The FINITE STRING Newsletter

Site Report

Controlling Complex Systems of Linguistic Rules

Rod Johnson
University of Manchester Institute of Science and Technology, U.K.
Steven Krauwer
Rijksuniversiteit, Utrecht, Holland
Mike Rosner
Fondazione dalle Molle, ISSCO, University of Geneva, Switzerland
Nino Varile
Commission of the European Communities, Luxembourg

[Most of the ideas in this paper evolved during work on design studies for the EEC Machine Translation Project Eurotra. We are grateful to the Commission of the European Communities for permission to publish these ideas, and to our Eurotra colleagues – particularly Maghi King, Dieter Maas, Bente Maegaard, and Serge Perschke – for many useful contributions, which have been influential in moulding our current thinking.]

Introduction

In this paper we discuss some of the design decisions we have made in defining a software environment for a large scale Machine Translation (MT) project. A general purpose MT system intended for commercial application should ideally have many features, such as robustness and transparency, in common with any large industrial software implementation. At the same time, consideration of the way in which a good MT system is built and maintained suggests an approach more characteristic of AI programs and Expert Systems (ES) in particular. There may be conflicts between the tight style of top-down design and implementation advocated by designers of conventional industrial software and the rather empirical, heuristic style of development typical of more loosely structured knowledge-based systems. Our suggested solution to these conflicts involves an enhanced form of controlled Production System (PS), which combines maximal transparency and modularity with the advantages of the characteristically declarative and locally unstructured organisation of the typical PS architecture. Although our ideas derive originally from our current preoccupation with MT system design, the general principles we have adopted should be equally valid for the construction of any large language processing system.

Production Systems

The advantages and disadvantages of the PS style of programming are well known – a good review is Davis and King (1977), although they say relatively little about the use of PS for linguistic problems. The PS architecture is particularly suited to knowledge-based systems which depend on having access to large amounts of relatively homogeneous, factual knowledge. It is also easy, in principle, to add to and subtract from the knowledge base since factual knowledge is intended to be decoupled from procedural application. In this regard, a PS has obvious attractions for applications in MT.

A typical PS used to represent linguistic computations might have the following organisation: the data base would be some collection of tree structures; the rules would consist just of a pair of structural descriptions; and the interpreter would repeatedly match the left-hand sides of rules against the data base, building right-hand structures every time there is a successful match. The paradigm example of a PS in MT is probably Colmerauer's (1970) Q-system, in which the TAUM METEO system is written (Chandioux 1976).

In theory, the PS style of programming looks very attractive for MT. In practice, however, as a PS becomes large it becomes increasingly difficult to control. Supposedly independent rules begin to interact in unforeseen ways, often with obscure consequences. When it becomes necessary to modify the behaviour of the interpreter – as inevitably happens – users are forced to introduce the necessary control information into rules. Because all communication between rules takes place through the data base, rules become complicated by extra tests on and assignments to arbitrary flags which have no meaning for anyone but the user responsible for their introduction, but which, once created, survive permanently in the data base. In the end, the PS becomes even more complex and obfuscatory than the corresponding procedural program it was intended to replace.

For the purposes of MT, these observations are particularly disturbing. General purpose MT systems are *de facto* large, and thus particularly prone to the dangers we have just described. Nor is there any reason to suppose that a single interpretation scheme will be appropriate for all the tasks necessary for MT – string manipulation, phrase structure analysis, arbitrary tree transductions, dictionary lookup, plausibility weighting of conflicting analyses, and so on. After all, we want to offer users flexibility and naturalness of expression,

and these features are not really consistent with a single way of doing things.

Thus, although we see the appeal of a PS architecture for MT, we have to concede that it not only fails to satisfy the requirements of any conventional industrial software, it does not even provide users with the flexibility they need to solve the kinds of ill-defined, open-ended problems that habitually arise in MT. In the rest of this paper, we describe a solution that maintains the declarative, empirical style, characteristic of a PS, within a framework conducive to the top-down, modular construction of robust systems.

Homogeneity

The first requirement we had, especially given the peculiar context in which we are working, was to define a degree of homogeneity over the whole system. So as not to conflict with the equally important criterion of experimental flexibility, this homogeneity is limited to rather superficial aspects of the system design. Thus we have imposed a uniform rule syntax, such that any interpreter in the system must be defined to operate with that syntax or a subset of it. Similarly, we have constrained the class of structures that rules can be written to manipulate. The nature of these constraints – which are in fact less restrictive than they seem – will be discussed elsewhere (Johnson, Krauwer, Rosner and Varile, in preparation), and we shall not discuss them further here.

Enhancing PS Control Facilities

The poverty of the control structure of a typical PS is evident if we consider a chain of Q-systems, representing the interpreter as a rule-applying automaton defined by the regular expressions

(1) $Q = P_1,P_2,...,P_n$

(2) $P_i = (r_{i1} \mid r_{i2} \mid ... \mid r_{ij} \mid ...)^* \qquad (j = 1,2,...,n_i)$

where the P_i are the individual Q-systems and r_{ij} is the jth rule of the ith Q-system. The interpretation of the regular expressions is as follows:

(1) Execute Q by executing the P_i in sequential order.
(2) Execute each P_i by iteratively applying all its rules in parallel until no applicable rule can be found.

