Linguistic tools are incidental to work in machine translation

by Russell Bateman

Russell Bateman is a computational linguist working on English-French development for Automated Language Processing Systems

During the development of natural language processing software, especially computer translation systems or computer aids to translation (MT, MAT, etc.), tools are created and research performed which are often wasted when commercial companies die or academic research projects break up. What is the nature of the tools which are developed and just what are we losing when the projects die with no more legacy than the spreading of good minds and ideas to the four winds?

I should like first to examine the evolution of development environments and me various vehicles which developers have originated to further MT. This is somewhat cursory and amounts to a fairly loose description of what I have encountered in my association with the MT world.

General Purpose Programming Languages

In the early days, the computer languages available to the developer were limited in scope. They had extremely poor string handling capability and were not the least bit suited to expressing linguistic algorithms involving parsing or pattern matching. Among these languages were various machine languages (early SYSTRAN code was written in assembly) and FORTRAN. Severe problems remained even after code was written in that the code was not easily maintained nor was it a simple matter to initiate a newcomer to developing in such an environment.

As early as 1960, formal specifications arose for general purpose programming languages which permitted superior implementation of algorithms through devices such as block structuring and modularity. Throughout the '60s and '70s languages like ALGOL, PL/I, PASCAL, and 'C' were born but with the exception of PL/I and 'C', they still lacked formally defined string handling. At Brigham Young University, the Translation Sciences Institute (TSI) as it was to be called, began to tackle machine translation and soon chose PL/I, primarily because they were using IBM hardware but also for its relatively superior algorithmic ability and its string handling. PL/I furnished a built-in data structure for strings but manipulation was still mostly in the hands of the programmer who was required to write libraries of routines for this.

The 'C' programming language has become an excellent system language because of its efficiency and portability, but it is nevertheless limited in the same way as PL/I and other 'structured' programming languages in that it has almost no built-in string manipulation functions at all. Furthermore, these languages do not allow linguistic algorithms, such as parsing of morphological analysis, to be written in concise and intuitively obvious ways. See Figure 1.

Special purpose languages

In the early seventies, new languages like LISP and Prolog surfaced, with a markedly superior ability in linguistic expression, particularly in expressing and manipulating semantic relationships, which was recognised early on as perhaps the greatest nightmare to overcome in machine translation. The Artificial Intelligence (AI) community has been most active in their use. Few commercial MT concerns have begun to use them. This is, however, understandable for three reasons. First, their initial availability on systems was quite limited and still is in any standard or portable form. Second, few offer native language compilation on any machine thus being difficult to integrate with existing programs which is often a requirement in the commercial setting. Companies like Automated Language Processing Systems (ALPS) rely upon a real-time environment for their translator aids and cannot accept the relative slowness of interpreters, particularly on the existing general-purpose non-LISP hardware. Last, truly integrated environments where one finds the capability of calling special purpose language functions are scarce or non-existent. Fully compiled versions of these languages cannot, by definition, have all the functionality of the interpreted versions. In any case, LISP and Prolog may be well suited to solving many linguistic or AI problems, but their use as systems programming languages is still not widely accepted nor possible (at least not on a wide enough range of hardware). Realistically, a commercial MT/MAT company must provide an entire translator environment, geared to helping the translator translate rather than frustrating him (1) and in doing such, must integrate a great deal of systems software like word processors and control environments which cannot be done practically in the current array of AI programming languages, because, as we said, of the relative unacceptability of systems programming in these languages. We look forward to the day when LISP machines are widely available and at reasonable prices. See Figure 2.

Comparing general purpose programming languages like ALGOL and 'C' with LISP and Prolog in our context, however, is quite pointless - like comparing apples and oranges. To create a good translation environment, one needs document production facilities foremost and that implies systems and general purpose programming languages for the reasons already established. To meet the needs of linguistic expression - which is worlds apart - special purpose linguistic languages are necessary, and most companies come to that realisation sooner or later investing a great deal of time and talent in this area.

Linguistic support software or 'Lingware'

All projects have used some system supported programming language or another but innovators on MT projects like those of SYSTRAN, the Translation Sciences Institute, Weidner Communications Inc. and ALPS have designed and implemented what one could call 'lingware' with the goal of permitting near direct expression of linguistic algorithms: structural analysis, syntactic and semantic transfer and inflection. This lingware had the benefit of being quite maintainable; that is, the algorithms they expressed were easily corrected and improved.

Debugging and ISAMs

Putting together a translation system is a big affair and the software becomes very large and complex. Debugging becomes a concern and is an indispensable tool to the linguist and programmer alike. It is almost always impossible to accurately diagnose a problem in a system based on the translated output. Writing debuggers of some importance is something that each project tackles sooner or later whether or not they first think their system-supplied debugger is adequate. An important part of creating linguistic support tools like a formalism for syntactic analysis is providing a means of interpreting and debugging their output before it becomes corrupted by another phase of the translation code.

