Accumulation of Lexical Sets: Acquisition of Dictionary Resources and Production of New Lexical Sets

DOAN-NGUYEN Hai
GETA - CLIPS - IMAG
BP 53, 38041 Grenoble, France
Fax: (33) 4 76 51 44 05 - Tel: (33) 4 76 63 59 76 - E-mail: Hai.Doan-Nguyen@imag.fr

Abstract

This paper presents our work on accumulation of lexical sets which includes acquisition of dictionary resources and production of new lexical sets from this. The method for the acquisition, using a context-free syntax-directed translator and text modification techniques, proves easy-to-use, flexible, and efficient. Categories of production are analyzed, and basic operations are proposed which make up a formalism for specifying and doing production. About 1.7 million lexical units were acquired and produced from dictionaries of various types and complexities. The paper also proposes a combinatorial and dynamic organization for lexical systems, which is based on the notion of virtual accumulation and the abstraction levels of lexical sets.

Keywords: dictionary resources, lexical acquisition, lexical production, lexical accumulation, computational lexicography.

Introduction

Acquisition and exploitation of dictionary resources (DRs) (machine-readable, on-line dictionaries, computational lexicons, etc) have long been recognized as important and difficult problems. Although there was a lot of work on DR acquisition, such as Byrd & al (1987), Neff & Boguraev (1989), Bläsi & Koch (1992), etc, it is still desirable to develop general, powerful, and easy-to-use methods and tools for this. Production of new dictionaries, even only crude drafts, from available ones, has been much less treated, and it seems that no general computational framework has been proposed (see eg, Byrd & al (1987), Tanaka & Umemura (1994), Dorr & al (1995)).

This paper deals with two problems: acquiring textual DRs by converting them into structured forms, and producing new lexical sets from those acquired. These two can be considered as two main activities of a more general notion: the *accumulation* of lexical sets. The term *"lexical set"* (LS) is used here to be a generic term for more specific ones such as "lexicon", "dictionary", and "lexical database".

Lexical data accumulated will be represented as objects of the Common Lisp Object System (CLOS) (Steel 1990). This object-oriented high-level programming environment facilitates any further manipulations on them, such as presentation (eg in formatted text), exchange (eg in SGML), database access, and production of new lexical structures, etc; the CLOS object form is thus a convenient pivot form for storing lexical units. This environment also helps us develop our methods and tools easily and efficiently.

In this paper, we will also discuss some other relevant issues: complexity measures for dictionaries, heuristic decisions in acquisition, the idea of virtual accumulation, abstraction levels on LSs, and a design for organization and exploitation of large lexical systems based on the notions of accumulation.

1 Acquisition

Our method combines the use of a context-free syntax-directed translator and text modification techniques.

1.1 A syntax-directed translator for acquisition

Transforming a DR into a structured form comprises parsing the source text and building the output structures. Our approach is different from those of other tools specialized for DR acquisition, eg Neff & Boguraev (1989) and Bläsi & Koch (1992), in that it does not impose beforehand a default output construction mechanism, but rather lets the user build the output as he wants. This means the output structures are not to be bound tightly to the parsing grammar. Particularly, they can be different from the logic structure of the source, as it is sometimes needed in acquisition. The user can also keep any presentation information (eg typographic codes) as needed; our approach is thus between the two extremes in acquisition approaches: one is keeping all presentation information, and one is transferring it all into structural representation.

Our tool consists of a syntax-directed translation (SDT) formalism called *h-grammar*, and its running engine. For a given dictionary, one writes an h-grammar describing the text of

its entry and the construction of the output. An h-grammar is a context-free grammar augmented with variables and actions. Its rules are of the form:

```
A(ai1 ai2 ...; ao1 ao2 ...) ->
    B(bi1 bi2 ...; bo1 bo2 ...)
    C(ci1 ci2 ...; co1 co2 ...) ... .
```