Expressed in this way, the available control strategies become clear: we can apply packets of rules in sequence, and, within a rule packet, we can apply rules iteratively in parallel. Nothing else is possible.

There is, however, no reason why we should not be able to generalise these three basic control notions of sequential, parallel, and iterated application to produce much richer and more interesting control strategies. The idea of a regular control language, which we have adopted in our design, is similar in spirit to the general scheme developed by Georgeff (1979, 1982) for characterising PS control.

The basic control construct in our model is a *process,* which may either be simple (composed only of rules) or complex (constructed out of other processes). Another name for a simple process, in our terminology, is a *grammar.* A complex process is defined by writing down a regular expression over the names of other processes in the system, for example

P = A,B,C
Q = X | Y | Z
R = (P | Q)*

This simple generalisation gives us a far more powerful range of strategic options than does a simple PS like the Q-system. However, it still leaves a number of important open questions, especially about what goes on inside a grammar and what data processes are actually applied to. We address these questions in the next two sections.

Limiting Side Effects

One of the most serious problems with a large PS is the impossibility of predicting what information will actually be present in the data base at any given time. It is this, more than any other aspect of PS design, that causes rule-writers to include in their rules all kinds of extraneous tests simply to avoid spurious rule application in situations where the rule is not intended to apply.

Now, given the kind of control organisation described in the last section, we can observe that, when we define a process in terms of a collection of embedded processes or grammars, the only important aspects of the behaviour of the embedded processes are the kinds of structure they accept as input and the kinds of structure they produce as output. To achieve the effect we need we introduce the notion of a *filter.* Every process or grammar has associated with it a pair of filters, which are syntactically just structural descriptions like the lhs of a rule. The input filter, or *expectation* of the process, is used to supply to the process just those structures that can be successfully matched: if nothing in the data base matches the expectation, the process is simply not invoked. When the process terminates, the output filter, or *goal* of the processes, allows to pass to the calling process just those structures built by the process that match the goal.

The effect of the introduction of filters into a controlled PS is quite dramatic. Side effects may appear but never survive the process in which they are created. It becomes possible to test modules in isolation, simply by simulating data base states on which they are supposed to operate. In the same way, processes can be designed top down, with fairly strong guaran-

tees that each process will do what it was supposed to do, provided its component parts deliver what they are supposed to deliver. Errors are easier to trace because the behaviour of a process can be unequivocally defined.

If to this we add the important side benefit that processes automatically become self-documenting, it is apparent that with this device we have been able to capture almost all of the advantages of structured programming, without losing the essentially declarative spirit of the enterprise.

Modifying the Interpreter

As we have remarked above, the semantics of a rule in a PS depends ultimately on the characteristics of the interpreter that applies it. It is important that a general and flexible system would be able to accommodate a wide variety of different task-oriented interpretation schemes. Within the framework we have been developing, it should not be difficult to implement safely virtually any interpreter that satisfies the homogeneity criteria stated above. As far as neighbouring processes are concerned, the internal behaviour of a grammar is of no interest provided it operates on and produces well-formed structures. As it happens, we have so far only considered implementation of one parameter-driven interpreter which applies rules according to the same principles as those that govern the application of processes discussed above ('Enhancing PS Control Facilities'). It appears that this interpreter is likely to be adequate for a wide variety of structure-processing tasks in the immediate future. When new interpreters are required, we do not anticipate serious difficulties in integrating them into the system.

Conclusion

In designing a software environment to support MT systems – a kind of meta-MT-system – we have attempted to provide a basic, declarative, problem-oriented architecture that is readily accessible to potential users. Our guiding theme has been to try to separate factual from control information without losing the declarative essence of a good PS. By adding filters to a controlled PS, we have been able to incorporate most of the principles of structured programming into a declarative framework. The design principles adopted should lead relatively painlessly to the construction of robust, modular, and easily extendible MT systems, while retaining the desirable flexibility for loosely structured experimental construction characteristic of a PS.

References

Chandioux, J. 1976 METEO: Un Systéme Opèrationnel pour la Traduction Automatique Des Bulletins Mètèorologiques destinès au Grand Public. *META* 21: 127-133.

Colmerauer, A. 1970 Les Systèmes-Q. Internal publication #43. University of Montréal.

Davis, R. and King, J.J. 1977 An Overview of Production Systems. In Elcock and D. Michie, Eds., *Machine Intelligence 8*. Ellis Horwood: 300-332.

Georgeff, M. 1979 A Framework for Control in Production Systems. AI Memo #322. Stanford University.

Georgeff, M. 1982 Procedural Control in Production Systems. *Artificial Intelligence* 18: 175-201.

Johnson, R.L.; Krauwer, S.; Rosner, M.A.; and Varile, G.B. A Flexible Data Model for Linguistics Representation. (in preparation)