It is also essential that an MT project have an ISAM - dictionary lookup capability, and if the development system they have chosen does not offer a suitable one, they will be under the obligation of writing one - a non-trivial endeavour. Their ISAM (or often ISAMs) must provide fast and accurate access to lexicons as well as any tables they may use for grammar, inflection and the like.

Simple morphological and synthesis tools

Occasioned by every MT effort are the essential building blocks which must be in place before one can begin the more academically satisfying and more talked-about stage of parsing. By this I am referring to programs that break up and define sentences and words and reduce words morphologically in order to identify and by look-up obtain the necessary syntactic and semantic information used by the parser. In English this often occurs in the form of a reasonably simple algorithm but in the case of other languages, extensive tables and character matching functions are designed.

On the other hand, of course, are the tools with which the target language is produced. We call this synthesis and it entails a host of useful programs to inflect nominal forms, conjugate verbs, determine capitalisation, etc., any of which could find their application in assorted CAI (Computer Aided Instruction), writers' workbench package and the like.

Research aids, dictionaries and grammars

There are differing requirements between the needs of the machine in language processing and the needs of the intelligent human dictionary user. One example of this is the reference work *L'art de conjuqer, dictionnaire des 12,000 verbes.* The 'Bescherelle' as it is commonly called has been the de facto, popular French verb conjugation bible for a very long time. It treats some eighty conjugation types of the French language in a way that anyone of reasonable intelligence can understand. The problem here is that the computer is not capable of the same 'reasonable intelligence', and so each MT project has had to reorganise the tables.

Another example - because I work mostly in French - is the Larousse *Dictionnaire des verbes,* a rich work full of simple, straightforward research on verb valency (the verb plus expected, possible complements) with examples galore. And yet the work was approached from a more traditional grammar standpoint in both the terminology it uses and the categorisations. I have personally adapted much of its codings in my work according to hit-and-miss, practical requirements imposed upon me by the necessity I have to 'get the best out for the current deadline'. There is a dearth of such reference material in languages other than French and German; MT researchers therefore have to research and codify their own. We have linguists at ALPS who have done this sort of thing two or three times as they went from project to project.

Allow me to quickly add to this list of references most often created and exploited by MT researchers, the backward or reversed dictionary and the text corpus. Brown University's corpus has served almost everyone since its tagging was finished in the early seventies, but other languages are not so fortunate. Corpora, let alone tagged corpora, are difficult to obtain in French and other Romance languages, even though some do exist in academic environments. Many MT companies must resort to compiling their own reverse dictionaries (essential to the establishment of morphological tables) and corpora (used for statistical and contextual analyses of words).

Summary

To recap the products which are created incidental to work on every MT project, allow me to re-enumerate specifically: 1) String handling functions for the programming languages used; 2) Parsers and/or special purpose programming languages for expressing grammatical formalisms developed by linguists; 3) Other tools often involving compilers and interpreters for performing the steps in the translation process such as morphological reduction, ordering, inflection/conjugation and capitalisation or other graphological adjustments of the output; 4) Debugging or diagnostic display packages created for use by the implementors; 5) ISAM capability as a basis for lexicons and tables if none is available or suitable on the development system chosen; 6) Compiled data from prepublished grammar research, corpus study, statistical lexicography, semantics research, etc.

Is it feasible to sell or otherwise distribute these materials?

Of course, that is the question that my superiors and board of directors would be most likely to ask! The problem of feasibility seems to lie in two principal areas: the possibility of producing a workable package to sell and the desire or willingness of the producing company to share its development with potential competitors.

The packaging of such information can take several of many traditional forms: publishing in the case of dictionaries and grammar research or installation in the case of producing actual software packages like parsers and verb conjugators.

The more obvious impediment to the proliferation of specific linguistic tools in the commercial let alone public domain is the understandable desire of any company which perceives its existence as depending on MT systems sales, to alone reap the benefits of its own R & D lingware and technological edge. To digress, I might state that this mentality prevails even when the tools they are currently developing and using are academically obsolete when compared with the latest as defined by the participants in conferences like CALICO, COLING and other ACL happenings, and the various conferences on AI; indeed, it is doubtful that any of the truly commercial MT companies are now employing any linguistic knowledge or techniques that are not at least five to ten years old. And it is also very doubtful that real AI is being used in any of the companies with actual products now on the market!

