Industry Canada
Centre for Information Technology Innovation (CITI)

TransSearch: A Bilingual Concordance Tool

Michel Simard simard@citi.doc.CA
George F. Foster foster@citi.doc.CA
François Perrault perrault@citi.doc.CA

Laval
October 1993

This document reports on research carried out at the Centre for Information Technology Innovation (CITI). The views expressed are strictly those of the authors.

Également disponible en français, sous le titre : “TransSearch : un concordancier bilingue”

Catalogue no. Co28-1/130-1996E

ISBN no. 0-662-24440-0

TransSearch: A Bilingual Concordance Tool

Michel Simard simard@citi.doc.CA

George F. Foster foster@citi.doc.CA

François Perrault perrault@citi.doc.CA

Centre for Information Technologies Innovation
1575, Chomedey blvd.
Laval, Québec
CANADA H7V 2X2

Abstract

TransSearch is a system for building and exploiting a *translation memory*, i.e. a textual database consisting of pairs of documents that are mutual translations. Each such pair of documents, when inserted into the database, is submitted to an *alignment* process, which makes explicit the relations that exist between the sentences of the two texts. A user may then search for ready-made solutions to specific translation problems, using a graphical query language.

1 Introduction

Computer automation is gradually coming into its own in the growing translation industry. Not, as one might have hoped, in the sense of automating the translation process itself, but rather through custom environments and tools to assist human translators. The earliest steps in this direction involved peripheral tasks like word processing, spell checking, access to on-line lexicons and terminology banks, storage and transfer of electronic documents, etc. CITI's *PTT* is a classic example of this type of environment. Recently however, an increasing number of systems have begun to focus more directly on the problem of translation itself. The best known is undoubtedly ALPNET's (formerly ALPS) *TSS*, but many others have since emerged; examples are products like Trados' *Translator's Workbench* and IBM's *Translation Manager*[1]. All of these systems share a common element: they are based on the concept of a *translation memory*.

It is apparent that existing translations contain more solutions to more translation problems than any other currently available resource. This is the main reason why all major translation services routi-

1. *Trados Translator's Workbench* is a product of Trados GmbH, Rotebühlstraße 87, 7000 Stuttgart 1, Germany; *Translation Manager* is an IBM product; We do not know whether ALPNET still distribute *TSS*.

nely archive their production. It is not uncommon for clients to request translations of documents which are virtually identical to previously translated texts, or which contain many duplicate passages. Faced with translating such a document, a translator with access to these related texts and their translations could use them as terminological references, an activity which would sometimes extend to the verbatim incorporation of whole passages into the new document.

The problem, of course, is how to provide efficient access to existing translations. Given well-organized archives, even paper ones, it should not prove too difficult to locate some previous version of a document to be translated. Finding *related* texts is a different story. To the extent that suitable electronic archives are available, conventional document-retrieval systems can provide a partial solution, but it is still up to the translator to pair the retrieved documents with their translations, and to identify the bilingual correspondences of interest within each such pair.

The concept of translation memory offers a solution to this problem. The basic idea is simple: archive existing translations in such a way as to facilitate their re-use. Rather than storing original documents separately from their translations, maintain electronic copies of document *pairs* in which the connecting translations are made explicit. This is how a system like Trados' *Translator's Workbench*, given a new sentence to translate and having found that sentence or a similar one in its memory, can quickly retrieve a candidate translation for the translator, who may then retain it or modify it as he sees fit (cf [2]).

Moreover, a translation memory has applications beyond the simple recovery of translated sentences: it can serve as the basis for a *bilingual concordance tool*, a program which finds occurrences of specified expressions or pairs of expressions, and displays them in their bilingual context. It is very likely that such a concordance would be a valuable resource for lexicographers (cf [3]) and translators alike. When confronted with an idiomatic expression like *out to lunch* or a fixed formula like *pitch a no-hitter*, a translator will frequently hesitate over an appropriate translation; traditional resources, such as bilingual dictionaries, are often relatively impoverished in this regard. A bilingual concordance tool permits the extraction from a translation memory of such expressions, along with different translations that have been used in the past. This idea is developed in greater detail by Macklovitch [10].

In what follows we describe a bilingual concordance tool called *TransSearch* that has been conceived and developed by the machine assisted translation (TAO) group at CITI. This program is the first in a set of tools intended as aids to the professional translator (see [7]).

2 Bilingual Concordances

Before tackling the subject of bilingual concordances, we need to briefly discuss ''ordinary'' (ie, unilingual) concordances. In the traditional sense of the term, a concordance is an ordered list of the

occurrences of all words in a text, each occurrence being presented within a small portion of surrounding text, typically at its centre. It is customary to call this surrounding text the *context* of the occurrence (whence the name *keywords in context*, or KWIC, for a concordance of this type). Given a minimal context, say five words on either side of each occurrence, the complete concordance will be eleven times larger than the original text. For a corpus of any significant size[2], and for contexts other than very minimal ones, such a quantity of text becomes extremely difficult to manage.

This motivates the idea of a *concordance program* which can dynamically produce subsets of an implicit ‘‘full’’ concordance for a text. Numerous programs of this type exist; examples are *lq-text*, *PAT*, and *gptx*[3].

Besides avoiding the problems associated with processing large quantities of text, such programs permit increased flexibility in the amount of context displayed with each occurrence as well as in the selection of the concordance subsets to be produced. The latter is usually accomplished by means of a special query language in which words and expressions whose contexts are of interest may be described. Such a language is normally optimized for ‘‘linguistic’’, rather than ‘‘document-oriented’’ queries. For example, when a user specifies a sequence of words, their order is generally considered significant (*home phone*, and not *phone home*). Even when this property does not hold, it is usually the case that the words which satisfy a query must appear within the same ‘‘linguistic context’’, eg, within the same sentence.

Concordance programs are particularly useful when queries can be carried out in real time. To achieve real-time performance on arbitrarily large corpora, it is necessary to rely on a specially structured representation of text. Whatever the exact nature of this representation, it must include some form of text *index* which permits all occurrences of a given token to be located in time proportional to the number of occurrences rather than to the size of the corpus.