A is a nonterminal; B, C, ... may be a nonterminal, a terminal, the null symbol §, or an action. ai1, ai2, ... are *input variables*, which will be initialized when the rule is called. ao1, ao2, ..., bo1, bo2, ..., co1, co2, ... are *output variables*. bi1, bi2, ..., ci1, ci2, ... are *input expressions* (in LISP syntax), which may contain variables. When an item in the right-hand side of the rule is expanded, its input expressions are first computed. If the item is a nonterminal, a rule having it as the left-hand side is chosen to expand. If it is a terminal, a corresponding token is looked for in the parsed buffer and returned as the value of its (unique) output variable. If it is an action which is in fact a LISP function, the function is applied to the values of its input expressions, and the result values are assigned to its output variables (here we use the multiple-value function model of CLOS). Finally, the values of the output variables of the left-hand side nonterminal (ao1, ao2, ...) are collected and returned as the result of its expanding.

With some predefined action functions, output structures can be constructed freely, easily, and flexibly. We usually choose to make them CLOS objects and store them in *LISPO* form. This is our text representation for CLOS objects, which helps to read, verify, correct, store and transfer the result easily. Finally, the running engine has several operational modes, which facilitate debugging the h-grammars and treating errors met in parsing.

1.2 Text modification in acquisition

In general, an analyzer, such as the h-grammar tool above, is sufficient for acquisition. However, in practice, some precedent modification on the source text may often simplify much the analyzing phase. In contrast with many other approaches, we recognize the usefulness of text modification, and apply it systematically in our work. Its use can be listed as follows:

(1) *Facilitating parsing*. By inserting some specific marks before and/or after some elements of the source, human work in grammar writing and machine work in parsing can be reduced significantly.

(2) *Obtaining the result immediately without parsing*. In some simple cases, using several replacement operations in a text editor, we could obtain easily the LISPO form of a DR. The LISPification well-known in a lot of acquisition work is another example.

(3) *Retaining necessary information and stripping unnecessary one*. In many cases, much of the typographic information in the source text is not needed for the parsing phase, and can be purged straightforwardly in an adequate text editor.

(4) *Pre-editing the source and post-editing the result*, eg to correct some simple but common type of errors in them.

It is preferable that text modification be carried out as automatically as possible. The main type of modification needed is replacement using a strong string pattern-matching (or precisely, regular expression) mechanism. The modification of a source may consist of many operations and they need to be tested several times; it is therefore advantageous to have some way to register the operations and to run them in batch on the source. An advanced word processor such as Microsoft Word™, version 6, seems capable of satisfying those demands.

For sources produced with formatting from a specific editing environment (eg Microsoft Word, HTML editors), making modification in the same or an equivalent environment may be very profitable, because we can exploit format-based operations (eg search/replace based on format) provided by the environment.

1.3 Some related issues

1.3.1 Complexity measures for dictionaries

Intuitively, the more information types a dictionary has, the more complex it is, and the harder to acquire it becomes. We propose here a measure for this. Briefly, the *structure complexity (SC)* of a dictionary is equal to the sum of the number of elementary information types and the number of set components in its entry structure. For example, an English-French dictionary whose entries consist of an English headword, a part-of-speech, and a set of French translations, will have a SC of (1+1+1)+1=4.

Based on this measure, some others can be defined, eg the *average SC*, which gives the average number of information types present in an entry of a dictionary (because not all entries have all components filled).

1.3.2 Heuristics in acquisition

Contrary to what one may often suppose, decisions made in analyzing a DR are not always totally sure, but sometimes only heuristic ones. For large texts which often contain many errors and ambiguities like DRs, precise analysis design may become too complicated, even impossible.

Imagine, eg, some pure text dictionary where the sense numbers of the entries are made from a number and a point, eg '1.', '2.'; and, moreover, such forms are believed not to occur in content strings without verification (eg, because the dictionary is so large). An assumption that such forms delimit the senses in an entry is very convenient in practice, but is just a heuristics.

1.4 Result and example

Our method and tool have helped us acquire about 30 dictionaries with a total of more than 1.5 million entries. The DRs are of various languages, types, domains, formats, quantity, clarity, and complexity. Some typical examples are given in the following table.