In view of the small number of MT companies in existence, the real

market for computational linguistic tools and information might Be the academic institution. Much information reaches the public domain through the conferences just mentioned and is not fully exploited by the MT and university communities. In addition, however, publishers of dictionaries should be interested as the increased computerisation of their industry, including their traditionally paper-medium products, will certainly overturn much of what has been compiled over the centuries. This, of course, applies dramatically to traditional schoolboy grammars but can also find its application in age-old authorities such as Bescherelle and Grevisse's *Le Bon Usage.*

To a large extent, the public domain would be benefitted by the efforts of MT researchers, past and present, particularly in two immediate areas: dictionaries and grammars.

France, for example, is currently in a great period of informatisation or computerisation to place a world of information in the form of on-line reference materials, shopping and banking services only as far away from each citizen as his or her telephone and television screen. Prototype systems have already been installed in various French cities. Soon, I believe, the reliance upon manual, intelligent methods will be upstaged by the arrival of automatic unintelligent ones and the publishers of grammars and dictionaries will be urged by software developers of these systems to modify their formats because the computer cannot operate on the implicit information.

In the area of grammatical theory and research, MT excels as a proving ground. To a great extent, the old schoolboy grammars have been shown to be inadequate by attempts made to apply them on the machine. It is true that present machine applications can be unfair, especially in light of difficult semantic considerations, but coding any grammar's rules in a machine can be very instructive as the BYU-TSI project found out during the decade of its work with Junction Grammar.

Conclusions

In conclusion, I have shown that a great variety of useful by-products in actual tools and important research are created 'from scratch' each time an MT project is launched. I believe that we in the MT world are doing a disservice to the general advancement of MT by not examining possible and (in the case of private ventures) commercially harmless, outlets for the mass of knowledge gained on each project.

Notes

1. A discussion of just what environments constitute aids to translation rather than frustration is found in Bateman, R., "Introduction to Interactive Translation" in the proceedings of the November 1983 ASLIB conference, London, England.

2. The grammar was discussed by Grischman, R. and Ngo Thanh Nhan, in a presentation entitled "Automated Determination of Sublanguage Syntactic Usage" and published in the proceedings of the July 1984 COLING conference held at Stanford University, California.


```
#include    <mytypes.h>
#include    <parsingtypes.h>

#define     TRUE                    1
#define     FALSE                   0
#define     ParseTreeNodeSize       sizeof(struct ParseTree)

extern struct ParseTree *BegSeg, *EndSeg;

void build_npn()
{
    char            *malloc();
    boolean         building_noun_phrase = FALSE,
                    first_node = TRUE;
    struct ParseTree  *lay, *new_np, *last_brother;

/*
 * Beginning at right of sentence, parse for simple noun phrases.
 */
    node = EndSeg;
    do
    {
        node = node->left;
        if ((not building_noun_phrase) && (node->cat == NOUN))
        {
            building_noun_phrase = TRUE;
            new_np = (struct ParseTree *)malloc(ParseTreeNodeSize);
            new_np->id = NOUN;
            new_np->son = node;
            node->father = new_np;
            last_brother = node;
        }
        else if (building_noun_phrase)
            switch (node->cat)
            {
                case DET :
                case ADV :
                case ADJ :
                case NOUN :
                    node->father = new_np;
                    node->brother = last_brother;
                    last_brother = node;
                    break;
                default :
                    building_noun_phrase = false;
            }
    }
    while (node != BegSeg);
}
```

Figure 1: "A Simple Noun Phrase Rule Fragment in 'C'"

```
sentence( s(NP,VP) ) -->  noun_phrase(NP), verb_phrase(VP).

noun_phrase( np(Det,Noun,Rel) ) -->
     determiner(Det), noun(Noun), rel_clause(Rel).
noun_phrase( np(Proper_noun) ) --> proper_noun(Proper_noun).

verb_phrase( vp(TV,NP) ) -->
     trans_verb(TV), noun_phrase(NP).
verb_phrase( vp(IV) ) --> intrans_verb(IV).

rel_clause( rel(that,VP) ) --> [that], verb_phrase(VP).
rel_clause( rel(nil) ) --> [].

determiner( d(a) ) --> [a].
determiner( det(every) ) --> [every].

noun( n(man) ) --> [man].
noun( n(woman) ) --> [woman].

proper_noun( name(john) ) --> [john].
proper_noun( name(mary) ) --> [mary].

trans_verb( tv(loves) ) --> [loves].
intrans_verb( iv(lives) ) --> [lives].
```

Figure 2: "Fragment of a Grammar in Prolog"

```
Simple Noun Phrase Rules:

    AJP  ->  {ADV}   ADJ.
    AJP  ->   AJP    ADJ.

    NOM  ->   NOUN   NOUN.
    NOM{1, 2=NOUN} ->   NOUN   NAJ.
    NOM  ->   NOUN.

    DET  ->  ?UNNY_WORD(all,both,just,quite,such)  DET.

    NP(features)  ->   {DET}   {AJP}   {NAJ}   NOM(features).
```

Figure 3: "A Rule Fragment in a Linguware Formalism"