A bilingual concordance can be defined along lines similar to a unilingual concordance: an ordered list of the occurrences of all words in a pair of mutually-translated texts, each of which is presented with its context plus its context’s translation. Needless to say, the problems associated with processing a unilingual concordance are doubled in the bilingual case: the utility of a program which can automatically produce subsets of a bilingual concordance should be obvious.

The implementation of a bilingual concordance program is more difficult than that of a unilingual concordance program. First there is the problem of ‘‘bilingual context’’. A unilingual program displays either a fixed number of characters or words on either side of a word occurrence, or a more ‘‘linguistic’’ context such as the sentence or paragraph to which the occurrence belongs. If the con-

---

2. For the majority of applications, it is not worthwhile to construct concordances from corpora of less than appreciable size.

3. *lq-text* (written by Liam Quin) and *gptx* (written by François Pinard) are public domain programs. *PAT* is a product of OpenText Corporation, Waterloo, Ontario.

text's translation is to be displayed as well, it must first be located. This is not always possible for an arbitrary portion of text; in general we must settle for an approximation, and in this regard it is preferable that the program over-estimate the size of the translation, provided that in so doing it does not swamp the user with unwanted information.

Furthermore, it can be shown that the time required to find the translation of a given context – even approximately – is proportional to the size of the two texts, so it is impossible to accomplish this operation in real time. It is therefore necessary to pre-calculate the correspondences between translated units and to store this information with the text in such a way as to permit rapid access. It is this data structure that gives rise to the notion of *translation memory*.

Finally, the query language of a bilingual concordance program must allow a user to take advantage of the bilingual structure of the text by offering, for example, the option of specifying pairs of bilingual expressions. This property makes it possible to examine the contexts in which some particular expression is translated by another.

In the following sections, we first present our model of a translation memory, then the mechanism which allows bilingual concordances to be extracted, and finally the user interface that gives access to this mechanism.

3 Translation Memory

We call our translation memory structure *TransBase*. A TransBase structure is specific to a particular pair of languages and consists of a set of documents, each of which comprises a pair of mutually-translated texts. When a document is added to the structure, it undergoes a *translation analysis* consisting of one bilingual and two unilingual phases. The results of the analysis are added to the TransBase data structure, which is made up of three main components corresponding to the three phases of analysis. The process is illustrated in Figure 1.

We begin by describing the translation analysis, then describe how the resulting data is organized in a TransBase structure.

3.1 Translation Analysis

The analysis of each document added to a TransBase structure proceeds in four distinct steps: *pre-processing*, *tokenization*, *segmentation*, and *alignment*. The first three steps together constitute a *unilingual* analysis which is applied separately to each of the unilingual halves of a document. The final step is a *bilingual* analysis which is applied to the results of the unilingual analyses in order to identify the translation relations that exist between any two mutually-translated texts.

The object of *pre-processing* is to normalize the contents of a text file in preparation for the next step (tokenization). Essentially, this involves re-coding the file using a standard character set (ISO-

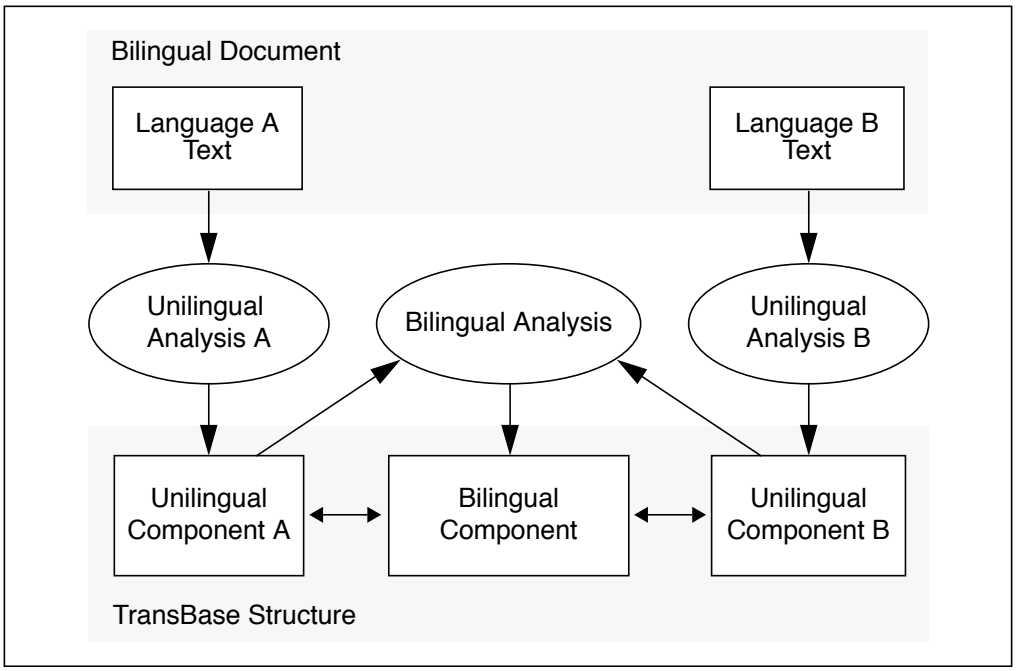

Figure 1: Insertion of a document into a TransBase structure

Latin 1), ensuring that the text adheres to normal typographic conventions for the language in which it is written, and eliminating markup codes not used by subsequent steps. Computer representation of text being what it is, each new text format encountered requires a different pre-processing procedure and hence the creation of a new pre-processing module.

Tokenization involves identifying and isolating words, punctuation, numeric and symbolic expressions, and markup from text. Apart from being a pre-requisite for later stages of analysis (segmentation and alignment), the information produced in this step is crucial to TransSearch's capacity to search for words rather than character strings. This is because the *index* component on which real-time searches are based is constructed from the words identified during tokenization. It is therefore advantageous that tokenization go beyond the simple process of separating character strings from unambiguous punctuation and whitespace, and attempt to separate ''aggregate'' words into their components using language-specific rules. Thus *jusqu'alors* becomes *jusqu'* + *alors*, and *women's-rights* becomes *women* + *'s* + - + *rights* (here + indicates the places where the aggregate has been split). Another major tokenization task is that of resolving the ambiguity between periods which are used to mark the ends of sentences and dots which are used to mark the ends of abbreviations. This is accomplished by consulting a list of standard abbreviations and common abbreviation patterns. The final product of tokenization is an ordered list of tokens, tagged to indicate their types: word, punctuation, expression, etc.