Dictionary Resource[1]	*SC*	*Number of entries*
DEC, vol. II (Mel'cuk & al 1988)	79	100
French Official Terms (Délégation générale à la langue française)	19	3,500
Free On-line Dictionary of Computing (D. Howe, http://wombat.doc.ic.ac.uk)	15	10,800
English-Esperanto (D. Richardson, Esperanto League for North America)	11	6,000
English-UNL (Universal Networking Language. The United Nations University)	6	220,000
I. Kind's BABEL - Glossary of Computer Oriented Abbrevations and Acronyms	6	3,400

We present briefly here the acquisition of a highly complex DR, the Microsoft Word source files of volume 2 of the *"Dictionnaire explicatif et combinatoire du français contemporain" (DEC)* (Mel'cuk & al 1988). Despite the numerous errors in the source, we were able to achieve a rather fine analysis level with a minimal manual cleaning of the source. For example, a lexical function expression such as

$$Adv_{(1)}(Real_1\ ^{II}F_6 + Real_2\ ^{II}F_6)$$

was analyzed into:

```
(COMP
 ("Adv" NIL (OPTIONAL 1) NIL NIL NIL)
 (PAREN (+ (COMP ("Real" NIL (1) 2 NIL NIL) ("F" 6))
           (COMP ("Real" NIL (2) 2 NIL NIL) ("F" 6)))))
```

Compared to the method of direct programming that we had used before on the same source, human work was reduced by half (1.5 vs 3 person-months), and the result was better (finer analysis and lower error rate).

[1] All these DRs were used only for my personal research on acquisition, conforming to their authors' permission notes.

2 Production

From available LSs it is interesting and possible to produce new ones, eg, one can invert a bilingual dictionary A-B to obtain a B-A dictionary, or chain two dictionaries A-B and B-C to make an A-B-C, or only A-C (A, B, C are three languages). The produced LSs surely need more correction but they can serve at least as somewhat prepared materials, eg, dictionary drafts. Acquisition and production make the notion of lexical accumulation complete: the former is to obtain lexical data of (almost) the same linguistic structure as the source, the latter is to create data of totally new linguistic structures.

Viewed as a computational linguistic problem, production has two aspects. The *linguistic aspect* consists in defining what to produce, ie the mapping from the source LSs to the target LSs. The quality of the result depends on the linguistic decisions. There were several experiments studying some specific issues, such as sense mapping or attribute transferring (Byrd & al (1987), Dorr & al (1995)). This aspect seems to pose many difficult lexicographic problems, and is not dealt with here.

The *computational aspect*, in which we are interested, is how to do production. To be general, production needs a Turing machine computational power. In this perspective, a framework which can help us specify easily a production process may be very desirable. To build such a framework, we will examine several common categories of production, point out basic operations often used in them, and finally, establish and implement a formalism for specifying and doing production.

2.1 Categories of production

Production can be done in one of two directions, or by combining both: "extraction" and "synthesis". Some common categories of production are listed below.

(1) *Selection* of a subset by some criteria, eg selection of all verbs from a dictionary.

(2) *Extraction* of a substructure, eg extracting a bilingual dictionary from a trilingual.

(3) *Inversion*, eg of an English-French dictionary to obtain a French-English one.

(4) *Regrouping* some elements to make a "bigger" structure, eg regrouping homograph entries into polysemous ones.

(5) *Chaining*, eg two bilingual dictionaries A-B and B-C to obtain a trilingual A-B-C.

(6) *Paralleling*, eg an English-French dictionary with another English-French, to make an English-[French(1), French(2)] (for comparison or enrichment, ...).

(7) *Starring* combination, eg of several bilingual dictionaries A-B, B-A, A-C, C-A, A-D, D-A, to make a multiligual one with A being the pivot language (B, C, D)-A-(B, C, D).

