hermon* or מפרץ חיפה *Haifa bay*.

In this project you will adapt an algorithm that transliterates Arabic to English (Hermjakob, Knight, and Daumé III, 2008) to the special case of Hebrew to English transliteration.

2.6 Corpus-based clustering of Hebrew terms

No background in NLP is required

For many natural language application it is useful to know whether words are related to each other in various words. In particular, it is useful to be able to cluster words according to their semantic function. Natural clusters would be, for example, color terms; fruits; fruits and vegetables; edible items; etc.

Several techniques were suggested to automatically cluster words which occur in large corpora of texts. One of the most popular techniques is *latent semantic analysis* (Deerwester et al., 1990), which clusters words according to the context in which they occur.

In this project you will implement an LSA-based system for clustering Hebrew words. You will use available corpora of Hebrew provided by the Knowledge Center for Processing Hebrew. The corpora are morphologically analyzed and disambiguated; you will experiment with the effect of using the morphological information for this task. Specifically, you will have to develop an evaluation scheme for assessing the quality of your results.

A variant of this project will consist of a similar implementation and evaluation of a different algorithm, Chatterjee and Mohan (2008).

2.7 Morphological analysis of dotted Hebrew

Introduction to Computational Linguistics recommended but not required. As you will be revising an existing Java code, knowledge of Java is mandatory.

Morphological analysis is the process of determining the base (also known as *lexeme*, or *lemma*) of a word, along with its morphological attributes. An example of the morphological analysis of a simple Hebrew sentence is depicted in Figure 1.

Hebrew has a complex morphology and hence the design of a morphological analyzer for the language is a complex task. We currently have a large-scale and relatively accurate morphological system for Hebrew (Yona and Wintner, 2005) which works for *undotted* texts. In this project you will create a variant of the morphological system for the *dotted* script.

The main task here is to understand the morphological rules that apply to works, as stipulated for the undotted case, and then revise and refine them for the dotted case. The greatest benefit of such a system is that it will facilitate, in conjunction with a morphological disambiguation system which is currently under development, an automatic vocalization of undotted texts.

2.8 A compiler from XFST to LexTools

Introduction to Computational Linguistics is required

Finite-state technology is widely considered to be the appropriate means for describing the phonological and morphological phenomena of natural languages. Several finite-state ”toolboxes” exist which facilitate the stipulation of phonological and morphological rules by extending the language of regular expressions with additional operators. Such toolboxes typically include a language for extended regular expressions and a compiler from regular expressions to finite-state devices (automata and transducers). Unfortunately, there are no standards for the syntax of extended regular expression languages.

In this project you will design and implement a compiler which translates grammars expressed in XFST (Beesley and Karttunen, 2003) to grammars LexTools, a simple language built on top of the FSM finite-state toolbox (Mohri, Pereira, and Riley, 2000). You will be able to use a front-end compiler of XFST (Cohen-Sygal and Wintner, 2005), but the back-end, generating the LexTools code, will have to be implemented from scratch.

```
הרכבת   [+noun][+id]18182[+undotted]הרכבה[+transliterated]hrkbh[+gender]+feminine
        [+number]+singular[+script]+formal[+construct]+true
הרכבת   [+verb][+id]19729[+undotted]הרכיב[+transliterated]hrkib[+root]רכב[+binyan]+Hif'il
        [+person/gender/number]+2p/M/Sg[+script]+formal[+tense]+past
הרכבת   [+verb][+id]19729[+undotted]הרכיב[+transliterated]hrkib[+root]רכב[+binyan]+Hif'il
        [+person/gender/number]+2p/F/Sg[+script]+formal[+tense]+past
הרכבת   [+defArt]ה[+noun][+id]18975[+undotted]רכבת[+transliterated]rkbt[+gender]+feminine
        [+number]+singular[+script]+formal[+construct]+false

שבתה    [+noun][+id]17280[+undotted]שבת[+transliterated]ebt[+gender]+feminine
        [+number]+singular[+script]+formal[+construct]+false[+possessiveSuffix]+3p/F/Sg
שבתה    [+verb][+id]9430[+undotted]שבת[+transliterated]ebt[+root]שבת[+binyan]+Pa'al
        [+person/gender/number]+3p/F/Sg[+script]+formal[+tense]+past
שבתה    [+verb][+id]1541[+undotted]שבה[+transliterated]ebh[+root]שבה[+binyan]+Pa'al
        [+person/gender/number]+3p/F/Sg[+script]+formal[+tense]+past
שבתה    [+subord]ש[+preposition]ב[+noun][+id]19804[+undotted]תה[+transliterated]th
        [+gender]+masculine[+number]+singular[+script]+formal[+construct]+true
שבתה    [+subord]ש[+preposition]ב[+noun][+id]19804[+undotted]תה[+transliterated]th
        [+gender]+masculine[+number]+singular[+script]+formal[+construct]+false
שבתה    [+subord]ש[+preposition]ב[+defArt][+noun][+id]19804[+undotted]תה[+transliterated]th
        [+gender]+masculine[+number]+singular[+script]+formal[+construct]+false
שבתה    [+subord]ש[+noun][+id]19130[+undotted]בתה[+transliterated]bth[+gender]+feminine
        [+number]+singular[+script]+formal[+construct]+false
שבתה    [+subord]ש[+noun][+id]1379[+undotted]בת[+transliterated]bt[+gender]+feminine
        [+number]+singular[+script]+formal[+construct]+false[+possessiveSuffix]+3p/F/Sg

אתמול   [+adverb][+id]12448[+undotted]אתמול[+transliterated]atmwl
```