The *segmentation* step divides a token sequence into sentences, paragraphs, and sections. For the most part, boundaries between these units will have been unambiguously identified by the prece-

ding steps: many text formats have clear conventions for marking paragraphs and sections, and these are suitably encoded by the pre-processor so as to be preserved during tokenization and segmentation. The real work performed by the segmentation step is finding sentence boundaries. This is usually straighforward because periods are identified during tokenization, but there are some tricky cases: sentences within quotes or parentheses; those which end with some mark other than a period (*? ! ...* etc); and those which end in abbreviations which finish with a dot.

Finally, *alignment* is the process of finding the relations between a source text and its translation. This consists in segmenting the two texts in tandem, so that the *n*th segment in one corresponds to the *n*th segment in its translation. Each pair of mutually-translated segments aligned in this way is called a *couple*. Independent of the text units (sentences, paragraphs, etc) on which the segmentation is based, it is desirable that it be *maximal*, ie that each segment contain the smallest possible number of units. Figure 2 illustrates an alignment for two paragraphs in which the segmentation units are sentences.

Couple	English Version	French Version
1	*The crisis our farmers are in right now will affect all of us at a certain point in time.*	*La crise que vivent en ce moment nos agriculteurs se répercutera sur tous et chacun de nous à un certain moment.*
2	*We are all consumers and we all need a strong and healthy agricultural sector.*	*Nous sommes des consommateurs.* *Nous avons tous besoin d'une agriculture saine et forte.*
3	*I am glad that the Hon. Member for Algoma (Mr. Foster) mentioned figures in his remarks.* *Otherwise , the Government might have eluded the problem once again .*	*Heureusement que le député d'Algoma (M. Foster) a mentionné des chiffres dans ses remarques, sans cela ce gouvernement s'en serait sorti en douce encore une fois.*
4	*The Hon. Member for Algoma suggested Tuesday night that the Government had to take a clear position and make a commitment to assist our farmers before it is too late.*	*Le député d'Algoma suggérait mardi soir qu'il fallait que le gouvernement se prononce clairement et s'engage à aider nos agriculteurs avant qu'il ne soit trop tard .*

Figure 2: Alignment of an English/French paragraph pair

The alignments in TransBase consist of sentence segments, and are produced by an algorithm based on that of Gale and Church [6]. For a given pair of texts, this algorithm considers all alignments which can be established using a fixed number of *translation patterns* for couples. These represent different ''strategies'' which a translator can use to translate text: translate one sentence by one sentence, split a source sentence into two translated sentences, merge two source sentences into a single translated sentence, etc. Each couple of each possible alignment is assigned a score which indicates how well its two segments match, and each alignment is scored as the product of the scores of its constituent couples. Dynamic programming is used to pick the alignment with the highest overall score in time which is quadratic in the number of sentences to be aligned.

Gale and Church's scoring function is based on a stochastic model which provides an estimate of the probability that the segments in a couple are mutual translations. It can be observed that the relation between character lengths of mutually-translated segments follows a normal distribution. The model combines this observation with an empirically-estimated a priori probability for each possible translation pattern (see table 1). Despite its apparent simplicity, this method produces excellent results when the translation does not deviate significantly from the original text.

Translation Pattern	***A priori Probability***
1-1	*0.89*
1-0 ou 0-1	*0.0099*
1-2 ou 2-1	*0.089*
2-2	*0.011*

Tableau 1: Estimated probabilities for translation patterns.
Source: [6].

Our alignment algorithm is essentially the same as that described above, except that it uses *cognate words* to make it more robust in difficult situations. Cognates are pairs of words in different languages which, usually due to a common etymology, share phonological or orthographic properties as well as semantic properties, so that they are often employed as mutual translations. Examples are *generation* in English and *génération* in French, or *reason* and *raison*. Such words abound in English and French, as they do in many other pairs of languages. We carried out a statistical analysis of translated texts and found that the distribution of cognates in a pair of translated segments is approximately binomial, $B(n,p)$, where n is the total number of words in the two segments and $p = .3$. On the other hand, the cognate distribution in pairs of *random* segments drawn from translated text is also approxmately binomial, but with $p = .1$. From this we can deduce that there are on average three times as many cognates in a correctly-aligned couple as there are in an incorrectly-aligned couple.

In difficult situations, it is often the case that Gale and Church's algorithm considers many alignments with similar scores. Instead of making a somewhat arbitrary choice in such cases, our algorithm retains the set of best-scoring alignments for a second pass. This begins with an estimation of the number of cognates in each couple of each alignment: we have found that a good approximation results from pairing words whose first four characters are identical. Next, a new score is assigned to each couple based on the number of cognates it contains, and the alignment which scores highest by this criterion is chosen.

When applied to the Hansard transcripts of Canadian Parliamentary Proceedings, our alignment algorithm achieved a success rate of over 98%. It is also quite efficient, and is capable of aligning more than 80,000 words in each language per minute on a SPARCstation 1+. A detailed description of the algorithm and an analysis of its performance are given in [11].

3.2 Structure

Conceptually, the TransBase structure can be seen as a database comprising six types of entities which can be organized into three components: a unilingual component for each language, made up of *Token* and *Form* entities; and a bilingual component, made up of *Document* and *Couple* entities. Figure 3 shows an exploded view of the TransBase structure as it appears in Figure 1, in the form of a simplified entity-relation diagram (where each box representing an entity contains a list of the attributes for that entity).