Numeric evaluations can be included in production, eg in paralleling several English-French dictionaries, one can introduce a fuzzy logic number showing how well a French word translates an English one: the more dictionaries the French word occurs in, the bigger the number becomes.

2.2 Implementation of production

Studying the algorithms for the categories above shows they may make use of many common basic operations. As an example, the operation

regroup *set* **by** *function1* **into** *function2*

partitions *set* into groups of elements having the same value of applying *function1*, and applies *function2* on each group to make a new element. It can be used to regroup homograph entries (ie those having the same headword forms) of a dictionary into polysemous ones, as follows:

regroup *dictionary* **by** *headword* **into** *polysem*

(*polysem* is some function combining the body of the homograph entries into a polysemous one.)

It can also be used in the inversion of an English-French dictionary *EF-dict* whose entries are of the structure <English-word, French-translations> (eg <love, {aimer, amour}>):

for-all *EF-entry* **in** *EF-dict* **do**
  split *EF-entry* **into** <French, English> pairs, eg split <love, {aimer, amour}> into {<aimer, love> <amour, love>}. Call the result *FE-pairs*.
regroup *FE-pairs* **by** *French* **into** *FE-entry*

(*FE-entry* is a function making French-English entries, eg making <aimer, {love, like}> from <aimer, like> and <aimer, love>.)

Our formalism for production was built with four groups of operations (see Doan-Nguyen (1996) for more details):

(1) *Low-level operations*: assignments, conditionals, and (rarely used) iterations.

(2) *Data manipulation functions*, eg string functions.

(3) *Set and first-order predicate calculus operations*, eg the **for-all** above.

(4) *Advanced operations*, which do complicated transformations on objects and sets, eg **regroup**, **split** above.

Finally, LSs were implemented as LISP lists for "small" sets, and CLOS object databases and LISPO sequential files for large ones.

2.3 Result and example

Within the framework presented above, about 10 dictionary drafts of about 200,000 entries were produced. As an example, an English-French-UNL[2] (EFU) dictionary draft was produced from an English-UNL (EU) dictionary, a French-English-Malay (FEM), and a French-English (FE). The FEM is extracted and inverted to give an English-French dictionary (EF-1), the FE is inverted to give another (EF-2). The EFU is produced then by paralleling the EU, EF-1, and EF-2. This draft was used as the base for compiling a French-UNL dictionary at GETA (Boitet & al 1998). We have not yet had an evaluation on the draft.

3 Virtual Accumulation and Abstraction of Lexical Sets

3.1 Virtual accumulation

Accumulation discussed so far is *real accumulation*: the LS acquired or produced is available in its whole and its elements are put in a "standard" form used by the lexical system. However, accumulation may also be *virtual*, ie LSs which are not entirely available may still be used and even integrated in a lexical system, and lexical units may rest in their original format and will be converted to the standard form only when necessary. This means, eg, one can include in his lexical system another's Web online dictionary which only supplies an entry to each request.

Particularly, in *virtual acquisition*, the resource is untouched, but equipped with an acquisition operation, which will provide the necessary lexical units in the standard form when it is called. In *virtual production*, not the whole new LS is to be produced, but only the required unit(s). One can, eg, supply dynamically German equivalents of an English word by calling a function looking up English-French and French-German entries (in corresponding dictionaries) and then chaining them. Virtual production may not be suitable, however, for some production categories such as inversion.

3.2 Abstraction of LSs

The framework of accumulation, real and virtual, presented so far allows us to design a very general and dynamic model for lexical systems. The model is based on some abstraction levels of LSs as follows.

(1) A *physical support* is a disk file, database, Web page, etc. This is the most elementary level.

[2] UNL: Universal Networking Language (UNL 1996).