Figure 1: Example morphological analysis

The contribution of such a project lies in the fact that the Xerox utilities are proprietary; compilation to LexTools will enable us to use grammars developed with XFST on publicly available systems. Furthermore, parallel investigation of two similar, yet different, systems, is likely to result in new insights regarding the two systems and there interrelationships. Finally, such a compiler will enable us to compare the performance of the two systems on very similar benchmarks.

2.9 Implementation of registered FSAs

Introduction to Computational Linguistics is required. Due to the Unix-only availability of FSM, this project must be implemented in a Unix environment.

Finite-state registered automata (FSRA, Cohen-Sygal and Wintner (2006)) extend standard finite-state automata by adding very limited memory, in the form of a finite number of finitely-valued *registers*, to networks. Provably equivalent to finite-state automata, FSRA have been shown to be useful for naturally implementing several non-concatenative phenomena which are observed in natural languages.

In this project you will implement a package which supports FSRA. This will consist in two main phases:

- Extending the regular expression language of XFST by adding dedicated operators for FSRA. You will be able to use a front-end compiler of XFST (Cohen-Sygal and Wintner, 2005), and will have to extend it to support also the operators introduced by Cohen-Sygal and Wintner (2006).

- Compiling extended regular expressions to FSM. Extending and modifying the back-end of the XFST compiler (Cohen-Sygal and Wintner, 2005), you will support register operations by compiling extended regular expressions directly to FSM (Mohri, Pereira, and Riley, 2000), a finite-state low-level toolbox.

3 Administration

Projects are to be implemented by groups of at most two students. All systems will be presented at the end of the semester for a final demo. A coordination meeting is planned for Wednesday, July 25th; all work must be completed by Wednesday, September 24th.

The programming language must be portable enough to be usable on a variety of platforms; Java is recommended, C++ or Perl will be tolerated, if you have a different language in mind discuss it with the instructor.

Grading will be based on comprehension of the problem, quality of the implementation and quality of the documentation. In particular, the final grade will be based on:

- Comprehension of the problem (and the accompanying paper, where applicable)

- Full implementation of a working solution

- Presentation of a final working system

- Comprehensive documentation

References

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite-State Morphology: Xerox Tools and Techniques*. CSLI, Stanford.

Chatterjee, Niladri and Shiwali Mohan. 2008. Discovering word senses from text using random indexing. In Alexander F. Gelbukh, editor, *CICLing*, volume 4919 of *Lecture Notes in Computer Science*, pages 299–310. Springer.

Cohen-Sygal, Yael and Shuly Wintner. 2005. XFST2FSA: Comparing two finite-state toolboxes. In *Proceedings of the ACL-2005 Workshop on Software*, Ann Arbor, MI, June.

Cohen-Sygal, Yael and Shuly Wintner. 2006. Finite-state registered automata for non-concatenative morphology. *Computational Linguistics*, 32(1):49–82, March.

Deerwester, Scott C., Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407.

Gale, William A. and Kenneth W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102.

Hermjakob, Ulf, Kevin Knight, and Hal Daumé III. 2008. Name translation in statistical machine translation: Learning when to transliterate. In *Conference of the Association for Computational Linguistics (ACL)*, Columbus, OH.

Kay, Martin and Martin Röscheisen. 1993. Text-translation alignment. *Computational Linguistics*, 19(1):121–142.

Mohri, Mehryar, Fernando Pereira, and Michael Riley. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32, January.

Och, F. J. and H. Ney. 2000. Improved statistical alignment models. In *Proceedings of ACL-2000*, pages 440–447, Hongkong, China, October.

Resnik, Philip and Noah A. Smith. 2003. The web as a parallel corpus. *Comput. Linguist.*, 29(3):349–380.

Yona, Shlomo and Shuly Wintner. 2005. A finite-state morphological grammar of Hebrew. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 9–16, Ann Arbor, Michigan, June. Association for Computational Linguistics.