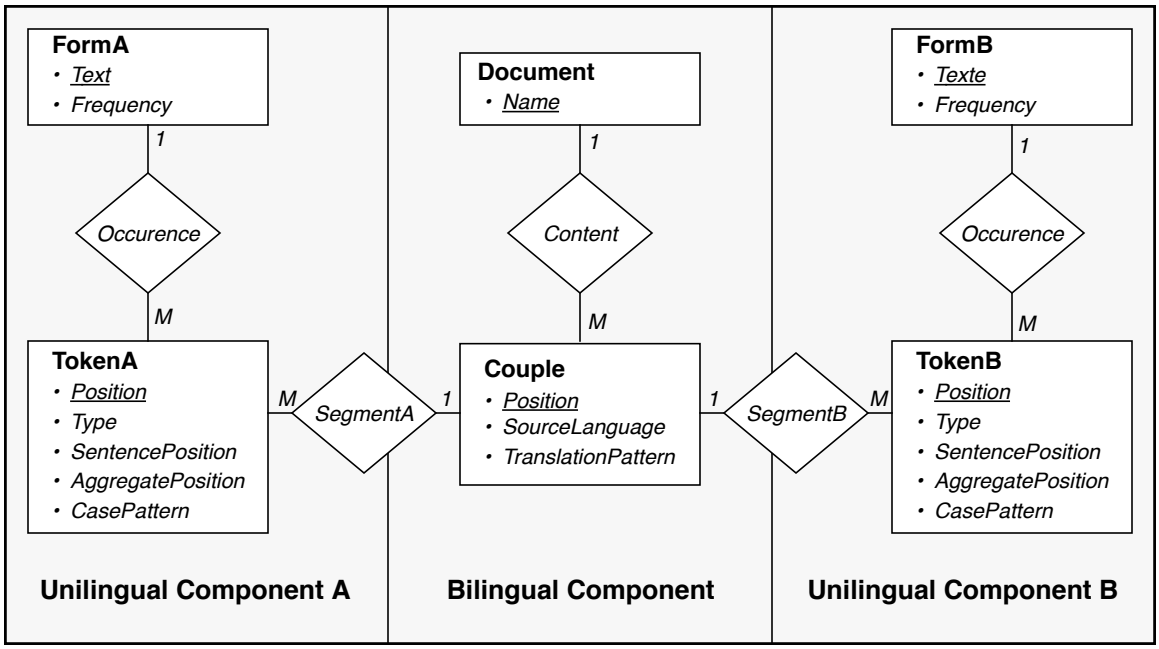

Figure 3: Entity-relation diagram of TransBase structure

The unilingual component A represents the set of all texts in language A (ie the language A versions of all documents) in the TransBase structure, and consists of all *TokenA* entities, all *FormA* entities, and the *Occurrence* relation that links them.

There is a *TokenA* entity for each token identified during the tokenization (see section 3.1) of any text in language A. The order in which tokens appear in text is preserved in the *Position* attribute, which takes an integer value that reflects the token's relative position: tokens with *Position* values *n*-1, *n*, and *n* +1 are consecutive. Each *TokenA* entity in the database has a unique *Position* value. The *Type*, *AggregatePosition*, and *SentencePosition* attributes record, respectively, the token's type (as determined during tokenization), its position within an aggregate (initial, middle, final, or not), and its position within the sentence (initial, middle, or final, as determined during segmentation). The *CasePattern* attribute is described below.

There is a *FormA* entity for each string of characters which occurs as a token in language A. The *Text* attribute contains the form's string, and is unique for each *FormA* entity in the database. The *Frequency* attribute gives the number of times the form has occurred as a token.

The *Occurrence* relation links *TokenA* and *FormA* entities: given a *TokenA* entity, it yields a single *FormA* entity containing the token's text; given a *FormA* entity, it yields a list of *TokenA* entities which are the form's occurrences. One additional subtlety is that, to save space and allow for efficient case-independent searches, forms are case-normalized; each token's *CasePattern* value specifies which of a number of simple transformations (make first letter upper case, make all letters upper case, etc) must be applied to re-create its original text from its form's text.

The unilingual component B plays the same role for language B as does component A for language A.

The bilingual component of the TransBase structure represents the connections between unilingual texts in the database; it consists of the *Document* and *Couple* entities and the *Content* relation that links them.

There is a *Document* entity for each document in the database. This currently has only the single attribute *Name*, but others would probably be needed for commercial applications: name(s) of authors, translators, editors, client, archive date, etc.

The *Couple* entity is the heart of the bilingual component; there is one for each pair of segments produced during alignment. The *Position* attribute plays the same role here as for the *Token* entity, that is, it serves both to indicate relative position within an alignment and as a unique identifier. The *TranslationPattern* attribute specifies the number of sentences to be found on each side of the couple, ie the ''strategy'' that the translator employed for this segment. The *SourceLanguage* attribute specifies which of the two segments (language A or language B) was the original and which the translation. One might think that the source language would be the same for all couples in the same document, but there are cases, such as the Hansard transcripts, where this does not hold. We can, however, safely assume that the source language does not change *within* a single couple, even if it contains more than one sentence in each language.

The *Content* relation links *Document* and *Couple* entities. Given a *Document* entity, it yields the set of couples which belong to that document; given a *Couple* entity, it yields the document to which it belongs.

The *SegmentA* and *SegmentB* relations provide the links between the bilingual and unilingual components of TransBase. Given a *Couple* entity, these relations furnish the sets of language A and language B tokens which it comprises; given a *TokenA* or *TokenB* entity, they indicate the couple to which it belongs.

The implementation of the TransBase structure is not of great interest here, but a few details are worth mentioning. The database is stored in a collection of files, one for each type of entity. These are deliberately redundant, so that the entire structure can be re-created from the files for the *TokenA* and *TokenB* entities if necessary. Access requires a minimum of memory (although large buffers can speed up certain operations), and any entity can be retrieved from its unique key in constant time (modulo disk random access time on systems where this is not constant in file size). One mode of access which is crucial to performance is the half of the *Occurrence* relation which maps a form's text to its list of token positions. By implementing this list as a B-tree (see [1]), we ensure rapid retrieval of the form's occurrences within any particular region of text.

4 The Search Mechanism

Our bilingual concordance program is called TransSearch. This program uses a fairly traditional search algorithm to extract concordance subsets from a TransBase structure. We first describe the query language by which subsets are specified, then the search algorithm itself.