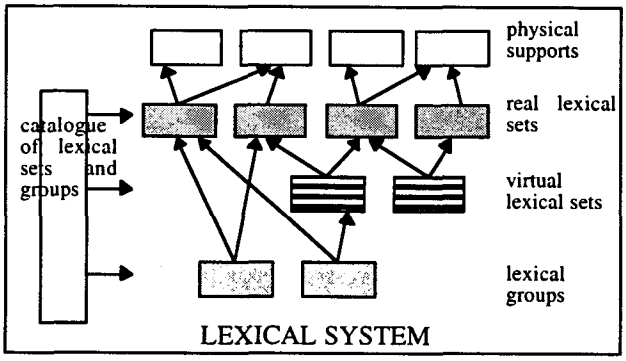
(2) A *LS support* makes up the contents of a LS. It comprises a set of physical supports (as a long dictionary may be divided into several files), and a number of access ways which determine how to access the data in the physical supports (as a database may have several index). The data in its physical supports may not be in the standard form; in this case it will be equipped with a standardizing function on accessed data.

(3) A *lexical set (LS)* comprises a set of LS supports. Although having the same contents, they may be different in physical form and data format; hence this opens the possibility to query a LS from different supports.

Virtual LSs are "sets" that do not have "real" supports, their entries are produced from some available sets when required, and there are no insert, delete activities for them.

(4) A *lexical group* comprises a number of LSs (real or virtual) that a user uses in a work, and a set of operations which he may need to do on them. A lexical group is thus a workstation in a lexical system, and this notion helps to view and develop the system modularly, combinatorially, and dynamically.

Based on these abstractions, a design on the organization for lexical systems can be proposed. Fundamentally, a lexical system has real LSs as basic elements. Its performance is augmented with the use of virtual LSs and lexical groups. A catalogue is used to register and manage the LSs and groups. A model of such an organization is shown in the figure below.



Conclusion and perspective

Although we have not yet been able to evaluate all the lexical data accumulated, our methods and tools for acquisition and production have shown themselves useful and efficient. We have also developed a rather complete notion of lexical data accumulation, which can be summarized as:

ACCUMULATION = (REAL + VIRTUAL) (ACQUISITION + PRODUCTION)

For the future, we would like to work on methods and environments for testing accumulated lexical data, for combining them with data derived from corpus-based methods, etc. Some more time and work will be needed to verify the usefulness and practicality of our lexical system design, of which the essential idea is the combinatorial and dynamic elaboration of lexical groups and virtual LSs. An experiment for this may be, eg, to build a dictionary server using Internet online dictionaries as resources.

Acknowledgement

The author is grateful to the French Government for her scholarship, to Christian Boitet and Gilles Sérasset for the suggestion of the theme and their help, and to the authors of the DRs for their kind permission of use.

References

Bläsi C. & Koch H. (1992), *Dictionary Entry Parsing Using Standard Methods*. COMPLEX '92, Budapest, pp. 61-70.

Boitet C. & al (1998), *Processing of French in the UNL Project (Year 1)*. Final Report, The United Nations University and L'Univeristé J. Fourrier, Grenoble, 216 p.

Byrd R. & al (1987), *Tools and Methods for Computational Lexicology*. Computational Linguistics, Vol 13, N° 3-4, pp. 219-240.

Doan-Nguyen H. (1996), *Transformations in Dictionary Resources Accumulation - Towards a Generic Approach*. COMPLEX '96, Budapest, pp. 29-38.

Dorr B. & al (1995), *From Syntactic Encodings to Thematic Roles: Building Lexical Entries for Interlingual MT*. Machine Translation 9, pp. 221-250.

Mel'cuk I. & al (1988), *Dictionnaire explicatif et combinatoire du français contemporain*. Volume II. Les Presses de l'Université de Montréal, 332 p.

Neff M. & Boguraev B. (1989), *Dictionaries, Dictionary Grammars and Dictionary Entry Parsing*. 27th Annual Meeting of the ACL, Vancouver, pp. 91-101.

Steele G. (1990). *Common Lisp - The Language*. Second Edition. Digital Press, 1030 p.

Tanaka K. & Umemura K. (1994), *Construction of a Bilingual Dictionary Intermediated by a Third Language*. COLING '94, Kyoto, pp. 297-303.

UNL (1996). *UNL - Universal Networking Language*. The United Nations University, 74 p.