4.1 Query Language

We have defined an abstract query language for TransSearch. The syntax of this language is described in table 2 in the form of a context-free grammar (terminal symbols are bolded, or within double quotes where confusion is possible; the vertical bar "**|**" indicates disjunction). Because this grammar is intended for queries on English and French texts, the symbol *language* produces the terminals **e** and **f** (*English* and *French*), and *character* generates only the characters which are permissible in these two languages. For another pair of langauges, it is clear that adjustments would be required.

query → language **(** expression **)**
query → negation language **(** expression **)**
query → query query.

expression → descriptor
expression → **(** expression **)**
expression → expression operator expression

operator → « » *("space")*
operator → **...**
operator → **..** integer **..**
operator → «**|**»

descriptor → form
descriptor → form expansion

form → character
form → character form

integer → digit
integer → digit integer

language → **e** | **f**

negation → **~**

expansion → **+**

character → **a** | **A** | **à** | **À** | ... | **z** | **Z** | **-** | «'» *("apostrophe")*

digit → **0** | **1** | ... | **9**

The juxtaposition (« ») and ellipsis (**...** and **..***k***..**) operators have identical priorities, which are smaller than that of the disjunction operator («**|**»).

Tableau 2: TransSearch query syntax

Each query defines a (possibly empty) solution set consisting of *bilingual contexts* from the translation memory, which in fact correspond to couples produced by the alignment process. Thus a TransBase database can be seen as a set of bilingual contexts *c*, each consisting of a pair of token sequences t_A and t_B. Table 3 gives a recursive definition of the semantics of expressions produced according to the grammar of table 2. In the ''Expression'' column of this table, the variables *q*, q_1, and q_2 designate *queries*, *l* designates a *language*, *e*, e_1 and e_2 are *expressions*, *d* is a *descriptor*, and *f* is a *form*. In the ''Semantics'' column, the variable *c* designates a bilingual context, the indexed *t* variables are token sequences, *a* is an isolated token, and *i*, *j* and *k* are positive integers. *S(x)* denotes the set of solutions of the object *x*. In the case of a query *q*, this is a set of contexts (or couples) *c*, for an expression *e* it is a set of token sequences *t*, and for a descriptor *d* it is a set of tokens *a*. In the last line of the table, *Morpho(f)* designates the set of *inflected* forms of *f*: if *f* corresponds to one or more *citation forms* in a dictionary for language *l*, then *Morpho(f)* is the set of all morphological inflections of these citation forms.

Expression	**Semantics**
$q = l(e)$	$c \in S(q) \Leftrightarrow t_l \in S(e)$, where t_l is the *l*-language segment of *c*
$q = \sim l(e)$	$c \in S(q) \Leftrightarrow t_l \notin S(e)$, where t_l is the *l*-language segment of *c*
$q = q_1 q_2$	$c \in S(q) \Leftrightarrow c \in S(r_1)$ and $c \in S(r_2)$
$e = d$	$t \in S(e) \Leftrightarrow \exists k$ such that $t = t_1 ... t_k ... t_n$ and $t_k \in S(d)$
$e = e_1 e_2$	$t \in S(e) \Leftrightarrow \exists k$ such that $t = t_1 ... t_k t_{k+1} ... t_n$, $t_1^k \in S(e_1)$ and $t_{k+1}^n \in S(e_2)$
$e = e_1 ... e_2$	$t \in S(e) \Leftrightarrow \exists i, j$ such that $t = t_1 ... t_i ... t_j ... t_n$, $t_1^i \in S(e_1)$ and $t_j^n \in S(e_2)$
$e = e_1 ..k.. e_2$	$t \in S(e) \Leftrightarrow \exists i, j$ such that $t = t_1 ... t_i ... t_j ... t_n$, $t_1^i \in S(e_1)$, $t_j^n \in S(e_2)$ and $j - i < k$
$e = e_1 \mid e_2$	$t \in S(e) \Leftrightarrow t \in S(e_1)$ or $t \in S(e_2)$
$d = f$	$a \in S(d) \Leftrightarrow a = f$
$d = f+$	$a \in S(d) \Leftrightarrow a \in Morpho(f)$

Tableau 3: TransSearch query semantics

In practice, this language permits considerable flexibility in the formulation of queries. First, it allows searches for specific words in either of the two languages, or in both at once:

f(contre toute attente) e(against all odds)

will return all occurrences of the French expression *contre toute attente* which appear in the same bilingual context as the English expression *against all odds*. Negation (**~**) permits restrictions to be imposed:

f(adresse) e(address) ~f(postale)

will return the set of contexts where *adresse* and *address* appear together, except in the cases where *postale* appears in the French portion[4]. The ellipsis (**...**) and the restricted ellipsis (**..k..**) make it possible to search for expressions which are ‘‘scattered’’ in the text, notably those which admit an internal complement or modifier. For example

e(has ... to do with)

will find occurrences of *X has to do with Y*, *X has nothing to do with Y*, and *X has that got to do with Y*. This operator is also useful for abbreviating the specification of expressions in which two or three words determine all of the others, as is the case, for example, in *come hell or high water*:

e(hell..2..water)

The disjunction operator (**|**) allows all forms of a variable expression to be sought:

e(augurs | bodes well)

(Due to the higher precedence of the disjunction operator, it is not necessary to use parentheses around **augurs | bodes** in this expression.) Finally, the morphological expansion operator (affix **+**) spares the user from having to laboriously specify large disjunctions when the expression of interest contains words that are subject to inflectional variation. This is particularly useful for French, eg:

f(rendre+ justice)

will find not only all occurrences of *rendre justice*, but also the forms *rendons justice*, *rendu justice*, etc.

4.2 Query Resolution

The query language semantics given in the previous section suggests a recursive method for resolving queries. Figure 4 contains the essentials of a Prolog program to accomplish this[5]: a call to the predicate `resolve_query` returns the identifier of a couple which satisfies the given query, and by forcing backtracking we obtain the complete solution set.

This program accepts queries which have been previously transformed into corresponding tree structures, possibly by a Prolog program derived from the grammar presented in table 2. Such a transformation is not, however, limited to a simple parse of the query: it also includes the tokenization, normalization, and morphological expansion of the forms which make up the query. Tokenization and normalization are applied to all forms, and are exactly the same procedures used to analyze new text when it is being added to the database (see sections 3.1 and 3.2). Note that tokenization can involve splitting some forms into sequences of forms. Morphological expansion is

4. For practical reasons, the program requires that at least one of the members of a query be ‘‘positive’’, so in fact our negation operator is equivalent to the ‘‘AND-NOT’’ found in other query languages.

5. Although TransSearch’s query resolution modules are not written in Prolog, we felt it was appropriate to use this language to describe them, because Prolog’s inference engine incorporates mechanisms which are analogous to those used in the implementation of TransSearch

```
resolve_query([], SENSITIVITY, COUPLE) :-
    !,
    nonvar(COUPLE).

resolve_query(query(MEMBER, QUERY_REMAINDER), SENSITIVITY, COUPLE) :-
    (MEMBER = negation(member(LANGUAGE, EXPRESSION)),
     resolve_query(QUERY_REMAINDER, SENSITIVITY, COUPLE),
     nonvar(COUPLE),
     (resolve_expression(EXPRESSION, LANGUAGE, no_min, no_max, SENSITIVITY, _, COUPLE)
      -> fail
      ;  succeed)
    ;MEMBER = member(LANGUAGE, EXPRESSION),
     resolve_expression(EXPRESSION, LANGUAGE, no_min, no_max, SENSITIVITY, _, COUPLE),
     resolve_query(QUERY_REMAINDER, SENSITIVITY, COUPLE)).



resolve_expression(disjunction(EXPRESSION_1, EXPRESSION_2),
                   LANGUAGE, MIN, MAX, SENSITIVITY, LAST, COUPLE) :-
    (resolve_expression(EXPRESSION_1, LANGUAGE, MIN, MAX, SENSITIVITY, LAST, COUPLE)
    ;resolve_expression(EXPRESSION_2, LANGUAGE, MIN, MAX, SENSITIVITY, LAST, COUPLE)).

resolve_expression(conjunction(EXPRESSION_1, DISTANCE, EXPRESSION_2),
                   LANGUAGE, MIN_1, MAX_1, SENSITIVITY, LAST_2, COUPLE) :-
    resolve_expression(EXPRESSION_1, LANGUAGE, MIN_1, MAX_1, SENSITIVITY, LAST_1, COUPLE),
    MIN_2 is LAST_1 + 1,
    (DISTANCE = unspecified -> MAX2 = no_max ; MAX_2 is MIN_2 + DISTANCE),
    resolve_expression(EXPRESSION_2, LANGUAGE, MIN_2, MAX_2, SENSITIVITY, LAST_2, COUPLE)).

resolve_expression(form(FORM, CASE_PATTERN), LANGUAGE, MIN, MAX, SENSITIVITY, LAST, COUPLE) :-
    transbase(LANGUAGE, ATOM, FORM, COUPLE, CASE)
    (SENSITIVITY = insensitive -> succeed ; CASE = CASE_PATTERN),
    (MIN = no_min -> succeed ; ATOM >= MIN),
    (MAX = no_max -> succeed ; ATOM =< MAX),
    LAST = ATOM.
```

Figure 4: Prolog program for TransSearch query resolution

applied only to forms marked with a "**+**" operator, and consists in replacing them with a disjunction of inflected forms supplied by a morphological dictionary.

In this simplified implementation, we have combined into a single predicate all the information from a TransBase database that is required to resolve queries:

```
transbase(?LANGUAGE, ?TOKEN, ?FORM, ?COUPLE, ?CASE_PATTERN).
```

This predicate has one clause for each token in the database: the `LANGUAGE` argument determines whether the token is an entity of type *TokenA* or *TokenB*. The `TOKEN` argument corresponds to the *Position* attribute, and `CASE_PATTERN` corresponds to the *CasePattern* attribute. The `FORM` argument takes the value of the *Text* attribute of the *FormA* or *FormB* entity linked to the token via the *Occurrence* relation, and `COUPLE` takes the value of the *Position* attribute of the *Couple* entity linked to the token via the *SegmentA* or *SegmentB* relation.

TransSearch uses a search algorithm which, although based on the program in Figure 4, adds two principal optimizations. First, searches in the database are constrained to relevant zones: as mentioned in section 3.2, the implementation of the *Occurrence* relation permits the use of `MIN` and `MAX`

variables to confine the search to tokens having positions within these limits. Second, we estimate the maximum *yield* of each element in a conjuction from the frequencies (via the *Frequency* attribute) of the forms it contains; by processing low-yield elements first, we minimize the amount of backtracking necessary to resolve the conjunction.

5 Graphic Interface

TransSearch is designed primarily for translators, terminologists, and lexicographers, whom we cannot necessarily assume to be computer wizards. Our desire to provide a user-friendly tool necessitated, therefore, the development of a graphic interface[6]. For practical reasons, we constructed our TransSearch prototype on Sun workstations, programming in C under UNIX. Given this platform, natural choices for a window system and API were Sun's OpenWindows and XView.

The interface consists of two main windows: a display window, which comes up when TransSearch is first invoked; and a query window. These can be seen as answers to the two principal questions posed by the problem of user-friendliness in the context of bilingual concordances: ''How should queries be formulated?'', and ''How should the results be displayed?''. (See Figure 5).

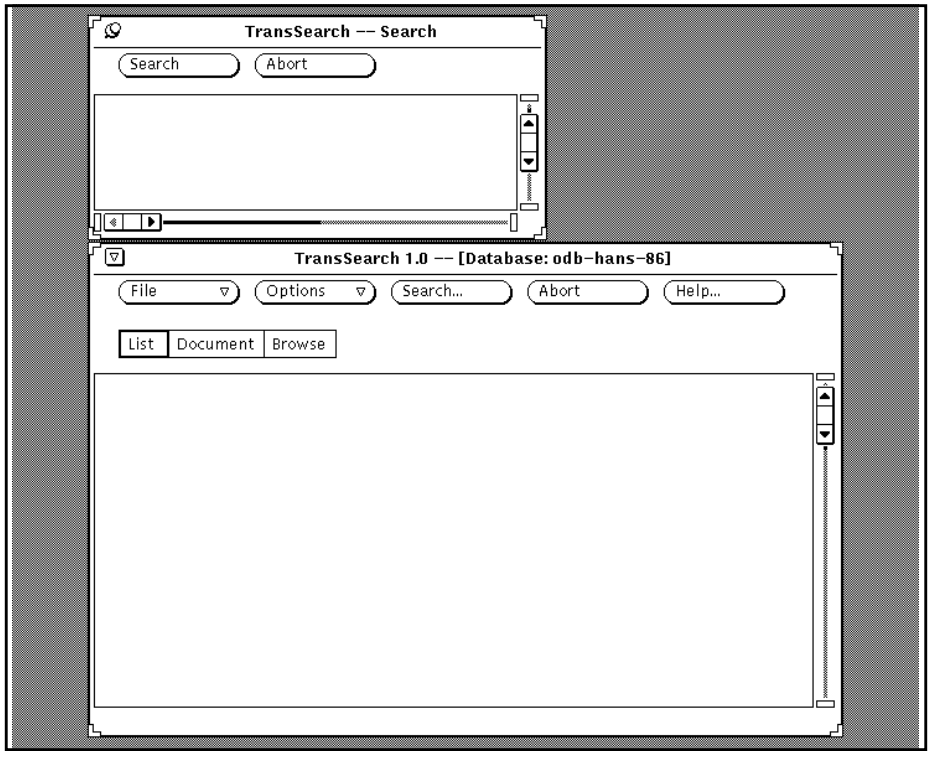

Figure 5: TransSearch's graphic interface

6. A command-line interface has also been developed for internal use by the TAO group. We describe only the graphic version here, but the two provide essentially the same functions.

The first of these questions led us to conceive an intuitive graphic language to represent TransSearch queries, and to develop a simple method of formulating queries in that language. The query window, which appears when the user selects the *Search* button in the display window, is thus essentially a *graphical query editor*. Such an editor has the advantage of almost completely eliminating problems due to syntax errors.

In our graphic language, a query appears as a vertical list of expressions to be satisfied, each of which is preceded by a label that indicates the language of the expression, and possibly by a symbol that indicates a negation. Simple expressions---sequences of contiguous forms---are represented by horizontal lists of forms separated by spaces (ie, as in ordinary text), and forms destined for morphological expansion are underlined. An ellipsis between two forms appears as a broken dash (for an unrestricted ellipsis), or as a dash with a number superimposed (for a restricted ellipsis). Finally, the elements in a disjunction are displayed as a vertical list linked by lines. Figure 6 illustrates the graphic representation of the query "**f(arriver+ ... (à temps) | pile) e(arrive+ ..5.. on | in time)**".

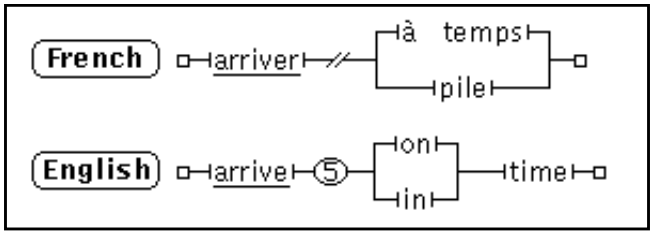

Figure 6: Representation of a query in TransSearch's graphic interface

The query editor can be used with both the mouse and the keyboard: all navigation within a query can be accomplished with the mouse, and all editing commands are accessible via an edit menu. Only for entering or modifying the text of an individual form is the keyboard required. On the other hand, it is possible to perform all editing functions and submit a query without using the mouse.

When a query is launched, the program begins by clearing the display window's central panel. The user can thus see the results as they are produced, and, in addition, can stop the search at any time by hitting the *Abort* button.

For representing the results, it is obvious that the ''classic'' concordance style – one line per result, with occurrences aligned down the centre of the display – would be inappropriate. First of all, it is difficult to reconcile this format with the complex expressions permitted by our query language. Second, it is ill-suited to the display of bilingual context. Finally, it seems clear that this style is a relic from the era when only paper concordances were available. (Those who have seen the size of traditional concordances will realize that expanding the context displayed with each occurrence would have necessitated supplying their users with wheelbarrows.) We have created two distinct display modes which we feel are better adapted to users' needs.

The *list* mode most resembles the classic format: all retrieved couples are displayed, one below the other, with horizontal lines between them. The text of each couple is printed in two columns, one for each language, with an arrow between the columns to indicate translation *direction* (pointing from the source language to the target language). The words ‘‘hit’’ by the query are displayed in bold characters[7] (Figure 7). The user can navigate over the list either via the scrollbar or by keyboard commands which focus on a particular result, which we call the *current* result. Additional information on the current result appears above the central display panel: document name, couple identifier, etc.

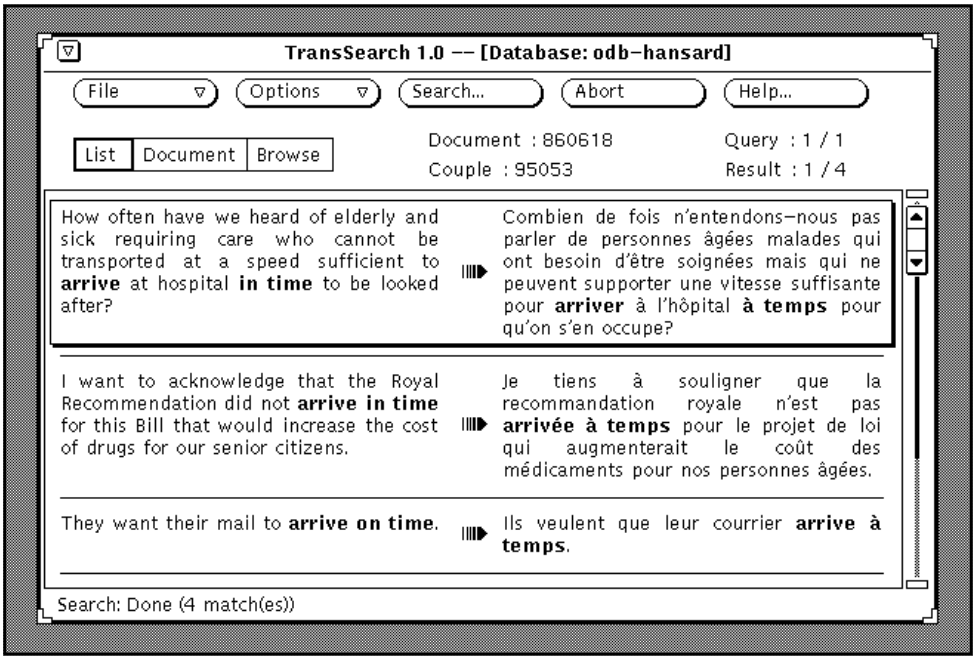

Figure 7: Display of a bilingual concordance in TransSearch’s display window

When the user selects *document* mode, the display panel is dedicated entirely to the current result, and contains the whole document to which it belongs. The couples which form the document are again displayed vertically in two columns. The current result is boxed to distinguish it from the others, and the tokens involved in the query are bolded. The user must employ the keyboard to go from one result to another in this mode, because the scrollbar permits movement only within the current document.

7.Here it is interesting to note that the way TransSearch interprets a query does not necessarily conform to a “naive” user’s intuition: while TransSearch looks for couples which satisfy a query, the user expects occurrences of a particular expression. For this reason, when a couple is found, TransSearch calculates and highlights *all* combinations of forms which satisfy the query within the couple, so as to better match the user’s expectation.

Although the interface handles a single query at a time, all queries which have been submitted during a session, and their results, are retained so that the user can quickly access them.

The interface also has a *browse* mode in which the contents of any document in the database can be viewed simply by specifying its name: no query is necessary. Finally, the interface can display either language exclusively, so it can be used as an ‘‘ordinary’’ concordance if desired.

Conclusions

The TransSearch UNIX prototype is now in its second version, and is in daily use at CITI by the linguists and computer scientists of the TAO group. It has been the object of several demonstrations, notably in Ottawa (ATIO 93), Paris (First AUPELF-UREF Linguistic Engineering Conference), and Japan (MT Summit 93), and each time the reactions of experts and potential users have been extremely positive. An operational trial in a translation service is planned for fall 93. These activities notwithstanding, we currently envisage several technical improvements to the system in the short and medium terms.

First, it is clear that most potential TransSearch users do not have access to Sun workstations, nor are they likely to in the near future. It is therefore necessary to port the system to a more common platform (PC or Mac).

The linguistic analysis of texts in the translation memory is currently quite rudimentary, and all indications are that there is much to be gained by making it more extensive. In particular, it seems that a morpho-syntactic word tagging would permit better filtering of ‘‘noise’’ from the solution sets, for example by eliminating occurrences of the verb *lead* from a search for the metal. Tagging also makes possible a more refined tokenization, and thus more intelligent query resolution which is capable of finding solutions currently hidden by peculiarities of the language. For example, splitting the form *des* into *de* + *les* would allow the expression *en dépit des apparences* to be correctly identified as an occurrence of *en dépit de*. Splitting *Mary’s* into *Mary* + *is* would allow *Mary’s blue* to be correctly identified as an occurrence of *X is blue*. Such tokenization is only possible, however, if we are capable of resolving the lexical ambiguities involved, such as determining that *des* is not an article, and that *Mary’s* is not a possessive form. To accomplish this, we plan to use lexical analyzers based on statistical language models such as that of Foster ([5]).

The suggestion which surfaces most often in consultations with users is that, for a query which contains only expressions in language ‘‘A’’, the words in language ‘‘B’’ which constitute the translation of the query should be highlighted. For example, the user who seeks all occurrences of the expression *to be out to lunch* would like to see the exact translation of this expression (*se met le doigt dans l’oeil* here, *est dans la choucroutte* there, etc) displayed in bold characters in the French portion of the retrieved contexts. This capacity requires finer correspondences than those on which Trans-

Search in currently based. Algorithms for word-to-word correspondence exist (see, eg [4] and [8]), but remain to be evaluated and integrated into the system.

Finally, while our translation memory structure resolves some problems for bilingual translators, it is not at present adapted for *multilingual* translations. This problem is very interesting, because it appears for one thing that the existence of multiple versions of a document should permit better correspondences to be established, and for another that multilingual correspondences would contain translation relations between pairs of languages for which terminological references are now lacking. This question is currently under preliminary investigation at CITI.

References

[1] Aho, Alfred V., John E. Hopcroft and Jeffrey D. Ullman. *Data Structures and Algorithms*, Addison-Wesley, 1982.

[2] Berry, Mark. "The Trados Translator's Workbench II". *Proceedings of the 33rd Annual Conference of the American Translators Association*, San Diego, California, November 1992.

[3] Church, Kenneth Ward et William A. Gale. "Concordances for Parallel Texts". *Proceedings of the 7th Annual Conference of the UW Centre for the NOED and Text Research*, Oxford, 1991.

[4] Dagan, Ido, Kenneth Ward Church and William A. Gale. "Robust Bilingual Word Alignment for Machine Aided Translation". *Proceedings of the Workshop on Very Large Corpora: Academic and Industrial Perspectives*, Ohio State University, Columbus, Ohio, July 1993.

[5] Foster, George F. "Statistical Lexical Disambiguation". Master's thesis, McGill University, School of Computer Science, Montréal, Canada, 1991.

[6] Gale, William and Kenneth Ward Church. "A Program for Aligning Sentences in Bilingual Corpora". *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL-91)*, Berkeley, California, June 1991.

[7] Isabelle, Pierre, et al. "Translation Analysis and Translation Automation". *Proceedings of the Fifth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-93)*, Kyoto, Japan, July 1993.

[8] Kay, Martin et Martin Röscheisen. "Text-Translation Alignment". *Computational Linguistics*, Vol. 19, No. 1, 1993.

[9] Macklovitch, Elliott. "Le PTT, ou les aides à la traduction". *La traductique : études et recherches de traduction par ordinateur*, Pierrette Bouillon et André Clas (éditeurs), Les Presses de l'Université de Montréal, 1993.

[10] Macklovitch, Elliott. “Corpus-based Tools for Translators”. *Proceedings of the 33rd Annual Conference of the American Translators Association*, San Diego, California, November 1992.

[11] Simard, Michel, George Foster and Pierre Isabelle. “Using Cognates to Align Sentences in Bilingual Corpora”. *Fourth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-92)*, Montréal, Canada, June 1992.

Acknowledgements

We would particularly like to thank Pierre Plamondon and Nicolas Viau for their invaluable contributions to the implementation of TransSearch, and Pierre Isabelle for his patience and continued support. We are also indebted to Elliott Maklovitch, Marc Dymetman, Marie-Louise Hannan, Jean-Marc Jutras, Xiaobo Ren, Benoît Robichaud, and Caroline Viel for their valuable advice during the evolution of multiple prototypes.